NJSZT
MUSZAKI ÉS TERMÉSZETTUDOMANYI EGYESÜLETEK SZOVETSÉGE
NEUMANN JÁNOS SZÁMÍTÓGÉPTUDOMÁNYI TÁRSASÁG
BUDAPEST VI., ANKER KÖZ 1 • • LEVÉLCIM: 1368 BUDAPEST PF. 240 • • TELEX: 22-5369 • • TELEFON: 229-870

PROGRAMOZÁSI RENDSZEREK '78
Konferencia

II. kötet

Szeged, 1978. november 8-10.

MTESZ
NEUMANN JÁNOS SZÁMITÓGÉPTUDOMÁNYI TÁRSASÁG
valamint az

MTA
MATEMATIKAI ÉS FIZIKAI OSZTÁLYA
által szervezett

PROGRAMOZÁSI RENDSZEREK '78
konferencia előadásai

II. kötet

SZEGED
1978. november 8-20.

Szerkesztette:

DÁVID GÁBOR

HAVASS MIKLÓS

TARTALOM

II. KÖTET

A STRUKTURÁLT PROGRAMOZÁS MÓDSZEREINEK ALKALMAZÁSA A SOFTWARE-FEJLESZTÉSBEN

Juhász Csongor
Államigazgatási Számitógépes Szolgálat

1. Bevezetés

1976. juliusában egy CalComp Model 563 Graph Plottert kapcsoltak a SZÁMOK IBM-37Ø-es számitógépéhez. Az IBM program product formájában egy nagyon primitiv /és mint később a kisérletek és az utasitás-szintü analizis során kiderült: hibás/ alap-software-t szállitott a device meghajtására, amely nem volt kompatibilis az ilyen plotterekhez Magyarországon elterjedt un. CalComp Basic Software-rel.

Ezért Rozsnyai Gáborral azt a feladatot kaptuk, hogy fejlesszünk ki CalComp-kompatibilis alap-software-t a plotterhez. Ez a következő komponensek kidolgozását jelentette:

PLOTS - Spooling-file nyitó
PLAREA - Rajz-terület definiáló
WINFIX - "Ablak" kijelölő
WINREL - "Ablak" eleresatő
SETMSG - Az effektiv rajzoláskor megjelenő konzol üzeneti blokk felvivő /a spooling-file-ra/
FACTOR - Centrális hasonlóság /nagyitás-kicsinyités/
OFFSET - Affin transzformáció
WHERE - Aktuális toll pozició visszajelző
HOWOFF - Affin transzformációs faktorok visszajelzője
PLOT - Toll mozgató /az egész rendszer magja/
SYMBOL - Karakter generátor
NUMBER - Lebegőpontos szám generátor
CHAR - FORTRAN-FORMAT-WRITE generátor
PLOT9999 - Program a spooling-file kirajzolására

Végül néhány grafikon rajzoló és segéd rutin /pl. kör-generátor, stb./

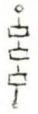
A munka elvégzésére 6 ember-hónap személyi és 12Ø számitott gépóra anyagi keretet kaptunk. Ez tekintettel

arra, hogy az elkészitendő termék kb. 8ØØØ assembler és 1ØØØ FORTRAN utasitásra volt becsülhető, és még a tervezési munkával sem voltunk egészen készen, igen feszitett és fegyelmezett munkastilust követelt tőlünk. Ugy döntöttünk, hogy a strukturált programozás módszereit hivjuk segitségül. Döntésünket igazolta, hogy bár a személyi keretet /mai tapasztalatommal indokolatlanul/ 2 ember-hónappal tulléptük, de 96 gépóra felhasználása után a munkát készre jelenthettük.

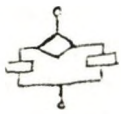
2. Implementációs szabályok

Bevezettük a következő un. kontroll strukturákat:

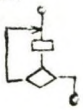
1/. Egyszerü utasitás sorozat: /begin...end/



2/. Döntés: /if...then...else.../



3/. Ismétlés: /do...while.../ ill. /for...do.../



4/. Eljárás hivás: /perform.../

/Az eljárás ill. szubrutin definicióját ld. később./

5/. Szubrutin hivás: /call... par1,par2,par3,... /

Látható, hogy a kontroll strukturák mindegyikének egy belépési pontja van - a tetején - és egy kilépési pontja - az alján. Kimondtuk, hogy a kontroll strukturák tetszőleges mélységben egymásba skatulyázhatók /azaz az előbbiekben közölt diagrammokon bármelyik ▭ vagy egyszerü utasitást, vagy kontroll strukturát jelent/.

Funkcionális egységnek a modult - egy önállóan forditható program egység - választottuk. A modulokon belül a következő logikai egységeket vezettük be:

1/. Szubrutin: /un. külső szubrutin/ a programrendszer egy önállóan teljes egészében egyszerre fejben tartható logikai egysége, amelyet hivással aktivizálunk és paraméter átadás-

sal kötünk össze a többi programrésszel /FORTRAN-CALL-hoz hasonlóan/. Korlátozott módon COMMON adat-modulok használatát is megengedtük /CSECT/. A szubrutinnak mindig egy belépési pontja van - a tetején - és egy kilépési pontja - az alján; és mindig a hivó utasitást közvetlenül követő utasitásra tér vissza. A szubrutinok kontroll strukturákból és eljárásokból épülnek fel.

2/. Eljárás: /un. belső szubrutin/ a szubrutin logikai egysége, amelyet hivással aktivizálunk /COBOL-PERFORM-hoz hasonlóan/. Nem tartalmaz paraméter átadási mechanizmust, mert a modul teljes területét cimezni tudja /IBM-Assembler sajátosság/. Egy belépési pontja van - a tetején - és egy kilépési pontja van - az alján; mindig a hivó utasitást közvetlenül követő utasitásra tér vissza. Kontroll strukturákból épül fel.

3. Kódolási konvenciók:

A "nevek" nem fantázia-nevek, hanem az illető változó, rutin, stb. funkciójára utalnak. Minden utasitáshoz tartozik komment. Minden modult, szubrutint, eljárást olyan fejléccel /több soros/ kell kezdeni, ami az illető program-egység teljes leirását tartalmazza. Tehát a kódnak "öndokumentálónak"kell lenni. A modulok listája legyen felülről-lefelé olvasható.

4. Implementációs eszköz

Az IBM DOS/VS Assembler Rel.32-höz létrehoztunk egy macro-rendszert, amelyet STRUCC=MACI -nak neveztünk el /Structurally Coded Macro Instructions/, és amely a kontroll strukturáinkat megvalósitja. /Magyar mnemonic használata mellett döntöttünk, hogy ne akadhassunk össze az esetleges későbbi IBM software fejlesztésekkel./ A macro-rendszer a következő elemekből áll:

```
Név   KEZDB
Cim   VÉGEB   Név      /Eljárás definiálás és hivás/
Cim   HIVB    Név
Név   KEZDK   (par.1,...,par.35),(partyp.1,...,partyp.35)
Cim   VEGEK   Név            /up to 35/
Cim   HIVK    Név,(par.1,...,par.35),SAVE=save area
     /Szubrutin definiálás és hivás/
```

Cim HA [Bal,Felt.,Jobb / ,HIVB,Név] [Y: / N:],(ut.1),...,[N: / Y:],...

/if...then...else.../

Ahol a Bal és Jobb helyén álló nevekhez a FORTRAN automatikus tipus deklarációjához hasonló név-kezdőbetü konvenciókat vezettünk be. Felt. helyén egy egyszerü feltétel kód áll /pl.: LTA - less than aritmetical, stb./. Bonyolult feltétel rendszer esetén un. feltétel kiértékelő eljárás hivható, amelynek a Név-4 cimre X'ØØ'-t kell tenni NEM és X'FF'-t IGEN esetben. A Y: ill. N: betük az igen-ágat ill. nem-ágat vezetik be. Az "ut.i" szimbólum - a megfelelő konvenciók szerint kódolt - assembler utasitást v. macro utasitást jelöl.

Cim ISMÉT (ut.1),...,(ut.35),K=,V=,N=,CVALT=cvalt.

/for...do.../ /up to 35/

A FORTRAN-DO utasitáshoz hasonló ciklus. Magja max.35 utasitásból állhat /ez nem korlátozás, mert "ut.i" HIVB is lehet/. A ciklus változónak külön szót kell lefoglalni.

Cim ADDIG (ut.1),...,(ut.35),F=[Bal,Felt.,Jobb / ,HIVB,Név]

/do...while.../ /up to 35/

A HA macrón keresztül megvalósuló ciklus.

Az IBM DOS/VS Assembler rekurziv hivást nem enged meg, tehát a kontroll strukturák tetszőleges mélységben való egymásba skatulyázhatóságát eljárás definiálással és hivással oldottuk meg. /KEZDB/VEGEB - HIVB utasitás zárójelezés/.

5. Implementációs tapasztalatok

Az implementáció során a modul struktura találkozott a szubrutin strukturával. A modulok mérete 1 printer lap körül mozgott /34 modulból csak 6-nak a mérete haladta meg az 1 lapot/. Az összes modul között csak egy van, amelynek mérete a 3ØØ utasitást meghaladja: a rendszer idő-kritikus részénél nem vállalhattuk a külsőszubrutin hivásból eredő plusz adminisztráció terhét, de ezt a modult is sikerült az eljárás definiciók segitségével jól strukturálni. Ezt mi sem bizonyitja jobban, mint az, hogy a 633 utasitásból álló modul belövése során 19 szintaktikai hibát kellett csak elháritani, szemantikus hibánk nem volt.

A következő statisztikai eredményeket értük el:
A Strucc-Maci rendszer 841 utasitásból áll, 27 szintaktikai hibánk volt, szemantikai hibát nem vétettünk. A rendszer létrehozásához 326 utasitásból álló segéd-macro-rendszert épitettünk fel, ebben 15 szintaktikai hibánk volt. A modulok 1768 Assembler utasitásból - 81 szintaktikai és 3 szemantikai hibával - és 72Ø FORTRAN utasitásból - 23 szintaktikai és 2 szemantikai hibával - állnak. Az elkövetett 5 szemantikai hiba között csak egy! volt olyan, amelynek kijavitása hosszabb időt - 2 napot - vett igénybe.

Ez tehát 8 ember-hónap alatt 3655 utasitás kódolását-belövését-dokumentálását jelentette, amelynek során a szintaktikai hibák aránya /146 db./ 4% alatt, a szemantikai hibák aránya /5 db./ Ø.14% alatt volt.

Az Assembler kódban a megfelelő macro használat több mint 4.2-szeres kompressziót jelentett. /Kifejtve az 1768 assembler utasitás 7425-öt tett ki./ A macro-rendszer bevezetése nélkül tehát 3655 utasitás helyett 8145 sor leirására lett volna szükség.

6. A dokumentálásról

A dokumentáció a rendszer implementálásával párhuzamosan az RIØ számitógépen futó Text Editor segitségével készült el. A Referencia Kézikönyv 3848 sorból áll. Egy kódsorhoz tehát kb. egy dokumentáció sor tartozik - leszámitva a Strucc-Maci dokumentációját, amelyet egy másik 32 oldalas SZÁMOK kiadvány tartalmaz.

Csak egyetlen dokumentáció készült, ugyanis a Referencia Kézikönyvből a Felhasználói kézikönyv /1441 sor/ és az Operátori Kézikönyv /23Ø sor/ egy-az-egybeh a szöveg változtatása nélkül kiollózható a Text Editor segitségével.

A Referencia Kézikönyv a következő fejezetekből áll:

1. Bevezetés /általános információk/
   - 1.1. A Plotter leirása
   - 1.2. Plotter parancsok
   - 1.3. Spooling /Hardware ismertető/
   - 1.4. Konvenciók
2. Felhasználói specifikáció
   /Az elvégzendő munka definiciója./

3. Globális tervezés

3.1. Modul struktura

3.2. A Modulok funkcionális követelményei

3.3. A Modulokban működő nem-triviális algoritmusok

3.3.1. Egyenes-közelitő algoritmus

3.3.2. A rajzolás menete

3.3.3. A Restart lehetőség biztositásának mechanizmusa

3.3.4. Tervezett JOB kontroll paraméterek.

3.3.5. Autolink konvenciók

3.4. A munka kiosztása személyekre.

4. Részletes tervezés

4.1. A PLAREA modul

4.1.1. Külső specifikáció

A. Azonositó:

Cim: SZÁMOK-PLAS rajzterület definiáló rutin.

Név: PLAREA /PLOTØØØ1/

Computer: IBM 37Ø/145

Op. Rendszer: DOS/VS Rel. 32.

Nyelv: IBM DOS/VS Assembler Rel. 32

B. Leirás:

C. Hivásáank módja:

D. Paraméterek:

Input:

Output:

E. Hivott szubrutinok:

F. Korlátozások

G. Függelék

4.1.2. Adat struktura

A. Belső:

B. Külső:

4.1.3. Algoritmus

4.1.4. Kapcsolódás a többi modulhoz

/Ugyan ez az összes többi modulról./

A PLOT9999 Spooling program Külső specifikációja a fentiektől eltér:

4.n.1. Külső specifikáció:

A. Azonositó:
   Cim: SZÁMOK-PLAS Spooling Program
   Név: PLOT9999
   Computer: IBM 37Ø/145
   Op.Rendszer: IBM DOS/VS Rel. 32

B. Általános leirás:

C. A program használata:
   1. JOB leirás.
   2. Vezérlő paraméterek
   3. Input adatok
   4. Output adatok

D. A felhasználás lehetőségeinek bővitése

E. Jellemzők:
   1. A program nyelve és szerkezete
   2. A program menete
   3. Alkalmazott módszer
   4. Korlátozások
   5. Irodalom

F. Szubrutinok használata

G. Memória és periféria igények

H. Kezelési utasitás

I. Speciális kezelési utasitás /Restart esetére/

J. Időbecslés

K. Hiba üzenetek

L. Függelék

Az algoritmusok leirását az ALGOL-hoz hasonló u.n. algoritmus leiró nyelven végeztük, amely pontosan fedte kontroll strukturáinkat. Igy az algoritmusok leirása a kóddal /a kódra irt kommentek segitségével/ pontosan fedésbe hozható.

Az azóta eltelt idő tapasztalatai megmutatták, hogy a fent leirt dokumentáció struktura - a megfelelő adaptálás után - más jellegü software termékek dokumentálására is felhasználható, és elegendő információt tartalmaz mind a felhasználói mind az üzemeltetési dokumentációk összeollózására, de a karbantartó softwares munkájának megkönnyitésére is.

PL/I ÉS COBOL DÖNTÉSI TÁBLA PROCESSZOR R-ES GÉPEKRE

Karlócai Miklós
Magyar Hiradástechnikai Egyesülés Számitástechnikai
és Szervezési Központja

Döntési táblákkal sokkal hatékonyabban fogalmazhatunk meg feladatokat, mint hagyomanyos folyamatábrakkal. A programkészités ideje felére - harmadára csökken. Áttekinthető, strukturált lesz a program, kevesebb, és könnyen javitható logikai hibával.
A döntési táblák felhasználásanak két feltétele van:

- a programozók ismerjék meg a döntési táblák készitését, használatát
- álljon rendelkezésre döntési tábla processzor, mely megérti és lefordítja a döntési táblakat

Az MHE SzSzK Számitastechnikai Osztálya 1973 óta foglalkozik döntési tábla processzorokkal. Az évek során 6 processzor készült el:

gép	op. rendszer	progr. nyelv
R 20-40	DOS	PL/I
R 20-40	DOS	COBOL
R 20-40	OS	PL/I
R 20-40	OS	COBOL
System-4	J 1800	PL/1
System-4	J 1800	COBOL

Mind a hat processzor korlátozott bejegyzésü ÉS/VAGY táblákat kezel a Thurner-elv alapján.

A processzorok inputja - a nyelvi különbségektől eltekintve - teljesen egységes, így jelenleg elég széles körben használhatók döntési tábla processzorok. Az R-es gépekre készült processzorokat a NOTO OSZV terjeszti, onnét egyszerüen megvehetők.

A döntési tábla

A döntési tábla algoritmusok megfogalmazására szolgál. Téglalap alakú tábla, egy függoleges és egy vizszintes vonal négy részre osztja. A bal oldalon állnak fent a feltételek, lent az akciók; a jobb oldalon a hozzajuk tartozó bejegyzések. A feltételrészben négyféle bejegyzés állhat:

Y ha a feltétel teljesülését kivánjuk

N ha a feltétel nem-teljesülését kivánjuk

\- vagy szóköz, ha a feltételt itt nem vizsgáljuk

Az akciórészben háromféle bejegyzés állhat:

X ha az akciót végre kell hajtani

\- vagy szóköz, ha az akciót nem kell végrehajtani

A bejegyzések adják a tulajdonképpeni algoritmust: azt rögzitik, hogy bizonyos feltételek teljesülése/nem-teljesülése esetén mi a teendő. Egy feltétel-együttállás és a hozzátartozó akcióvégrehajtási bejegyzés a táblázat jobb oldalán oszlopot alkot, ezt szabálynak nevezzük. A teljes feladatot több, egymás mellé irt szabály adja meg.

Nézzünk példát döntési táblára!

```
P L / 1   D E T A B   P R O C E S S O R   L I S T     ACNY
                                                              1
                                                    1 234 567 890
          DETAB-KERD!
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LATOTT MAR ON DONTESI TABLAT?
                                              <1000>   NYY|YYY|Y
ERTI A DONTESI TABLA ELVET?
                                              <1001>   NNY|YYY|Y-
KESZITETT MAR DONTESI TABLAT?
                                              <1002>   --N|YYY|Y
IRT MAR DONTESI TABLA PROGRAMOT?
                                              <1003>   ---|NNY|Y
TUD COBOL-BAN PROGRAMOZNI?
                                              <1004>   ---|N-Y|-
TUD PL/I-BEN PROGRAMOZNI?
                                              <1005>   ---|N--|Y
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
ON MOST EPPEN DONTESI TABLAT NEZ
                                              <1006> -|X--|---|-
ONNEK NAGYSZERU KEPESSEGEI VANNAK
                                              <1007> X|---|---|-
FIGYELJEN A TOVABBIAKRA
                                              <1008> -|XX-|---|-
KESZITSEN 5 DONTESI TABLA VAZLATOT
                                              <1009> -|--X|---|-
ISMERJE MEG A PL/I VAGY COBOL NYELVET
                                              <1010> -|---|X--|-
TANULJA MEG A DONTESI TABLA
PROCESSZOR HASZNALATAT
                                              <1011> -|---|-X-|-
IRJON 10 PROGRAMOT DONTESI TABLAKKAL
                                              <1012> -|---|-X-|-
VIZSGALJA ELOLROL A DETAB-KERD
DONTESI TABLAT
                                              <1013> -|XXX|XX-|-
ON KEPES COBOL DONTESI
TABLAKKAL PROGRAMOZNI
                                              <1014> -|---|--X|-
ON KEPES PL/I DONTESI
TABLAKKAL PROGRAMOZNI
                                              <1015> -|---|---|X
ES<<>><><<>><><<>><><<>><><<>><><<>><><<>><><1016><><<>><><<>><>
```

```
************************          *****************
* COMPLETENESS CONTROL *          * TABLE ANALYSIS *
************************          *****************

THE ELSE RULE CONTAINS THE NEXT RULES :                  RULE
                                                            2    =--|---|-
                                                            3     =-|---|-
                                                            4      =|---|-
    <1000> NY                                               5       =<-|-
    <1001> YY                                               6        =-|-
    <1002> -Y                                               7         =|O
    <1003> -Y                                               8          =
    <1004> -N
    <1005> -N
```

A táblát két lépésben értékeljük ki:

1/ A feltételrész kiértékelése

Megállapitjuk, hogy mely szabályok teljesülnek. Egy szabály akkor teljesül, ha a benne Y-nal jelölt feltételek mind teljesülnek, es az N-nel jelölt feltételek közül egy sem teljesul. A közbmböseket nem vizsgáljuk.

Pl. Ha ön már látott döntesi táblát, érti az elvét, de még nem készitett döntési táblát, akkor a 4-es szabály teljesül. Ha már készitett is döntési táblát, de nem irt döntési tabla programot, és nem ismeri sem a PL/I, sem a COBOL nyelvet, akkor két szabály teljesül, az 5-ös és a 6-os. Ha ön még nem látott dontesi táblát, de erti az elvét, akkor egy szabaly sem teljesül: ilyenkor all be az első, az ún. else szabály.

2/ Az akciórész kiértékelése

A teljesült szabályok akciórészében X-szel jelölt akciókat végrehajtjuk. A kiértékelés fentről lefelé halad.

Pl. a fenti esetekben

- a 4-es szabálynak megfelelően készitsen 5 döntési tábla vázlatot, majd menjen a tábla elejére, kezdje újra a kiértékelést
- az 5-ös szabály alapján ismerje meg a két programozási nyelv egyikét, a 6-os szabály alapján tanulja meg a processzor használatát és irjon 10 programot
- az else szabály szerint önben nagyszerü képességek vannak. További bejegyzés nincs: alul kilépünk a táblából, vége a feladatnak

A döntési tábla processzorok

A programban a döntési táblákat a logikailag megfelelő helyre tesszük. A feltétel- és akcióbejegyzéseket külön kártyákra irjuk. E kártyák elején E betü áll. A táblára rácsuroghatunk, odaugorhatunk, vagy hivhatjuk CALL/PERFORM utasitással.

A processzor a döntési táblát bináris alakra hozza, munkaterületet deklarál hozzá, mindezt PL/I ill. COBOL nyelven. Az igy előállt "tiszta" PL/I ill. COBOL programot le kell forditani. Linkeléskor kerül hozzá az interpreter rutin: ez 140 byte-os assembler modul, a táblák végrehajtásának vezérlését végzi. Az interpreter előnye a rövid processzálási idő, a rugalmas táblavégrehajtás /mindig csak a még érvényben lévő szabályokkal foglalkozik/ és a tábla dinamikus módositásának lehetősége. Tapasztalatunk szerint nagy programok esetében sem rosszabb az interpretálás hatásfoka, mint a döntési tábla nélkül irt hagyományos kódolásé.

A bináris kódolás során a feltételrész minden bejegyzés-sorából két gépi szó generálódik: az első az ún. NO-szó, a második a YES-szó /mindegyik 32 bit/. Egy adott feltétel teljesülése/ nem-teljesülése szabja meg, hogy a két szó közül melyik az aktuális. Logikai ÉS müvelettel szürjük ki azokat a szabályokat, amelyek nem teljesülhetnek. Az akciórész minden bejegyzés-sorából egy gépi szó készül. Ha a teljesült szabályokat tartalmazó CASE-szó és e szó logikai szorzata nem nulla, akkor végre kell hajtani az akciót.

Pl. a mintatábla harmadik feltétel-sora igy kódolódik:

```
        - - N Y Y Y Y
  NO:   0 1 1 1 0 0 0 0
  YES:  0 1 1 0 1 1 1 1
```

Kiértékeléskor a harmadik feltételt akkor kell egyáltalán vizsgálni, ha az eddigi feltételek eredménye:

```
  CASE: 0 0 0 1 1 1 1 1
```

Ha a harmadik feltétel igaz, akkor az esetszó igy módosul:

```
  CASE: 0 0 0 0 1 1 1 1
```

Ha a feltétel nem igaz, akkor viszont

```
  CASE: 0 0 0 1 0 0 0 0
```

Az akciórész harmadik sorának kódolása:

```
        - X X - - - - -
        0 1 1 0 0 0 0 0
```

Mivel ennek és a CASE szonak logikai szorzata nulla, most nem hajtjuk végre az akciót.

Tapasztalatok a döntési tábla processzorok használatáról

A DOS PL/I döntési tábla processzort idáig mintegy 15 helyen installálták. Amennyire egyszerü az installálás, annyira nehéz rávenni a programozókat, hogy használják a processzort. A hagyományos blokkdiagram-kódolás folyamatban való gondolkodást kiván, ehhez képest szokatlan a döntési táblák állapotokat rögzitő gondolatvilága. Tanfolyamokkal és raszoritással lehet csak előbbre jutni. Az a programozó, aki 5-10 programot már megirt döntési táblákkal, magától is igy fog programozni a továbbiakban.

A programozás hatékonysága jelentősen javul. Mivel a döntési táblák közvetlenül irhatók a programba, kimarad a folyamatábrából történő kódolás időigényes, és rengeteg hibát okozó lépése. A program logikája a döntési táblákból világosan látszik, könnyen ellenőrizhető és javitható. A processzor analizáló és teljességellenőrző opciója hozzásegit a lekötetlen ágak ill. ellentmondások felderitéséhez. A processzor futási ideje egy nagyságrenddel kisebb, mint a PL/I ill. COBOL fordítóé, igy jelentéktelen a fordítási idő növekedése. A processzor nyomkövető opciókkal segiti a programbejátszást. A processzor listaja egyben dokumentumnak is használható.

Legújabb kisérleteinkben rendszerprogramokat is irunk döntési táblákkal.

Saját tapasztalataink alapján javaslom a PL/I és COBOL programozással foglalkozóknak, hogy ismerkedjenek meg a döntési tábla processzorokkal.

INTERAKTIV BASIC PROGRAMOZÁSI RENDSZER MEGVALÓSITÁSA TÖBBTERMINÁLOS IDŐOSZTÁSOS KÖRNYEZETBEN

Kaufmann Kálmánné, Szendrényi Tibor
Távközlési Kutató Intézet

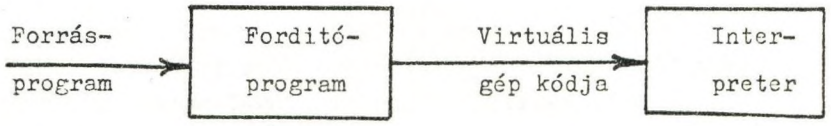
A Távközlési Kutató Intézetben az R10 kisszámitógép szolgáltatásaként egy többterminálos interaktiv müködtető rendszert fejlesztettünk ki /TSM-Time Sharing Monitor-/. További lépésként létrehoztunk egy, a monitorra támaszkodó BASIC programozási alrendszert.* Az előadás célja az alrendszer felépitésének és tulajdonságainak ismertetése.

Magasszintü nyelv gépi megvalósitásakor két lehetőség kinálkozik. A legelterjedtebb módszer szerint a forrásprogramot egy forditóprogram a tárgygép gépi kódjára forditja le. Másik lehetőség a tárgygép szintjének megemelése a forrásprogram szintjére, vagyis a program közvetlen végrehajtása. Az első megoldás előnye, hogy a leforditott program az adott tárgygépen gyorsan fut és egyszeri forditás után akárhányszor lefuttatható. Ezzel szemben a forditási idő - miután a gépi kód és a forrásprogram szintje között nagy a különbség - meglehetősen lassú. A módszer másik hátránya, hogy a gépi kódú program futás közben kevéssé kézbentartható, valamint a forditás folyamán olyan információk is elvesznek, melyekre a program belövése során szükség lehet /változók szimbolikus nevei, sorszám, stb./. Ezek a hátrányok interaktiv használatú nyelvek esetében megengedhetetlenek. A forrásprogram közvetlen végrehajtása ugyan mindezeket a hátrányokat kiküszöböli, a program futási ideje azonban jelentősen megnő, mivel a programsorok szintaktikus elemzését minden végrehajtáskor meg kell ismételni /pl. ciklusoknál/. Ez a megoldás akkor célszerübb, ha a gép szintje a forrás-

*A munkát a VIDEOTON megbizásából végeztük.

program szintjének felel meg, vagyis külön software interpreter nem szükséges.

Az általunk választott megoldás a két módszer ötvözete. Definiáltunk egy, a tárgygépnél magasabb szintü virtuális gépet. A forditóprogram erre a gépre generál kódot, magát a virtuális gépet pedig egy software interpreter realizálja. A forditás igy gyorsabb, mintha a generált kód



1. ábra

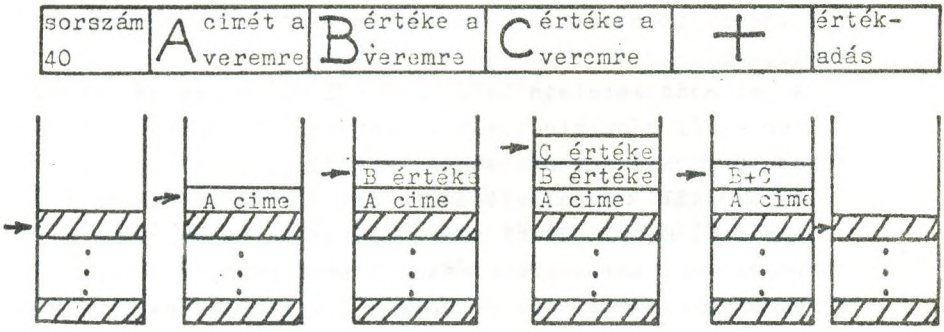
az R10 gépi kódja lenne és a virtuális gép nyelvébe a forrásprogram nyomkövetés szempontjából lényeges információi is átvihetők. Az interpreterrel a futó program kézbentartása is lehetővé válik, ugyanakkor a futási idő a közvetlen forrásprogram végrehajtásnál rövidebb, mivel a virtuális gép nyelve a magasszintü nyelvnél jóval egyszerübb.

A virtuális gép nyelvének definiálásakor a fentieknek megfelelően a következő szempontokat vettük figyelembe. Egyrészt a nyelv struktúrája nem eshetett messze a forrásnyelvétől a forditás gyorsasága érdekében, ugyanakkor elég egyszerünek is kellett lennie ahhoz, hogy gyorsan futtatható legyen. A nyelv a forrásprogram operátorait és operandusait a Jan Lukasiewicz féle un. Reverse Polish /RVP/ formátumban ábrázolja. Ez azt jelenti, hogy a megszokott sorrendtől eltérően az operandusok megelőzik a hozzájuk tartozó operátort /pl. az A+B kifejezés RVP alakja AB+/. Ebben az ábrázolásban az operátorok és operandusok sorrendje egyértelmüen meghatározza a végrehajtási sorrendet, igy a kifejezések zárójelezése feleslegessé válik /2 ábra/. A forrásprogram kulcsszavait és vezérlő jeleit is operátornak tekintjük, igy pl. a LET A=B+C forrássor RVP alakja ABC+LET.

2. ábra

Az RVP formátumuvá transzformált program interpretálásánál verem módszert alkalmaztunk. A beérkező operandusokat egymás után egy veremre, az un. müveletvégző veremre helyezzük, majd az operátor hatására a verem legfelső néhány /meghatározott számú/ elemén végrehajtjuk az operációt és redukáljuk a vermet. A 3. ábra egy BASIC forrássort, RVP megfelelőjét, valamint az interpreter által kezelt veremtartalmak alakulását illusztrálja a program végrehajtása során.

Forrássor: 40 LET A=B+C
RVP:



3. ábra

A kiragadott RVP részlet minden egyes eleme a virtuális gép egy-egy utasitását reprezentálja. A "sorszám" utasitás a verem állapotát nem módositja, szerepe csupán a forrássor

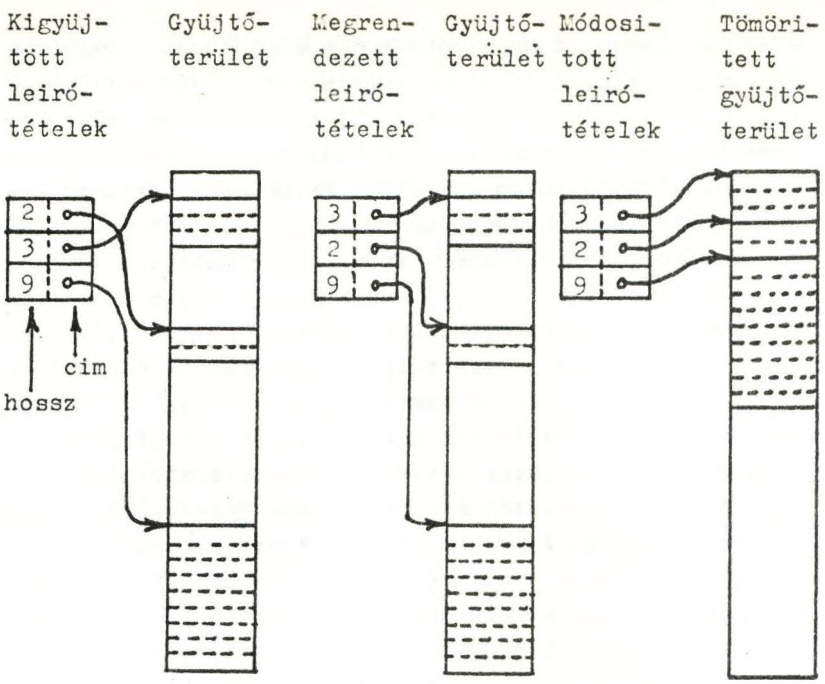
sorszámának megőrzése. A következő utasitás célja egy változó címének a veremre töltése. Az utána jövő két utasitás egy-egy változó veremre töltésére ad parancsot. A "+" utasitás hatására a verem két felső eleme által tartalmazott érték összeadódik, az eredmény az alsó operandus helyére kerül, a verem legfelső eleme lesöprődik /redukció/. Végül a "LET" utasitás a verem tetején levő értéket az alatta levő elem által kijelölt cimre tölti. A virtuális gép utasitáskészlete a müveletvégző verem szempontjából három csoportra osztható. A verem tartalmát nem módositó /sorszám, vezérlésátadások/, a vermet töltő /változó vagy konstans értékét, illetve változó címét a veremre/, valamint a vermet redukáló /értékadás,+,-,*,stb./ utasitások. A vermet töltő utasitások operandusaként szereplő változót a változó un. szimbólumtábla indexével azonositjuk. /Konstans esetén az operandus a konstans értéke./ A szimbólumtábla a programban előforduló összes változó, függvény, valamint a függvények formális paramétereinek leirását /szimbolikus név, tipus, paraméterszám, stb./ tartalmazza. Ezt a táblát az interpreter az RVP kiegészitő információjaként lényegében készen kapja a forditóprogramtól.

Az előadás keretein belül nem áll módunkban az interpreter ennél részletesebb ismertetése, csupán két érdekesebb algoritmusát szeretnénk kiemelni.

A TSM BASIC megvalósitásakor egyik szempontunk volt a szokásosnál kiterjedtebb szövegmanipulációs eszközök biztositása /pl. karaktersorozatok összefüzése, rész-karaktersorozat képzése, stb./. Ezek a lehetőségek nagyszámú karaktersorozatot tartalmazó programot eredményezhetnek. Amennyiben egy adott változóhoz rögzitett hosszúságú memóriaterületet rendelnénk hozzá, úgy ezt a területet meglehetősen nagyra kellene választani, mivel a karakterváltozó hossza előre meg nem határozható. Ez igen nagy memóriaterületet igényelne, ugyanakkor rövid karaktersorozatoknál rossz memóriakihasználást eredményezne. Az R10 TSM keretei között a fenti - statikus - karaktersorozat ábrázo-

lást nem lehetett megvalósitani a korlátozott memóriaterület miatt. /Az R10 viszonylag kis memóriáján kivül az időosztásos üzemmód is korlátot jelent./ Ezért a jobb memóriakihasználású dinamikus karaktersorozat kezelést alkalmaztuk. A futás során létrejött karaktersorozatokat a memória kijelölt részében, az un. gyüjtőterületen helyezzük el. Minden egyes karakter tipusú változóhoz egy leirótétel tartozik, mely a szimbólumtáblában vagy a müveletvégző vermen a karaktersorozat értékeként szerepel. A leirótétel tartalmazza a karaktersorozat aktuális hosszát és címét. Valahányszor egy karakter tipusú változó uj értéket kap, vagy egy karaktersorozatot tartalmazó kifejezés kiértékelése során keletkezett részeredményre már nincs szükség, a gyüjtőterületen kihasználatlan részek, "lyukak" keletkeznek. Ezekhez nem tartozik leirótétel. Ha a gyüjtőterület végén már nem tudunk helyet biztositani új karaktersorozat számára, a gyüjtőterületet tömöritjük. A tömöritéshez először a területre mutató leirótételeket gyüjtjük ki, majd az általuk meghatározott cim szerinti növekvő sorrendbe rendezzük. A létrejött sorrendnek megfelelően az egyes karaktersorozatokat a gyüjtőterület elejétől kezdve egymás mögé mozgatjuk, és módositjuk a leirótételek cimrészét. Igy a gyüjtőterület végén ismét szabad területhez jutunk /4. ábra/.

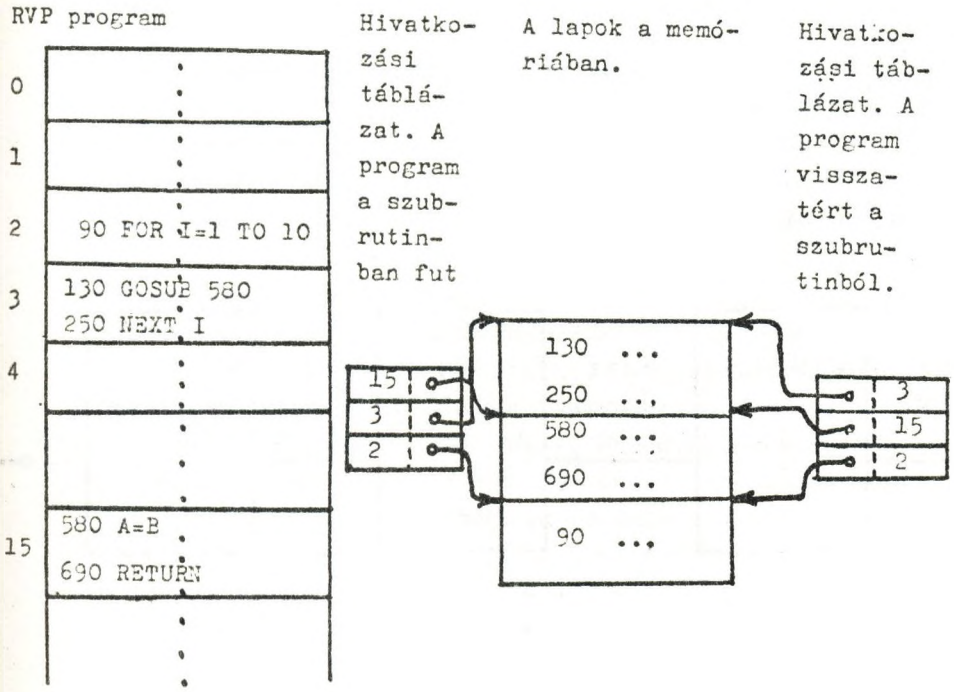
Arra a gyakorlati tapasztalatra támaszkodva, hogy a programban szereplő karakter tipusu változók egy részét adatok ideiglenes tárolására használják, más részük ezzel szemben viszonylag hosszabb ideig nem változtatja értékét, az algoritmus hatékonysága javitható. A gyüjtőterület vége felé ugyanis az elejéhez képest sokkal több "lyuk" keletkezik a gyakoribb értékváltozás következtében. Ezért a tömöritést mindig csak az előző tömörités után létrejött gyüjtőterület részre végezzük el. A teljes gyüjtőterület tömöritésére csak akkor kerül sor ismét, ha az utolsó tömörités után keletkezett szabad terület nagysága egy előre meghatározott méretnél kisebb.



4. ábra

A dinamikus karaktersorozat kezeléssel a korlátozott memória méret miatt felmerülő problémáknak csak egy részét oldottuk meg. A végrehajtandó RVP formátumú program mérete ugyancsak sokszorosan meghaladhatja a rendelkezésére álló memóriát. Ezért az interpreternél virtuális tárkezelést alkalmaztunk. A futó program számára a memóriában egy meghatározott méretü területet allokáltunk - rendszerünkben ez a terület a program méretétől függően max. 1280 byte -, és amennyiben a program ennél hosszabb, akkor csak az éppen szükséges darabja tartózkodik a tárban. Ezért a futtatandó programot lapokra osztjuk, és a futásnak megfelelően az aktuális részletet a diszkről beolvassuk. A diszktranszferek meglehetősen időigényesek, ezért érdemes a számukat a minimálisra csökkenteni. Ennek érdekében a rendelkezésre álló területnél ki-

sebb lapméretet célszerű választani, mivel igy a futó program sajátosságaihoz jobban tudunk alkalmazkodni. A tapasztalat szerint ugyanis a programok "csomósan" futnak, vagyis bizonyos szükebb környezetben aránylag hosszabb ideig tartózkodnak /pl. ciklusok végrehajtása közben/. Az 5. ábra egy speciális helyzetet mutat be. A választott lapméret a kijelölt memóriaterület harmada. Ezzel a választással elérhetjük, hogy a programban szereplő "FOR" ciklus teljes végrehajtásáig a memóriában tartható az általa hivott szubrutinnal együtt. Ha a választott lapméret megegyezne a rendelkezésre álló memóriaterülettel, akkor a ciklus egy végrehajtása négy laptranszfert igényelne.

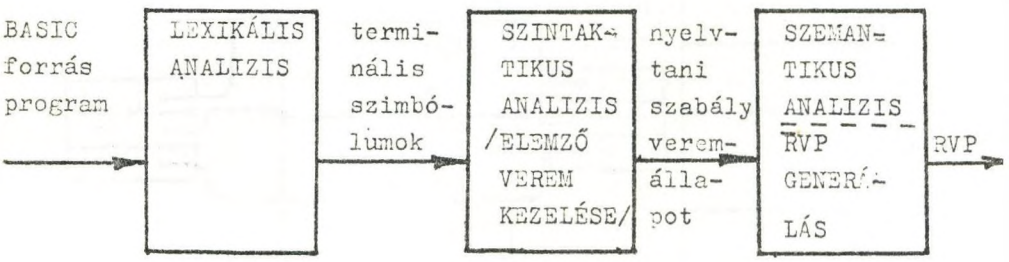


5. ábra

Abban az esetben, ha egy olyan lapra történik hivatkozás, amely nem tartózkodik a tárban, egy már bennlévőt kell felülirni az uj lappal. Az "áldozat"kijelölésére többféle stratégia kinálkozik. Mi az LRU /Least Recently Used/ módszert választottuk. Az az algoritmus lényege, hogy a legrégebben használt lapot irjuk fölül. A hivatkozások sorrendjének adminisztrálására egy "hivatkozási táblázatot" használunk. /A tábla hossza megegyezik a memóriában lévő lapok számával./ A tábla elemei a bennlévő lapok RVP -beli sorszámát valamint a memóriabeli címét tartalmazzák. A táblázatot mindig úgy rendezzük át, hogy az elemek sorrendje a hozzájuk tartozó lapok hivatkozási sorrendjének feleljen meg. Igy új lapot a táblázat utolsó eleme által kijelölt cimre olvasunk be. Az új lap beolvasása is természetesen a tábla átrendezésével jár, hiszen a tábla legfelső elemét kell az új laphoz rendelni.

A választott lapozó algoritmus hatékonyságát kisérleti tapasztalataink alátámasztották.

Most rátérünk az RVP-t létrehozó forditóprogram felépitésének és müködésének rövid ismertetésére. A program lényegében három modulból áll./6. ábra/.



6. ábra

A lexikális analizátor a beolvasott forrássor terminális szimbólumait /kulcsszavak, változók,konstansok,müveleti jelek,stb./ felismeri, és az azonositott terminális szimbólumot továbbadja a szintaxis elemzőnek. A lexikális analizis során épül a szimbólumtábla is. Erre az ad lehetőséget, hogy a BASIC-ben a változók tipusát a változók neve egyértelmüen meghatározza./ A,A$ /. A szintaxis elemzőt egy redukáló /bottom up/ tipusú táblavezérelt automata realizálja. Az automata az un. MSP /Mixed Strategy Parsing/ módszer megvalósitása. A szintaxis elemző balról jobbra olvastatja a forrássor szimbólumait, majd ezeket mindaddig egy un. elemző veremre helyezi, amig elegendő információ össze nem gyülik egy nyelvtani szabály felismeréséhez. Ennek az állapotnak a bekövetkezését egy vezérlő tábla rendszer detektálja. Az aktuális táblaelemet az elemző verem tetején lévő egy vagy maximum két szimbólum, valamint a soron következő terminális szimbólum jelöli ki. Amennyiben nyelvtani szabály felismerésére lehetőség van, az alkalmazandó szabályt egy újabb tábla alapján jelöljük ki.

Az elemző veremmel párhuzamosan egyéb munkavermek is töltődnek, ezek tartalmazzák a változók tipusát, szimbólumtábla indexét, a konstansok értékét, stb. A vermek redukálása előtt a felismert nyelvtani szabályt és a feltöltött vermeket a veremmutatókkal együtt az elemző átadja az RVP generátornak. Az RVP generátor a kapott információk alapján létrehozza a megfelelő RVP részletet, majd a vezérlést visszaadja az elemzőnek. Ekkor történik a tényleges redukálás. ../7. ábra/. A redukciót követően az elemzés tovább folytatódik. /Ujabb redukcióval, vagy a következő terminális szimbólum veremre kerülésével./

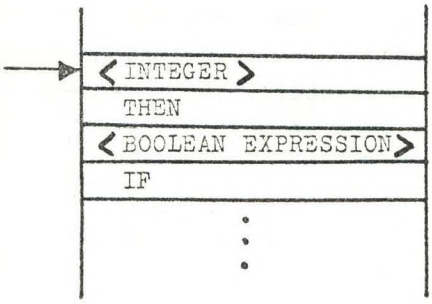
A szintaxis elemző vezérlő tábláit automatikusan állitottuk elő az XPL szintaxis elemző generátor segitségével, a BASIC nyelv BNF formátumban definiált nyelvtana alapján. Ez a lehetőség a forditóprogramot a nyelv bár-

milyen bővitésével szemben rugalmassá teszi, hiszen csak az új nyelvtannak megfelelő vezérlőtáblákat kell a régiek helyére beiktatni.

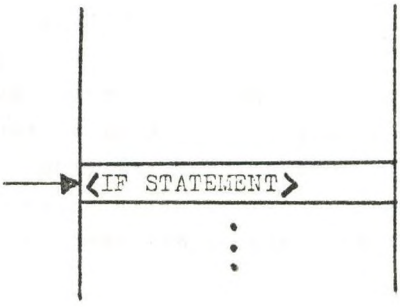
Az azonositott nyelvtani szabály:

<IF STATEMENT>::=IF<BOOLEAN EXPRESSION>THEN<INTEGER>

elemző verem a redukció előtt:



elemző verem a redukció után:



7. ábra

A továbbiakban kiemelnénk az R10 TSM BASIC bővitéseit az ANSI 1976-ban kibocsátott alapnyelvéhez képest. A bővitések nagy részét a karaktermanipulációs eszközök bevezetése képezi. A nyelv "+" operációját karaktersorozatokra is kiterjesztettük. Két karaktersorozat összege a karaktersorozatok összefüzését jelenti /katenáció/. A karaktersorozatokat - mint azt már emlitettük - dinamikusan kezeljük. A relációkat értelmeztük karaktersorozatokon is. Megengedtük karaktersorozat értékü felhasználói függvények definiálását, valamint a standard függvények készletét kiegészitettük karaktersorozatokkal kapcsolatos függvényekkel. /Pl. rész-karaktersorozat képzése, karaktersorozat hosszának megállapitása, stb./ A függvények

formális paraméterei is lehetnek karaktersorozatok.

A "+" operáció, az értékadás, a függvények aktuális paramétereinek megadása, stb. esetén a kétféle - numerikus ill. karaktersorozat - adattipus miatt tipusegyeztetésre van szükség. Ha a várt karakterérték helyett numerikus érték szerepel, automatikusan karaktersorozattá konvertáljuk. Ezzel szemben a forditott eset nem megengedett.

Bővitést jelent még, hogy az IF utasitásban megengedünk relációkon értelmezett logikai müveleteket is: a "vagy", az "és" ill. a "kizáró vagy" operációkat. Formai egyszerüsités, hogy az értékadásnál a LET kulcsszó elhagyható, valamint hogy több értékadás is megadható egy forrássoron belül.

A forrásprogram irásánál könnyitést jelent, hogy a sorok számozását a rendszer automatikusan elvégzi, hacsak a felhasználó másképp nem rendelkezik. A sorszámok alapértelmezés szerinti növekedésének ütemét a felhasználó kivülről megváltoztathatja.

A forrásprogram felvitelekor a rendszer interaktiv javitási lehetőségeket biztosit. A beolvasott BASIC forrássort a forditóprogram szintaktikusan ellenőrzi, amennyiben hibát talál hibaüzenetet küld, és a felhasználó azonnal korrigálhatja a hibát. Sorok beszúrása és törlése, valamint már felvitt sorok cseréje a sort azonositó sorszám alapján történik. A forrássorok felvitelekor a rendszer bizonyos parancsok felismerésére és végrehajtására is képes. A program irása közben a forrássor inputot paranccsal lehet terminálra ill. diszkre váltani. Az automatikus sorszámozás inkremensét az INCREM paranccsal állithatjuk be. A LOOK parancs a megjelölt tartományba eső, már felvitt sorokat a terminálon megjeleniti. A DELETE parancs végzi a már emlitett sortörlést. A BOUNDS parancs segitségével a felhasználó információt kaphat a rendszer generálási paramétereiről. Végül az EXIT parancs kiadásával a felhasználó a programfelvitel végét jelezheti a rendszernek.

A programok belövéséhez, a futás kézbentartásához nyújtanak segitséget az interpreter parancsai. Az interpreternek két alapállapota van. Az egyikben a program futtatását végzi, a másikban pedig a felhasználó által megadott parancsokat értelmezi és végrehajtja. Parancsértelmező állapotból az EXEC illetve a GOTO parancs hatására megy át a futtató állapotba. A futtató állapotból parancsértelmező állapotba az adott sorra kiadott PAUSE utasitás vagy parancs, illetve a futás közben fellépő, még javitható hiba következtében vált át. Az interpreternek adható parancsok két csoportra oszthatók. Az egyik csoportba olyan parancsok tartoznak, melyeknek megfelelő BASIC utasitás is létezik. Ezeknek az a célja, hogy futás közben - a program újraforditása nélkül - az egyes változók aktuális értéke megnézhető ill. megváltoztatható legyen, a DATA verem elemeit módositani tudjuk, valamint az utasitások végrehajtásának sorrendjét kivülről meg tudjuk változtatni. Ide tartoznak a READ,DISPLAY,LET, DATA,GOTO,STOP,PAUSE és RESTORE parncsok. A másik csoport elemei BASIC utasitásokkal ki nem jelölhető feladatokat definiálnak. Ezek segitségével lehet a futási idő maximumát megadni, egyes változók értékváltozásának illetve a program menetének nyomkövetését szabályozni. Ide tartozik az input illetve output diszkről terminálra vagy terminálról diszkre váltását elvégző parancs és az input lista elemeit egymástól elválasztó szeparátor karaktert kijelölő parancs is. Ezeket a feladatokat a TRACE, UNTRACE, a CHANGE, a TIME és a SEPARATOR parancsok végzik el.

A parancsok kialakitásánál törekedtünk arra, hogy - a BASIC alapszellemével összhangban - az egyszerü programok futtásához ne legyen szükség a parancsok ismeretére illetve használatára, valamint az egyes parancsok értelme magától értetődő legyen. Ahol lehetséges volt, a parancsok paramétereihez alapértelmezést rendeltünk.

Az a szolgáltatás, hogy a TSM BASIC rendszer input ill. output perifériaként nemcsak a terminált használhatja, teljes egészében a TSM-ben meglévő, általános célú file-keze-

lési lehetőségekre támaszkodik. A felhasználók fejlesztés alatt álló BASIC programjaikat, valamint a futtatható RVP programot a konfigurációtól függően diszken, minidiszken vagy mágnesszalagon tárolhatják. A futó BASIC program input illetve output adatmezeje ugyancsak elhelyezhető ezeken a tárolókon. Minden input és output file a rendszer katalógusrendszerébe beiktatott, névvel azonositott szegmensét képezi. A felhasználók a TSM általános célu parancsainak segitségével kommunikálhatnak a rendszerrel. A program felvitelét, forditását és futtatását ugyancsak TSM parancsok segitségével lehet kezdeményezni, ezek a parancsok végzik el az input ill. output file hozzárendeléseket is.

Végül egy táblázatban bizonyos szempontok szerint összehasonlitjuk az általunk implementált BASIC rendszert néhány elterjedt BASIC változattal, valamint az ANSI szabvány követelményeivel. A táblázatot a Communications of the ACM 1976 áprilisi számában található adatokból épitettük fel.

Az ismertetett rendszer kivitelezésében a szerzőkön kivül Dobos Magdolna, Kocsis Ferenc, Nagy Antal és Szekeres Zsuzsa vett részt. A szerzők ezúton is köszönetet mondanak Németh Józsefnek a rendszer megvalósitásakor nyújtott értékes szakmai útmutatásaiért.

	ANSI	DEC	HP	IBM	RAPIDATA	TSM BASIC
Duplapontos aritmetika	-	-	-	i	i	-
Legnagyobb szám	E+38	E+38	E+38	E+75	E+76	E+63
Legkisebb szám	E-38	E-38	E-38	E-75	E-76	E-64
Maximális sorszám	10^4	10^5	10^4	10^3	10^5	10^4
Karaktersorozat maximális hossza	19	10	73	256	256	generálási paraméter
Maximális tömb-méret	-	-	49000	32767	-	65536 (256,256)
Maximális dimenzió-szám	2	2	2	2	3	2
Többsoros felhasználói függvény definiálása	-	i	-	i	i	-
Több paraméteres felhasználói függvény	-	i	-	i	i	i
Nem BASIC szubrutinok szerkesztése	-	-	i	i	-	-
File kezelés /szekvenciális/	-	i	i	i	i	i
Direkt elérésü file	-	i	i	i	i	-
Időlimit megadása	-	i	-	i	i	i

GENDAX ADATGYÜJTŐ RENDSZER

Kisdi Gábor - Szabó Mihály - Szendi Gabriella -
Szentes Rezső
SZÁMKI

Intézetünkben többirányu fejlesztés folyik azzal a céllal, hogy az R10-es kisszámitógépét alkalmassá tegyük tár-adatfeldolgozási /TAF/ feladatok végzésére.

A gép hardware adottságai e szempontból igen kedvezőek, mivel real-time orientált, többszintü megszakitási rend-szerrel biró, mikro-programozott, modul szervezésü kisgép. A többszintü megszakitási rendszer nyujtotta multiprogramozást kihasználó real-rime, majd ennek fejlesztéseként létrejött multitask monitorok megfelelő alapot adnak a TAF irányu software fejlesztéshez.

Rendszerünk, melynek kidolgozását a mult évben kezdtük el, egy adatkoncentrátor. Főbb jellemzői a következők:

- az R10 multitask monitorára /MTM/ és ennek diszkes file kezelője épül

- szinkron és aszinkron vonalakat kezel CLA, CLS, COS csatoló tipusokkal, melyekre terminálként telex, VT340-es display és VTS56100 berendezés illeszthető
- generálása külön menetben történik, melynek során a konkrét felhasználói igényeknek és konfigurációnak megfelelő "testre szabott" rendszer alakitható ki.
- a rendszer üzemeltetését az R10 operátor végezheti; vezérlésére inditást, zárást, stb végző operátori parancsok szolgálnak

- a konfigurációhoz tartozó terminálokat a generálásokat meghatározott vonali protokoll szerint kezeli

- a felhasználónak /user programozó/ speciális supervisor szolgáltatásként megvalósitott, a file kezelésben megszokott hasonló logikai szintü adatkezelést biztosit
- a fizikai és logikai szintü adatmozgatás időbeni szétválasztása miatt szükséges átmeneti adattárolást automatikusan végzi.

Célrendszer generálás

A külön generálási menet célja, hogy a rendszer alapkoncepciójának megtartása mellett az adott igényeknek leginkább megfelelő célrendszert tudjunk a felhasználónak adni.
Igy software rendszerünk nem csak egy adott konfigurációt tud kezelni, nemcsak egy definiált adatstrukturán manipulál, s a vonali algoritmus során sem csak egyfajta procedurát hajt végre.
Rugalmasságát azzal biztositjuk, hogy a generálás során, a felhasználótól kapott információk alapján egy adatstruktura leiró és rendszer-vezérlő táblát /LDT/ alakitunk ki.

Ehhez a célrendszer generálását megelőzően a felhasználóknak egy ürlap csomagot kell kitölteniük. Egy-egy ürlap egy-egy terminálra vonatkozóan tartalmazza a terminál jellemzőit, a csatoló tipusát, az adatstruktura és a vonali algoritmus specifikációját, valamint a terminált azonositó állomás nevét és a hozzá tartozó jelszókat /az utóbbiak opciomálisak/.

Az ürlapokat kiértékeljük, s az igy nyert információk alapján készitjük el generáló programunk /LTDGEN/ kötött formáju input paramétereit és a rendszerdiszk felosztását.

A generáláshoz nem szükséges a terminálokkal kibővitett célgép használata, elegendő hozzá egy átlagos kiépitettségü R10-es.

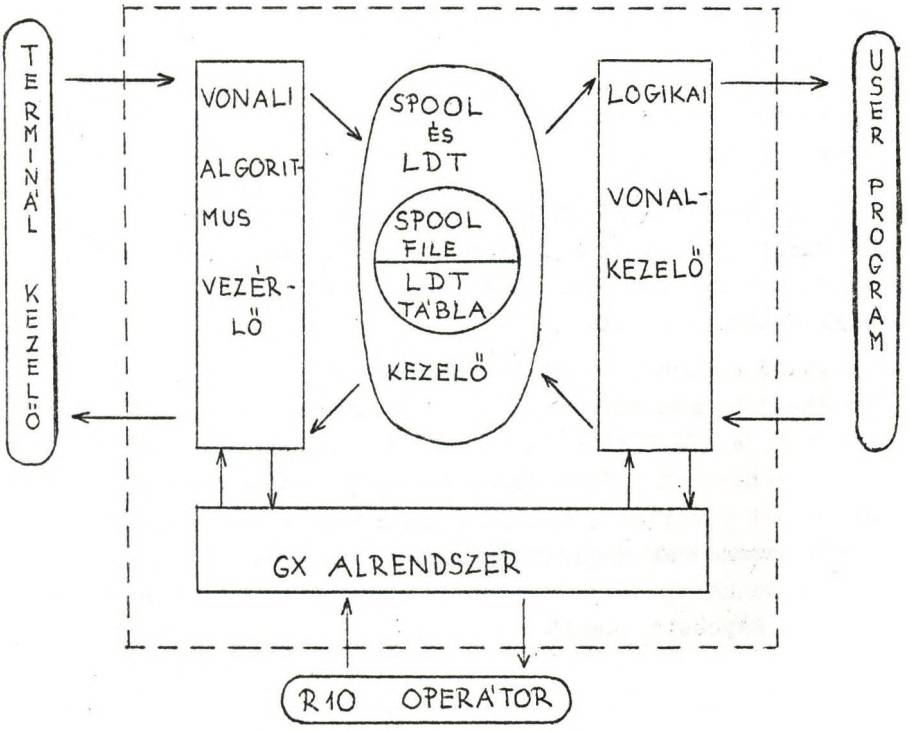
A célrendszer kialakitása 3 fázisban történik.

1. fázis: A GENDAX supervisor szolgáltatásait tartalmazó speciális multitask mónitor /MTM / generálása
2. fázis: Az LDTGEN program futtatása az ürlapok alapján elkészitett paraméterekkel. A program outputja egy információs file, mely tartalmazza a terminál jellemzőket és vonali algoritmus leiró LDT táblát, a rendszerben használatos állomás neveket és a hozzájuk tartozó jelszókat, valamint a GENDAS munkafileok méreteit és azonositóját.

3. fázis: Az R10 standard EP könyvtárásnak kibővitése a GENDAX rendszer-taskjaival.

A kivánt célrendszer /GENDAX / ezen előkészitett elemek célgépen történő betöltésével áll elő.

GENDAX funkcionális váza



Mint az 1. ábra mutatja az adatgyüjtő rendszer kapcsolatot tart

- az R10 operátorral
- a terminál kezelővel
- a feldolgozói programmal.

Ennek megfelelően beszélhetünk operátori, terminál kezelői és user interfacce-ről.

Operátori interface

Az RIO operátor feladata az adatgyüjtő rendszer üzemeltetése;
- a kezdeti betöltés
- az üzemi müködés elinditása
- az üzemi müködés zárása
- a vonalak állapotának ellenőrzése.

Az operátor üzeneteit a GX alrendszer fogadja operátori parancsok formájában.

A kezdeti betöltés /SYSLOAD/ a rendszer generálásakor előállitott információs file alapján történik. Eredményeként alapállapotba kerülnek a rendszer-diszken a GENDAX munka file-jai:

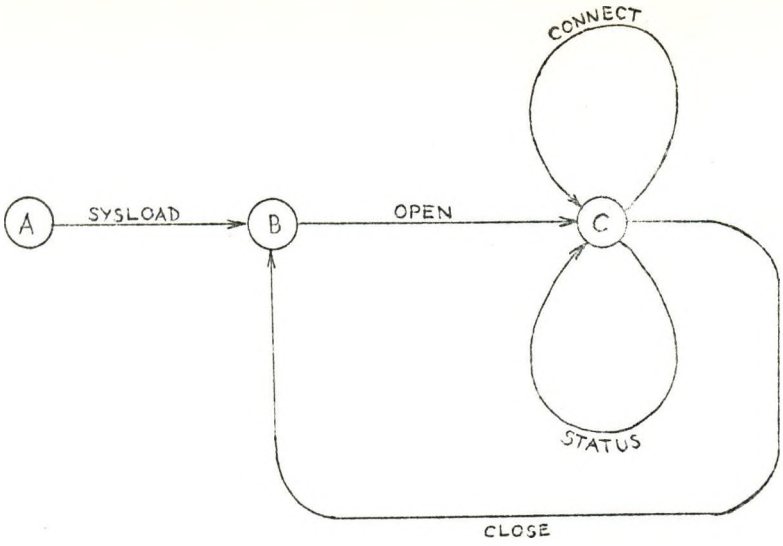
- a SYSLDT amelynek tartalma az LDT file
- az átmeneti adattárolásra szolgáló spool file, a SYSIO
- a SYSSN és SYSPW file-ok, melyek a rendszerben előforduló állomás neveket /SN/és jelszókat /PW/ tartalmazzák.

A SYSLOAD-t követően a rendszer üzemképes állapotban van és OPEN paranccsal megnyitható. OPEN hatására az LDT tábla a memóriába kerül, majd elindul a kapcsolatfelvétel valamennyi pont-pont kapcsolt vonalon.

A vonalak inicializálását követően megindulhat az adatforgalom, s a rendszer dinamikusan müködik mindaddig, amig egy CLOSE parancs nem érkezik. A CLOSE végzi el a rendszer zárást, mely a vonalak lekapcsolását és az aktuális /rendszerállapotot tükröző/ LDT tábla kimentését jelenti. A CLOSE-t követő OPEN a rendszer müködés folytatását eredményezi.

E három parancson kivül, melyek a rendszer állapotváltozásait okozzák még két parancs, a vonalakra vonatkozó STATUS és CONNECT ad lehetőséget operátori beavatkozásra.

E parancsok csak üzemi állapotban érvényesithetők; a STATUS-szal lekérdezhető a vonalak akutális állapota, a CONNECT-tel pedig az üzem közben /pl vonal hiba miatt/ bekapcsolt terminálokat lehet ujrainicializálni.



2. ábra

A vonali forgalom adatstruktúrája

A számitógép és a terminál között áramló adatinformáció strukturája három szintre tagolható:

rekord
mondat
üzenet.

Az adatstruktura legkisebb egysége a rekord, mely a terminál tipusától függően 5,7 vagy 8 bitből álló karakterek sorozata.
Logikailag összekapcsolt rekordok alkotják a mondatokat, ezek pedig az üzeneteket.

A rekordot rekord szeparátor /cr/, a mondatot mondat szeparátor /S/, az üzenetet üzenet szeparátor határolja /M/, s minden üzenethez tartozik egy információs rekord /HEAD/.

A feldolgozói program rekord szinten férhet hozzá az adatstrukturához. Rekordonként "olvashatja" a terminálról bejött adatokat és rekordonként "irhat" adatokat a terminálra. Olvasásnál jelzést kap arról, ha egy mondat vagy üzenet végéhez ért, irásnál jelezheti, ha egy mondatot vagy üzenetet le akar zárni.

Terminál kezelői interface

A GENDAX a vonali algoritmus keretében párbeszédet folytat a terminál kezelővel. A párbeszéd során rekordok haladnak időben felváltva az R10 felől a terminálra, ill. a terminálról az R10 felé. E rekordok két tipusba sorolhatók, adatrekordok és adminisztrációs rekordok.

Az adatrekordok tartalmazzák a felhasználó szempontjából értékes információkat, az adminisztrációs rekordok pedig a rendszer számára szükségesek az adatforgalom vezérlésében.

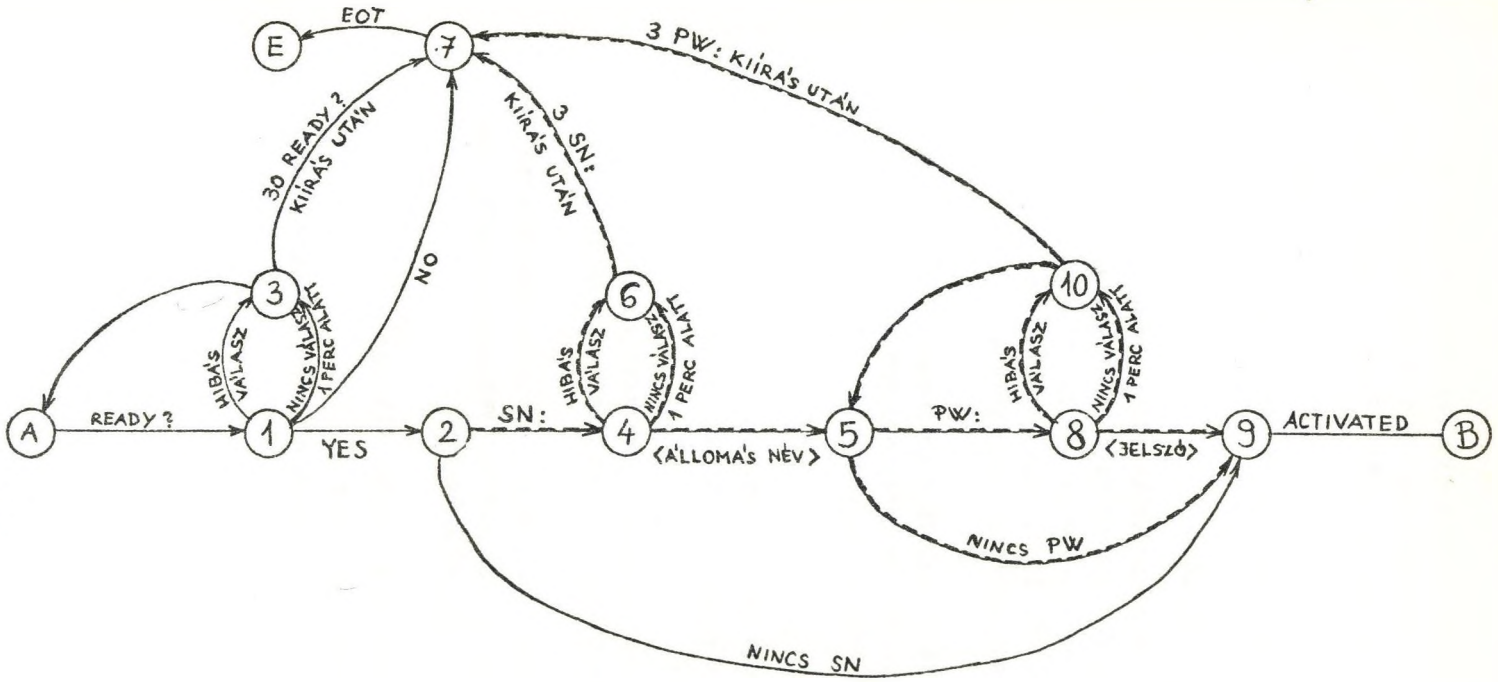
A párbeszéd formája és az adminisztrációs rekordok tartalma jól leolvasható a vonali algoritmus állapotdiagramjairól /ld. 4,5 6 ábra/.

A vonali algoritmust rendszer taskokkal valósitottuk meg a 4,5 6. ábra állapotdiagramjának megfelelően.

E megoldás, a memória optimális helykihasználását teszi lehetővé. A taskok prioritása ugy van meghatározva, hogy a rendszer mindig vételkész legyen.

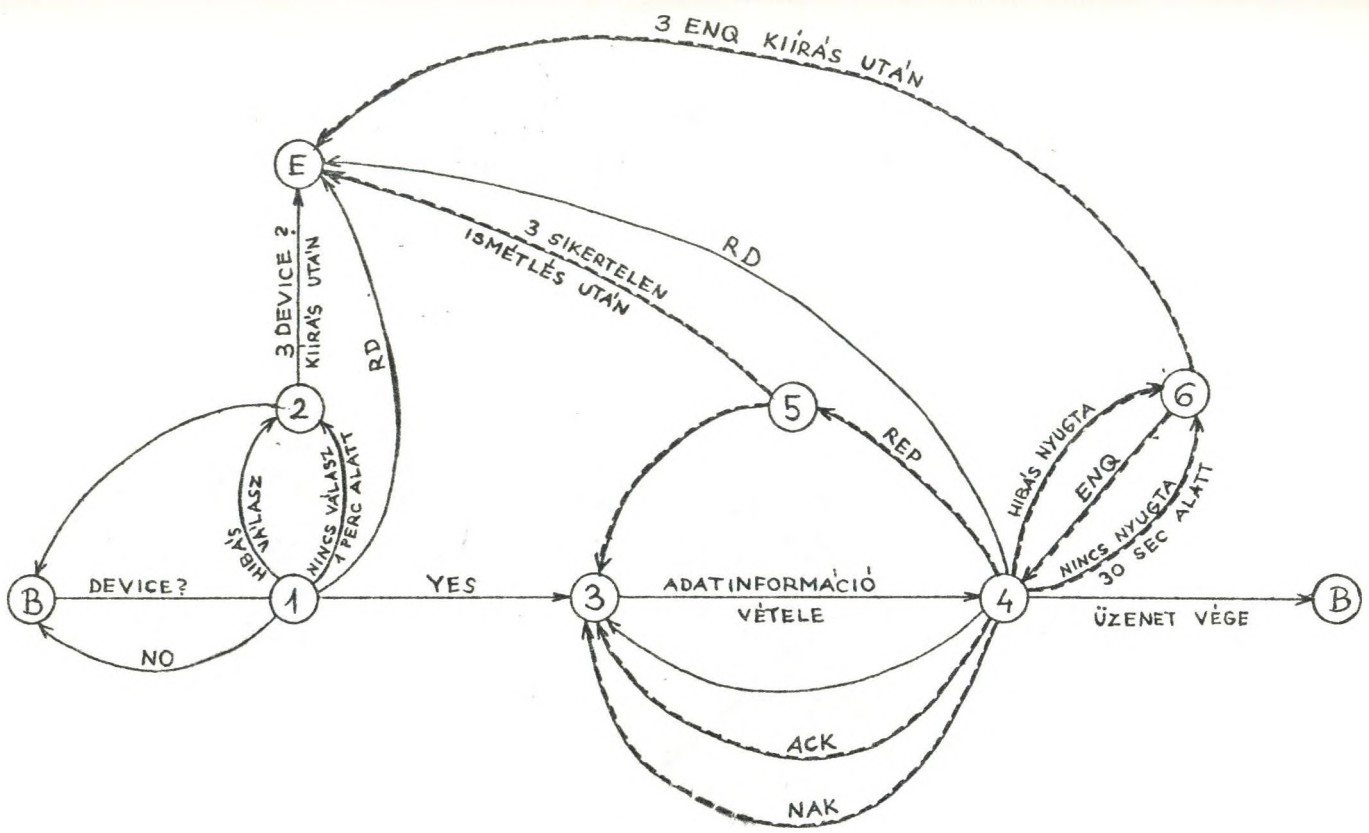
A vonali algoritmust két fő rész alkotja:

- a kezdeti kapcsolat felvétel
- a vonali forgalom, mely adási és vételi részre bontható

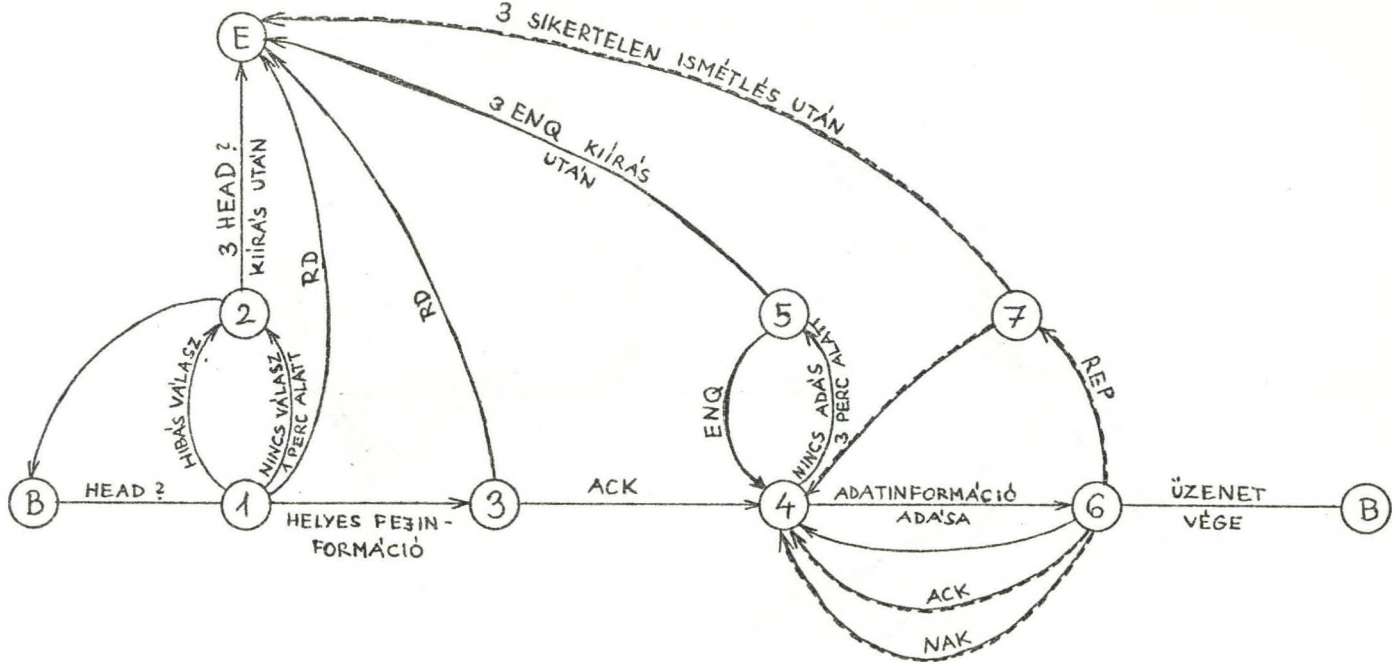


Kezdeti kapcsolat felvétel állapotdiagramja

4. ábra



A terminál oldali vétel állapotdiagramja
5. ábra



A terminál oldali adás állapotdiagramja

6. ábra

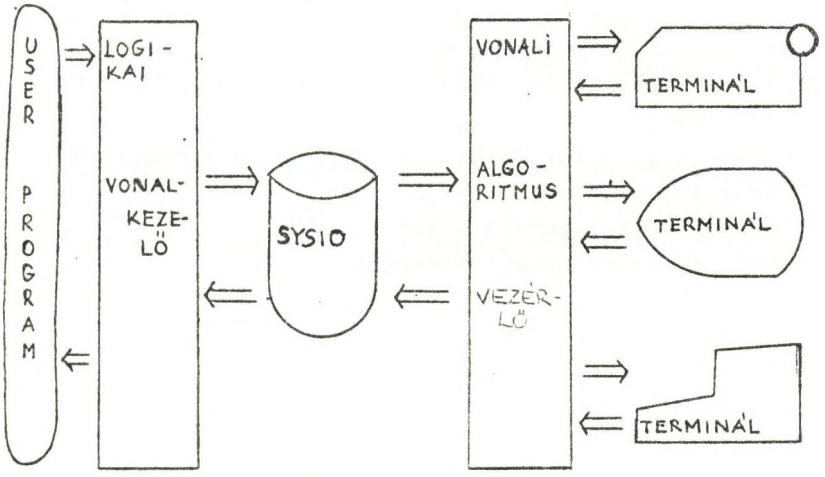
Átmeneti adattárolás

A vonali algoritmus során a terminál kezelés, az üzenetek vétele és adása a feldolgozói rendszer futásától függetlenül történik. Ezt a függetlenséget egy spool file /a SYSIO/ alkalmazásával biztositjuk.

A terminálokról begyüjtött adatok e speciális szervezésü file-ba kerülnek, s itt tartózkodnak mindaddig, amig a user program nem igényli őket. Forditott irányban, a feldolgozói programból terminálra küldött adatok is először a SYSIO-ba kerülnek, a tényleges kivitel a terminálok letelelésétől függően történik, a GENDAX vezérlete alatt.

A SYSIO-ban egy-egy fizikai vonalhoz egy-egy dinamikus hosszuságu láncot rendeltünk, melyben a vonalra küldött és még ki nem adott, ill. a vonalnról beérkezett de még fel nem dolgozott rekordok vannak. A SYSIO-t kezelő spooling rendszertől a rekordok olyan sorrendben kaphatók meg /mindkét irányban/, amilyen sorrendben bele kerültek. Tehát a feldolgozói program a rekordokat a terminálról történt beérkezés sorrendjében kapja meg, ill. a terminálra a feldolgozói programból történt kiküldés sorrendjében érkeznek ki az adatok.

Az adatok utja a rendszerben



7. ábra

User interface

A rendszer dinamikus müködése közben a logikai vonalkezelő, a LINE szekció tart' kapcsolatot a feldolgozói programmal.

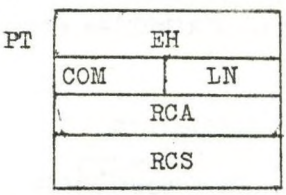
A LINE funkciói a következők:

- a vonalak logikai megnyitása és lezárása
- a terminálról vett és a SYSIO-ban elhelyezett rekordok átadása a feldolgozó rendszernek
- a feldolgozó rendszerből terminálra küldendő rekordok átvétele, és azok SYSIO file-ba küldése

valamint e tevékenységekkel kapcsolatos ellenőrzések, adminisztrációk és a szükséges adatkonverziók elvégzése.

A szekciót gyakori használata miatt a supervisorba épitettük be, igy egy példányban állandóan jelen van a rendszerben. Paraméterezése hasonló az RIO real-time file kezelő SV szekciójához.

Hivásakor egy CB-hez hasonló paraméter tábla cimét kell megadni az A-regiszterben.



ahol EH - a LINE által visszaadott, esemény, ill. hiba kód
COM- az igényelt funkciónak megfelelő parancskód
LN - a vonalszám
RCA- a rekord puffer kezdőcime
RCS- a rekord hossz byte-okban

Az igényelhető funkciók a következők:

Vonal logikai megnyitása, OPEN
Vonal logikai lezárása, CLOSE
Rekord olvasás, GET
Rekord olvasás wait-tel, GETW
Adatrekord irás, PUT
HEAD-rekord irás, PUTH
Mondat szeparátor irás, PUTS
Üzenet szeparátor irás, PUTM

E parancsok használata a felhasználótól nem kiván speciális vonalkezelési ismereteket, a user file kezelési tapasztalataira támaszkodva oldhatja meg a távadatfeldolgozási feladatot. Magyarázatra talán csak a GET és GETW közötti különbség szorul. Ez abban áll, hogy GET esetén ha éppen nincs feldolgozásra váró rekord a vonalról, a felhasználó erről jelzést kap /EH-ban/ s programja futhat tovább, mig GETW hatására csak akkor történik visszatérés az SV-ból ha a terminálról érkezett ujabb adat. A GET és GETW müködése abban az esetben ha van feldolgozandó adat, azonos.

A LINE SV szekció a parancsok végrehajtása közben a kritikus szakaszokat mint "oszthatatlan utasitást" hajtja végre.

Az eddigiekben a dinamikus müködés közbeni, adatkezelésre vonatkozó user interface-t ismertettük. Van ezen kivül a felhasználóval való kapcsolattartásnak egy másik formája is.

Mint a cikk elején ismertettük a GENDAX rendszer állapota 3 fázisban, a SYSLOAD, az OPEN és a CLOSE operátori parancsok hatására változik. Ezen rendszer állapot változá-

sokról is értesitjük a feldolgozót, kötött nevü US:SYL, US:OPN és US:CLO taskok megkreálásával.

Igy a usernek módja van az egyes fázisokhoz tartozó feldolgozói tevékenységek elvégzésére /pl. saját file-jainak inicializálására, ill. előfeldolgozás végzésére kezdeti betöltéskor, az adatkezelést végző taskok megkreálására inditáskor, a feldolgozás folytatásának előkészitésére záráskor, stb/ s a feldolgozói rendszer operátori parancsok szerinti vezérlésére.

A GENDAX alkalmazások

A GENDAX müködéséről még csak egy teszt-rendszer kapcsán vannak tapasztalataink. E teszt rendszer egy üzenet kapcsoló, melyben a rendszerhez tartozó terminálok az üzenetek első rekordjával cimezhetők, s igy irányitottan küldhetők adatok egyik terminálról a másikra. Segéd funkcióként dokumentációt készit az adatforgalomról. Futásával kapcsolatos tapasztalataink kedvezőek.
A GENDAX első tényleges felhasználója a Szolnoki MÁV, ők rendezőpályaudvari feladatok szervezésére kivánják felhasználni. Az üzenetek itt a beérkező és kimenő vonatok leirását tartalmazzák. A GENDAX-nak a vasutaknál használatos szigoru szabványokhoz való igazitása kisebb változtatást igényelt a vonali algoritmusban, és speciális telex kezelést követelt.

A módositásokat 2 ember/hónap ráforditással tudtuk elvégezni, ami rendszerünk rugalmasságát bizonyitja.

A későbbiekben rendszerünket alkalmassá kivánjuk tenni a terminálok interaktiv üzemmódban való alkalmazására is, ezzel jelentősen bővitve felhasználásának körét. E fejlesztés jelenleg folyik.

TERMINÁL ILLESZTÉS A CII-HONEYWELL-BULL 66 RENDSZERHEZ

Kovács Ödön - Gáti Pál
Államigazgatási Számitógépes Szolgálat

1. Bevezetés

Az Államigazgatási Számitógépes Szolgálat CII Honeywell-Bull /CHB/ 66/60-as rendszere kollektiv használatú számitógép. Nagyságát és kiépitettségét tekintve helyi batch jellegü feldolgozáson kivül egyidőben lehetővé teszi a felhasználónak a távoli elérési módot. A rendszer az ESZR elemek hozzákapcsolását nem támogatja, mégis törekedni kellett arra, hogy a szolgáltatásai hazai kisszámitógéppel, illetve terminálokkal is elérhetők legyenek.
Erre pedig szükség volt azért, mivel a rendszerhez egyre többen szeretnének hozzáférni, igénybevenni a time-sharing szolgáltatást és a remote batch lehetőség előnyeit. Az ÁSZSZ adathálózat egyre bővül és egyre több problémát jelent a különböző tipusú esetleg már meglévő elemek illesztése.

Az Intézet komoly erőfeszitéseket tett arra, hogy hazai berendezésekkel helyettesitse a CII Honeywell-Bull standard termináljait. A fontosabb tipusai ezeknek a következők:

- teletype kompatibilis terminálok
- CHB VIP tipusú képernyős párbeszédes vagy korlátozott mértékben batch terminálok
- IBM 2780 tipusú batch terminálok
- CHB RIP batch terminál és koncentrátor

Jelen dolgozatban röviden szeretnénk ismertetni a System 66/60 adatfeldolgozó hálózatához egy csatlakozási lehetőséget és egy hazai terminál, a VTS 56100 mikroprocesszoros végberendezés illesztését.

2. A VTS 56100 illesztésének szempontjai

A továbbiakban a CII-Honeywell-Bull VIP display-vel kompatibilis terminál kifejlesztését szeretnénk ismertetni. A feladat megoldásakor három alapvető probléma merült fel:

- az adatátvitel során pontosan meg kellett valósitani a CHB rendszerben használt vonali algoritmust. Ez elviekben eltér a hazai rendszerekben alkalmazott algoritmustól;

- figyelembe kellett venni a CHB 66/60 time-sharing rendszerének speciális szolgáltatásait és a hozzá kapcsolódó terminállal szemben támasztott követelményeit;

- alkalmazkodni kellett a kiválasztott hazai gyártmányú VTS 56100 terminál sajátságaihoz.

2.1. Vonali algoritmus

A CHB display tipusú terminájai kétféle üzemmódban dolgozhatnak:

- pont-pont /non-poll/
- multi-pont /poll-select/ összeköttetésben.

A CHB rendszerében a hálózatvezérlő processzor és a terminál folyamatosan vált üzenetet egymással. Az alkalmazott üzenettipusokat és azok felépitését az 1. táblázat mutatja. Itt tüntettük fel azt is, hogy mely üzeneteket használja a központi gép és melyeket a terminál.

	SYN	SYN	SYN	SYN	SOH	ADR	STA	FC1	FC2	STX	TEXT	ETX	LP	EOT	Közp. non-pol	Közp. pol-sel	Terminál
SELECT	X	X	X	X	X	X	X			X		X	X	X		X	
ADAT	X	X	X	X	X	X	X	X	X	X	X X X X	X	X	X	X	X	X
POLL	X	X	X	X	X	X	X			X		X	X	X		X	
STATUS	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X
ÜRES	X	X	X	X	X									X	X		X

1. táblázat

Üzenettipusok jelentése:

- "üres" üzenetet küld a berendezés, ha nincs hasznos információ
- "adat" üzenet tartalmazza az adatokat
- "status" üzenet az "adat" üzenetet nyugtázza /ACK/, ismétlést kér /NAK/ vagy foglaltságot jelez /BSY/
- "select" üzenet kiválaszt egy terminált, melyhez a következő "adat" üzenetet küldi

- "poll" üzenet felszólit egy terminált adásra. A végberendezés információ hiányában "üres" üzenettel válaszol.

Az üzenetekben szereplő karakterek jelentése a következő:

- SYN: karakter szinkronizációt biztositja
- SOH: üzenet kezdetét jelzi /start of header/
- ADR: "adat" üzenetben egy terminálon belül egy-egy periféria cimét, a többi üzenetben a terminál cimét adja meg
- STA: "status" üzenetben a nyugtázást jelenti, lehet ACK, NAK vagy BSY, egyébként nincs jelentősége, NUL vagy PRT karakter
- FC1, FC2: Funkciós kód, tetszőlegesen használható
- STX: adatrész kezdete /start of text/
- TEXT: tetszőleges adatok
- ETX: adatrész vége /End of text/
- LP: hosszparitás karakter
- EOT: üzenet vége /End of transmission/

2.2. A CHB hálózat speciális szolgáltatásai és funkciói

A központi rendszer magában foglal néhány olyan tulajdonságot, mely a VTS 56100 berendezéssel önmagában nem használható ki, de programvezérléssel beépithető volt. Alábbiakban nézzünk meg néhányat ezek közül.

A központi rendszer alkalmazza az úgynevezett "Formátum mód"-ot. Ez előre elkészitett táblázatok kitöltésére alkalmas.

Programfejlesztéssel oldottuk meg a tabulálást, mely nincs a VT terminál eredeti lehetőségei között.

Szükség volt a képernyő törlésére a cursor poziciójától a sor végéig, illetve a lap végéig.

Lehetséges a képernyő bármely területén az alfabetikus karakterek titkositása. Ez elsősorban a felhasználók azonositóinak és jelszavainak titkos kezelésénél szükséges.

2.3. VTS 56100 - VIP 7750 közötti különbség

Az illesztésre kiválasztott terminálnál figyelembe vettük, hogy felépitése és lehetőségei minél jobban hasonlitsanak a CHB VIP berendezéséhez. Természetesen ez csak megközelitően sikerült. Jelentős eltérések vannak a két egység között. A két berendezés részletes ismertetése nélkül röviden felsoroljuk ezeket:

- a memória kapacitása erősen korlátozott /VIP: 24 KByte, VTS 56100: 8 KByte/
- VTS 56100 nem tartalmaz interrupt logikát
- a magyar berendezés mikroprocesszora és felfrissitő memóriája viszonylag lassú
- a memóriavédelem és rendszer kijelzések eltérő formája

- képernyő mérete közötti különbség /VIP: 24x80, vagy 12x80 karakter; VTS 56100 16x80 karakter/

- VTS 56100 rendelkezik ROLL funkcióval

3. VTS-78 Program

A VTS-78 program olyan terminál vezérlő program, amely biztositja a VTS 56100 képernyős terminál illesztését a CII-Honeywell-Bull 66-os rendszeréhez.

A 4 KByte Read only Memory-ban beégetett program gondoskodik a szinkron adatátvitel végrehajtásáról a képernyő, a tasztatura és a terminálhoz kapcsolt mátrix nyomtató vezérléséről. /Az eredeti Honeywell VIP 7750 terminál 24 KByte memóriát igényel!/

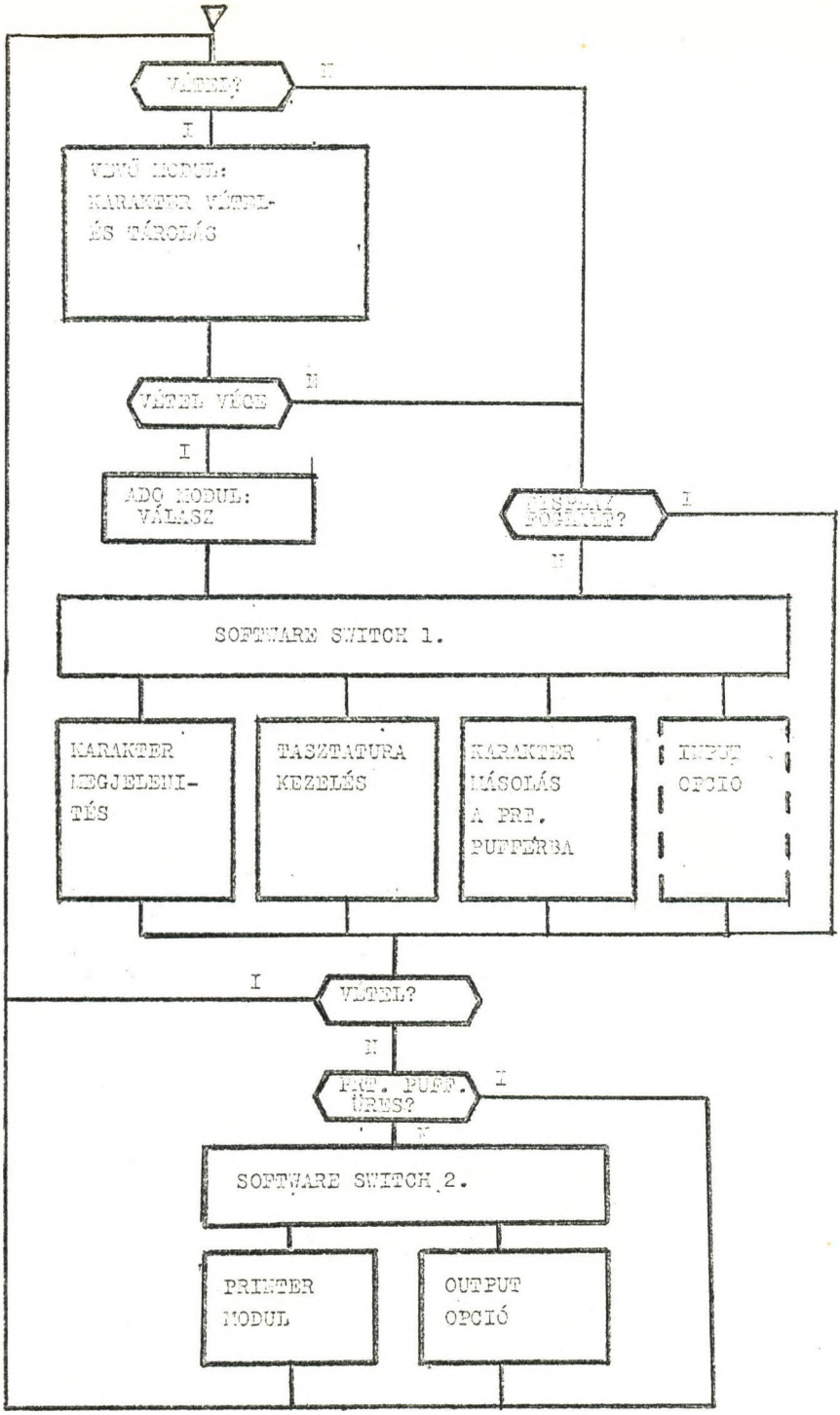
A VTS 78 program felépitését és müködését 1. ábra foglalja össze vázlatosan. A továbbiakban csupán néhány tervezési problémával foglalkozunk.

Lehetőség van periféria bővitésre /lyukszalag olvasó, lyukasztó vagy mágnes kazetta/, az alap firmware módositása nélkül, mivel az már tartalmazza az opcionális perifériákat kezelő modulok software interface-et.

4. A VTS-78 program tervezésének főbb szempontjai

4.1. Software switches

A VTS-78 program megirása során kezdetben az okozott a legnagyobb problémát, hogy miként lehet megoldani egy interrupt logika nélküli processzorral a szinkron adó/vevő egység, vagyis a real-time perifériák vezérlését és közben számos spare-time


VÉTEL?
I
N
VEVŐ MODUL:
KARAKTER VÉTEL-
ÉS TÁROLÁS
VÉTEL VÉGE
N
I
ADÓ MODUL:
VÁLASZ
I
N
SOFTWARE SWITCH 1.
KARAKTER
MEGJELENÍ-
TÉS
TASZTATURA
KEZELÉS
KARAKTER
MÁSOLÁS
A PRF.
PUFFERBA
INPUT
OPCIO
I
VÉTEL?
N
PRF. PUFF.
ÜRES?
I
N
SOFTWARE SWITCH 2.
PRINTER
MODUL
OUTPUT
OPCIÓ

1. Ábra

4.2. Kontrol táblák

Az adatátviteli algoritmusban alkalmazott számos vonali és periféria, főleg display vezérlő karakterek, valamint a tasztaturáról érkező parancs kódok dekódolása komparálással túlságosan sok /160/ byte-ot igényelt volna, ez tette szükségessé az úgynevezett kontrol táblák definiálását. Ehhez viszont be kellett vezetni a fent emlitett indirekt switch fogalmát. A kontrol táblákban az egyes vezérlő rutinok belépési pontjainak, a kontrol tábla végére vonatkozó relativ cimei helyezkednek el. A relativ cimek pedig csak 1 byte-osak igy 32 byte elegendő az összes vezérlő funkció kiválasztására. Kontrol karakterek detektálásakor a vezérlés a kontrol rutinok belépési pontját kiszámitó subrutinra jut, amely a kontrol táblából kiveszi az észlelt karakternek megfelelő relativ cimet. Ezen cimhez a vezérlő táblák végcimét hozzáadva adódik a kontrol rutin belépési cime, amelyet az INDIRECT switch-be töltve és az arra való vezérlés átadás után kezdődik meg a kiválasztott kontrol rutin végrehajtása.

A VTS-78 programban 3 db kontrol táblát kellett alkalmazni, egyet a real-time perifériától érkező vezérlő karakterek dekódolására, kettőt pedig a tasztatura parancsok felismerésére.
Ezen két kontrol tábla használatát az úgynevezett formatum mode létezése indokolja.

4.3. FORMATUM MODE

Formatum módban a képernyő nem homogén, hanem állandó és változó mezőpárokra oszlik. Az állandó mezők memória védettek, tehát a tasztaturáról nem lehet letörölni azokat! Az állandó mezők csak a szinkron vonalról, tehát a számitógép felől definiálhatók. A változó mezők paraméterekkel rendel-

keznek, melyeket a terminált vezérlő számitógép adhat meg. Ezek különböző tulajdonságokat határoznak meg, úgy mint nyomtatás tiltás, vagy adás tiltás, illetve csak numerikus karakter bevitele a változó mezőbe. A tasztaturáról csak a változó mezők tölthetők fel, és csak ezek jutnak a számitógép felé adás kérés esetén.
Ilymódon lehetőség van arra, hogy a bizonylatok /vagyis a változó mezők/ kitöltésével teljes képernyőnyi adatot továbbitsunk.

A formatum mode-ban teljesen másfajta vezérlő utasitások érvényesek, ezért volt szükség 2 db kontrol tábla alkalmazására a tasztatura vezérlő rutinban.

4.4. ROLL és PRINT

A VTS-78 terminál vezérlő program rendelkezik egy nagyon kényelmes és hasznos display funkcióval, amit az eredeti Honeywell VIP terminál nem tud. Ez a ROLL funkció.

A VTS 56100 hardware lehetőségei alapján kinálkozott a ROLL funkció beépitésére, azonban ez számos kompatibilitási problémát vetett fel. A ROLL alkalmazásakor ignorálni kell néhány vonali vezérlő funkciót igy: a képernyő törlést, a cursor cinzést, az adás kezdet- és vég kijelölő escape szekvenciát, amelyek nyilvánvalóan értelmetlenek, sőt tilosak a ROLL alkalmazásakor, de ezen funkciók eldobásával szegényebb lett volna a terminál intelligenciája.

Végül is mindkét megoldás megmaradt a tasztaturán elhelyezett úgynevezett FORM/SW kapcsoló segitségével, amellyel ki lehet választani a ROLL üzemmódot.

A ROLL üzemmód önmagában még nem hozott volna lényegesen újat. Komoly előre lépést jelentett az, hogy a ROLL funkció hatása során a képernyő felső részén kilépő sorok nem vesznek el, hanem a terminálhoz kapcsolt printeren jelennek meg!
Igy a felhasználónak lehetősége van arra, hogy a munkája közben is állandóan hard copy-t nyerjen, azonban mégis használhatja a katódsugárcsöves terminálok összes előnyeit.

A ROLL és Print funkció megvalósítása során számos software probléma merült fel. A terminálba beépitett display periféria dinamikus MOS tárral rendelkezik, melynek átlagos elérési ideje 2 ms. A fentemlitett ROLL print funkció alkalmazása esetén ez a hozzáfordulási idő 4 ms-re növekszik /irás + olvasás/. Ez korlátozza a maximális vonali sebességet 1200 baud-ra.

A ROLL müveletet a hardware elvégzi, de a software-nek kell érzékelni a ROLL előtti pillanatot, hogy időben kimenthesse a képernyő legfelső sorát. Nyilván ez idő alatt nem lehet feltölteni a képernyőt, ezért csak a Receive bufferben akkumulálódhat a vonali üzenet.

A ROLL során kilépő sor a Printer bufferba kerül, ahonnan pedig a printert kezelő rutin juttatja el az adatot a printerre.

A ROLL és PRINT alkalmazásának másik előnye, hogy 24 soros VIP-ként lehet használni a VTS 78 programmal ellátott VTS 56100 terminálokat, noha azok csak 16 fizikai sorral rendelkeznek.

4.5. Space tömörítés és adás

A tasztatura vezérlő program gondoskodik az adáskezdet kijelöléséről. Mindig az első újonnan beütött karaktertől kezdődik az üzenetek továbbitása a számitógép felé. Az adási szöveg végét pedig a cursor által kijelölt pozició jelzi. A terminál a space tömörités során a számitógép felé csak az információt hordozó sorokat továbbitja, és a sorvégi betüközöket pedig nem.

Ennek a funkciónak a megvalósitására kellett alkalmazni az úgynevezett end of line táblát, mivel a képernyőre nem lehetett felirni sorvég jelző karaktereket, igy minden karakter képernyőre való beléptetésekor meg kell jegyezni az újonnan létrejött sorvégeket.

1280 karakter fér el a teljes képernyőn, ez túlságosan hosszú ahhoz, hogy egyszerre továbbitsuk a számitógép felé. Ezért belső opcionális lehetőség van 256 byte-os blokkokra tördelni az adási üzenetet.

5. VTS 78 gyakorlati felhasználása

A terminál program egy adott rendszerbe való illesztés céljából készült. Felhasználási területe csak a CHB System 66 adatátviteli hálózatában van. Azonban elmondhatjuk, hogy az adott eszköz lehetőségei mellett kisebb módositásoktól eltekintve teljesiti a CHB standard termináljának összes fontosabb tulajdonságát, azt teljes mértékben helyettesiti.

Az ÁSZSZ tulajdonában jelenleg 15 db VTS 56100 terminál müködik a felhasználóknál 1978 májusa óta.

A VTS 78 program a Videoton és az ÁSZSZ közös tulajdona, tudomásunk szerint a Videoton a programrendszert forgalmazza.

AZ ALAP-2 ADATBÁZISKEZELŐ RENDSZER

Kovács Zoltán
KGISzSzI

A hardware és alap-software szempontból harmadik generációs számitógépek széleskörü elterjedése Magyarországon /1972-től/ arra ösztönözte a felhasználókat, hogy adatfeldolgozási rendszereikben is generációt váltsanak. Térjenek át a hagyományos folyamatszervezésről az integrált adatkezelést megvalósitó rendszerek kialakitására.

A hagyományos adatfeldolgozási rendszerek rendszerint egyedi programozással készülnek és igen mereven kötődnek ahhoz az alkalmazási területhez, amelyet kiszolgálnak. Az érintett terület minden funkcionális szerve külön létrehozza a saját feldolgozásaihoz szükséges adathalmazokat, általában több független file formájában. Ilymódon az adatok decentralizáltan és többszörösen fordulnak elő, és minden felhasználó a saját igényeinek megfelelően tárolja /csoportositja, nyomtatja/ lényegében ugyanazokat az adatokat. A többszörösen előforduló adatok karbantartásának nehézkes szinkronizálása és az egész rendszer merevsége a változtatásokkal szemben nem teszi lehetővé tetszőleges operativitású és nagyságú számitógépes programhalmaz kialakitását és megbizható üzemeltetését.

Az integrált tárolást megvalósitó rendszereket adatbanknak nevezzük. Az adatbank /bár körül vannak terminológiai viták/ két részből áll:

- az adatbázisból, mely az adatok speciálisan szervezett halmazát jelenti és
- az adatbázist vezérlő programrendszerből, mely biztositja az adatok bevitelét, visszakeresését, aktualizálását, ujjászervezését és egy esetleges hiba következtében szükségessé váló visszaállitását.

Az adatbank pontos definiciójához fel kellene sorolni még a vezérlő programokkal és adatszervezéssel - eléréssel szemben támasztott kritériumokat is. Ezek azonban egyrészt eléggé bonyolultak, másrészt nem állandók, az adatbank technika fejlődésével válnak mind igényesebbekké.

Az adatbanki tárolás a hagyományossal szemben a következő főbb előnyökkel rendelkezik:

- minimalizált adatredundancia, ami csökkenti a tároláshoz szükséges tárkapacitást, valamint centralizált tárolást, karbantartást és elérést biztosit;
- az adatbázis az adatokon kivül azok logikai kapcsolatát is tárolja, ami gyors és több szempont szerinti bonyolult visszakeresést tesz lehetővé /egyszerü programokkal/;
- az adat - program függetlenség különböző szintü megvalósitásai érzéketlenebbé teszik a programokat az adatszerkezet és az adatmennyiség változásaival szemben. Ennek eredményeképpen a kész rendszer software karban-

tartása szinte kizárólag az uj igények kielégitésére korlátozódik / a régi programok átirása nélkül/;

- az adatbázis vezérlő rendszert általában generálható programcsomag formájában forgalmazzák, ami jelentősen megkönnyiti a rendszer bevezetését.

Az ALAP-2 az ALAP továbbfejlesztéseként kidolgozott, IBM vagy ESzR számitógépeken DOS rendszerben üzemeltethető adatbázis-kezelő programcsomag.

Az ALAP-2 által megvalósitható adatszerkezet

A főleg mágneslemezes adathordozókon elhelyezkedő adathalmaz /az adatbázis/ az ALAP-2 által kezelt legnagyobb egység. Az adatbázis részadatbázisokra / az adatbázis önállóan üzemeltethető halmaza/, azok file-okra, a file-ok rekordokra, a rekordok szegmensekre, a szegmensek adatmezőkre bonthatók.

Az adatbázis /részadatbázis/ file-jait három fő tulajdonság jellemzi:

- a file tipusa: egy file lehet törzsfile vagy kapcsolati /lánc/ file. A törzsfile kulccsal is elérhető rekordokból áll. A láncfile egy vagy több törzsfile szegmensei közötti kapcsolatokat és a kapcsolatokat jellemző mennyiségeket tartalmazza;

- a file szervezési módja. A szervezési mód lehet szekvenciális /SAM/, közvetlen elérésü /DAM/, kontroll szekven-

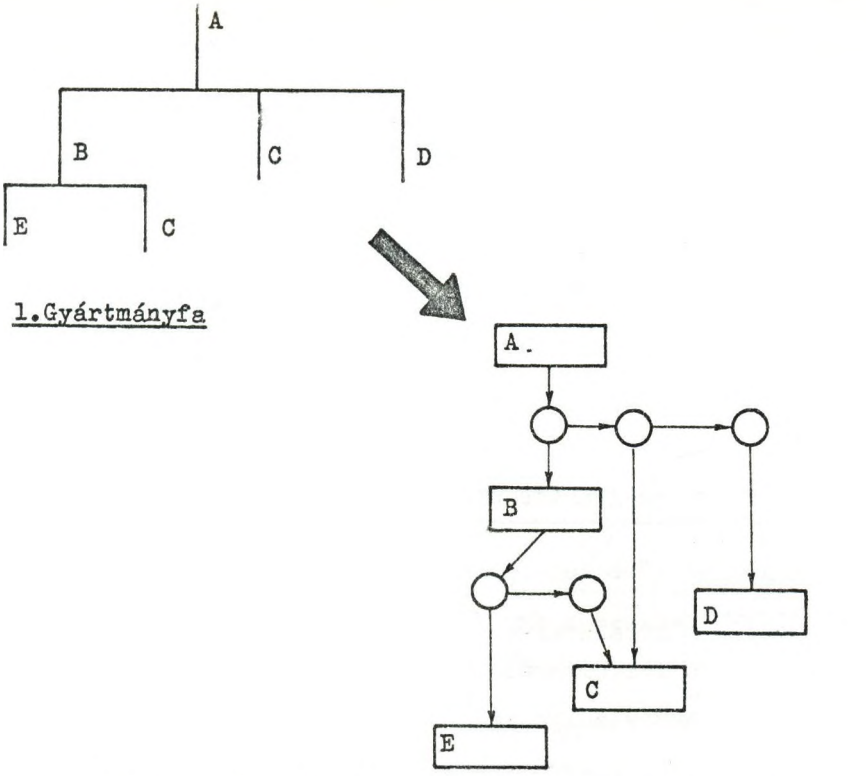
ciális /CSAM/ vagy különválasztott indexü /SIAM/. A SAM és DAM a DOS operációs rendszer által támogatott file-szervezési módszerek, mig a CSAM és SIAM az adatbázis kezelő segitségével valósitható meg. A CSAM és SIAM a DOS-ban gyakran használt index-szekvenciális /ISAM/ file-okhoz hasonló, de annál hatékonyabb megoldások. A láncfile-okat az ALAP-2 DAM szervezésre épitve modellezi;

- az adatbázis más file-jaival való kapcsolat, mely lehet inaktiv /nincs kapcsolat/, részlegesen aktiv /csak file-on belüli kapcsolat van/ vagy aktiv /a file kapcsolatban áll más adatbázis file-okkal/. A kapcsolatot a láncfile-ok biztositják az un. cimláncolási technika segitségével.

Az ALAP-2 programcsomag felhasználásával tetszőlegesen sok file-ból álló adatbázisok épithetők fel.

Az 1.ábra a cimláncolási technikát és a kapcsolódó file-okat illusztrálja egy olyan A gyártmány leképzésének példáján, ahol az A a B szerelvényből, a C és D anyagból áll. A B szerelvény az E és C anyagokból áll. Az adatbáziskezelő rendszer az 1. ábrán feltüntetett láncokon kivül az "anyagoktól" a "szerelvények" felé irányuló láncokat is felépit, de azok ábrázolása rontaná az ábra áttekinthetőségét.

A rekordok részben az adatbázis kezelő /system/, részben a felhasználó által definiált /user/ szegmensekből állnak.



1.Gyártmányfa

2.ALAP-2 leképezési elv

3.File-szerkezet

1.ábra. Darabjegyzék leképezés

A felhasználó által elérhető legnagyobb egység a szegmens. Egy adatbázis hivással több szegmens is elérhető. Minden felhasználó csak a számára kijelölt szegmensekhez férhet hozzá /ld. 2. ábra/ Az ALAP-2 titkositási-adatvédelmi opcióval is rendelkezik, melynek segitségével megakadályozható a szándékos illetéktelen beavatkozás vagy a véletlen adatroncsolás. A szegmenskezelés nagymértékben megnöveli az adatprogram függetlenséget. A logikai rekord változása ugyanis nem érinti a korábban definiált szegmensekre épülő programokat.

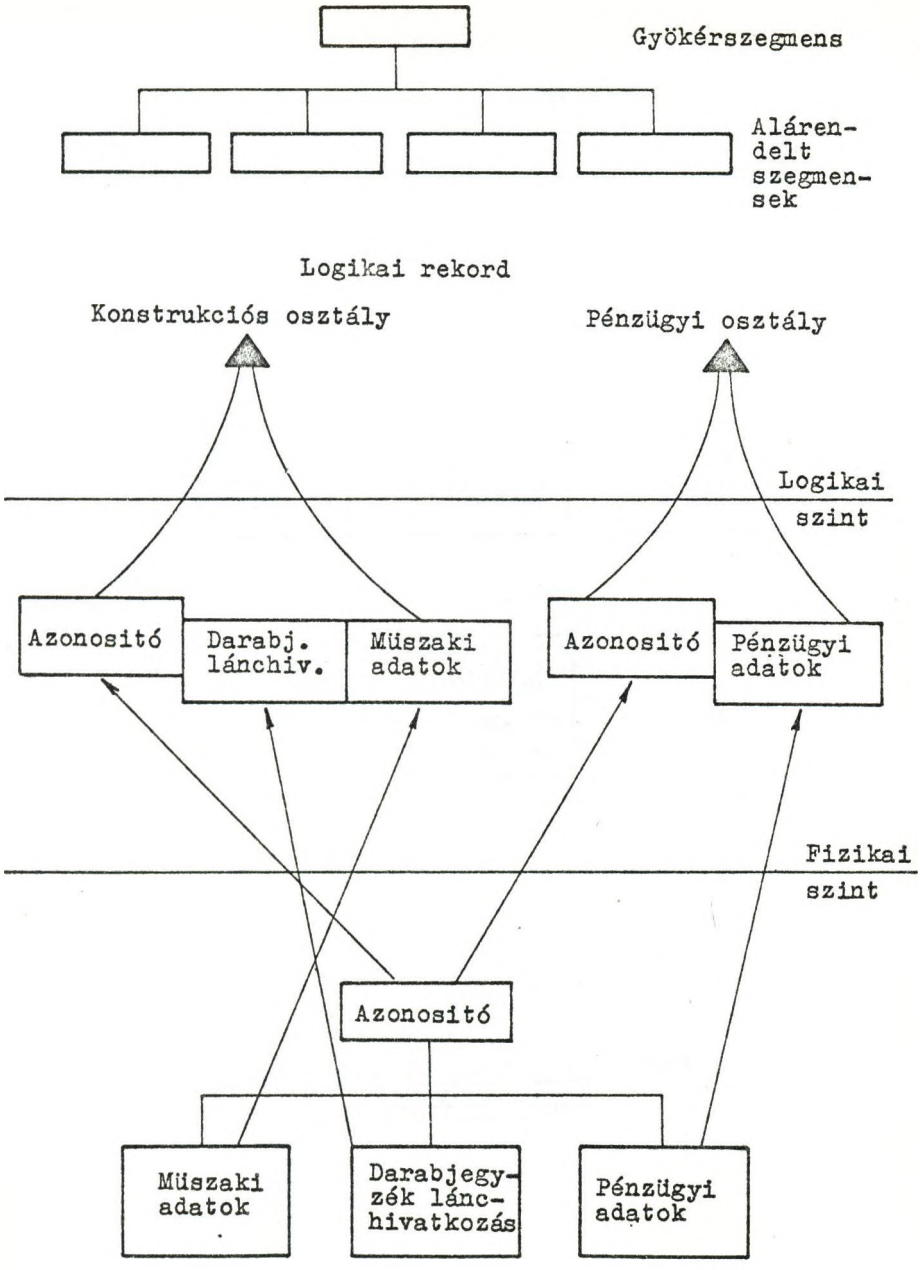
Az ALAP-2 programtechnikai felépitése

Az ALAP-2 főbb moduljainak kapcsolatát a 3.ábra mutatja. Az ALAP-2 rendszert egy /a felhasználó által irt, vagy bizonyos tipusu feldolgozásokra a rendszer által szolgáltatott/ főprogram aktivizálja egy

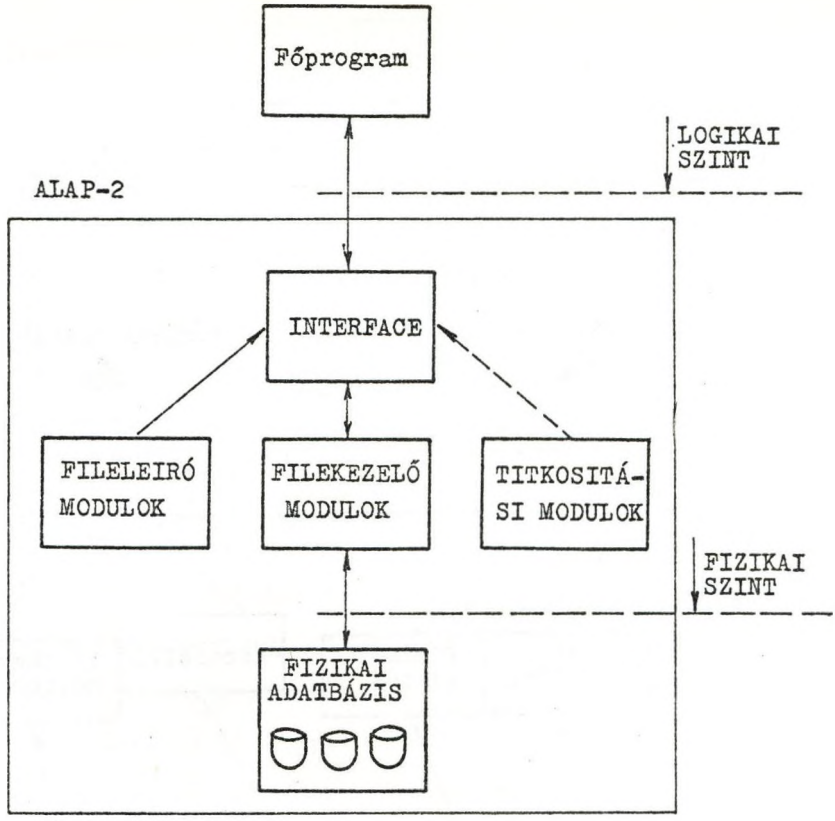
CALL ALAP-2 /paraméterlista/

tipusu utasitással. Az ALAP2 közös belépési pont assembler, COBOL és PL/I programok számára. A hivást az INTERFACE modul fogadja, és amennyiben az I/O kérés, akkor az INTERFACE a FILELEIRÓ és a TITKOSITÁSI modulokban tárolt információ figyelembe vételével továbbitja a különböző file-kezelő modulok felé. A filekezelő modulok vezérlik az adathalmazzal kapcsolatos I/O müveleteket. A beolvasott szegmenseket és a kivánt adatbázis funkció elvégzésének sikerességét mutató mezőket az INTERFACE adja vissza a főprogramnak.

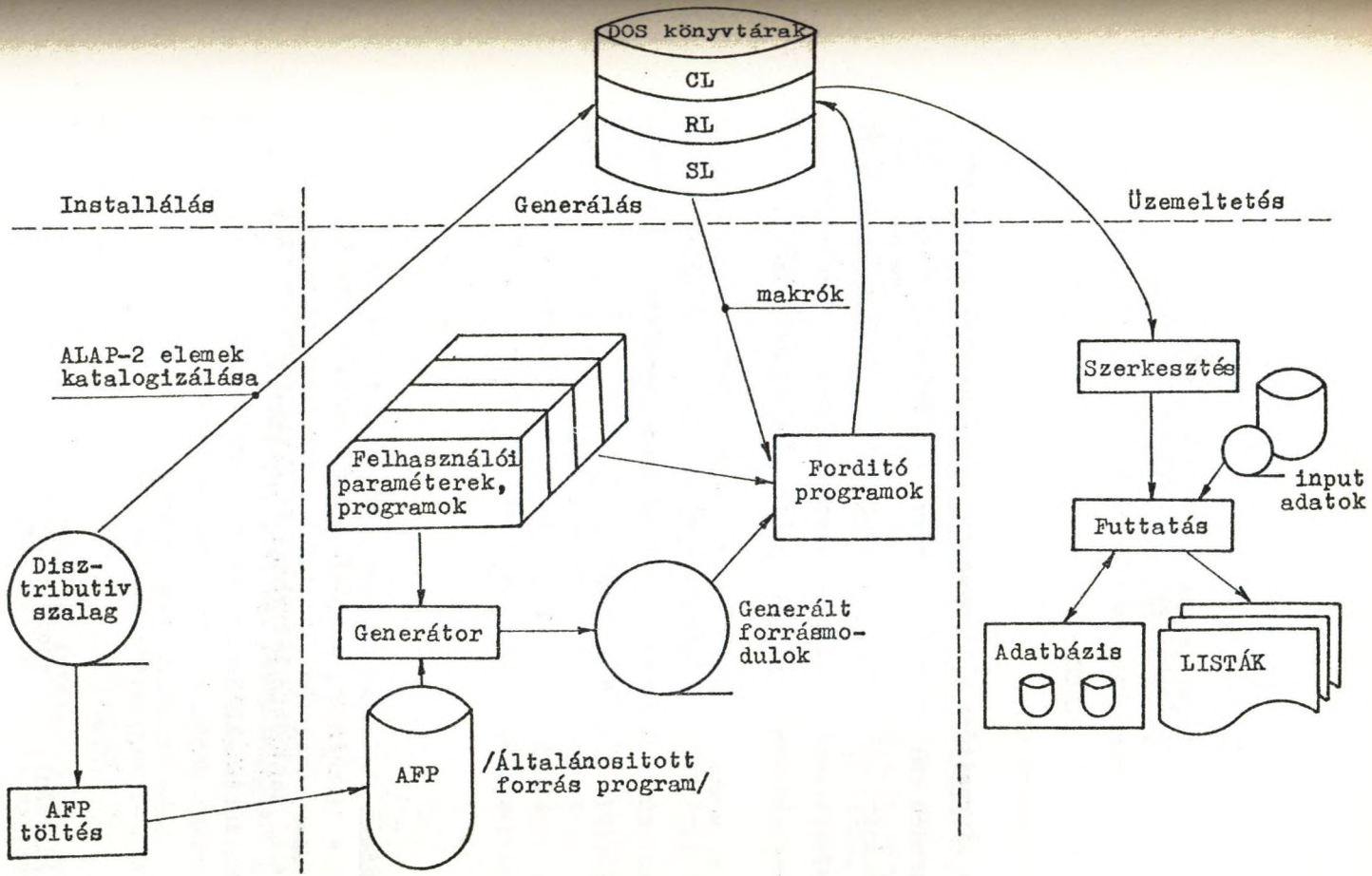
A felhasználó által tervezett adatbázist kezelő konkrét adatbázis vezérlő programokat az ALAP-2 eszközeivel kell generálni /részint a felhasználónak magának megirni/.



2.ábra. Szegmenskezelés



3.ábra. Az ALAP-2 adatbázis kezelő rendszer felépitése



4.ábra. Installálás,generálás,üzemeltetés

A programok létrehozása során:

- minden adatbázis file számára szükséges egy FILELEIRÓ modult generálni, mely megadja a file fizikai jellemzőit és a szegmens felosztást,
- minden részadatbázisra szükséges egy INTERFACE modult generálni, mely megadja az adatbázis konfigurációját /INTERFACE modult elegendő egy általánosat is generálni/;
- a titkositási - adatvédelmi opció használata esetén minden kulcsszóval /jelszóval/ rendelkező felhasználó számára kell egy titkositási modult generálni. Ez megszabja, hogy az adott kulcszóval mely file-ok, mely szegmenseinek irása vagy olvasása engedélyezett.

Az ALAP-2 programcsomagot a felhasználó disztributiv szalagon kapja. A rendszer "élesztését", generálását és üzemeltetését a 4. ábra illusztrálja.

IRODALOM

1. ALAP adatbázis Létrehozó és Aktualizáló Program, KG ISZSZI 1973

2. ALAP-2 Fejlesztési dokumentáció

3. Adatbázis Létrehozó, Aktualizáló és Lekérdező Programrendszer ALAP-2, Alkalmazói kézikönyv

KOMMUNIKÁCIÓS ESZKÖZÖK EGY TÖBBGÉPES RENDSZERBEN*

Kozma László - Simonfai László
Számitógépalkalmazási Kutató Intézet

Összefoglalás

A tanulmány egy szeizmikus alkalmazásokra kialakitott, Rlo bázisú többgépes rendszer felügyelő programjának kommunikációs eszközeit mutatja be. A kommunikáció kapcsolat-orientált, az adatátvitel ún. logikai csatornákon történik. A kommunikációs mechanizmus egységesen kezeli a perifériákat, a szekvenciális file-okat és a logikai csatornákat.

Az alkalmazási követelmények és a csatorna-csatorna összekötésen alapuló hardware konfiguráció áttekintése után a tanulmány összefoglalja az alapvető tervezési megfontolásokat, leirja a processzek rendelkezésére álló kommunikációs eszközöket és példákat mutat az eszközök használatára. Befejezésül a tanulmány kitér az implementálás legfontosabb kérdéseire majd rámutat a továbbfejlesztés lehetőségeire.

1. Bevezetés

Több számitógépből felépitett, szorosan vagy lazán csatolt multiprocesszoros rendszerek, hálózatok jelentősége egyre nő, hiszen sokasodnak azok a feladatok, amelyeket csak igy, vagy legalábbis gazdaságosan csak igy lehet megoldani [1]. A tanulmány egy szeizmikus alkalmazásokra kialakitott, Rlo bázisú többgépes rendszer felügyelő programjának /MUSCLE-MUlti System ControL Environment/ kommunikációs eszközeit mutatja be.

---

* A munka az Eötvös Lóránd Geofizikai Intézet /ELGI/ megrendelésére készült. A hardware fejlesztést is az ELGI végezte.

Szeizmikus adatgyüjtési alkalmazásoknál az alábbi fő feladatokat kell megoldani:

- nagy adattömeg összegyüjtése és redukciója, real-time üzemben /mérési ciklus: kb. 18 sec; adatgyüjtés: 6 sec-ig; a beérkezett adatok mennyisége ezalatt 3oo kByte/
- viszonylag bonyolult szürési és zajcsökkentési müveletek elvégzése;
- az adatgyüjtési ciklus vezérlése egyéb /ugyancsak real-time gyüjtött és feldolgozott/ paraméterek alapján.

Olyan teljesitményü real-time gép, amely mindhárom feladat elvégzésére alkalmas, nem áll rendelkezésre. Célszerünek látszik az elvégzendő feladatok megosztása több, együttmüködő gép között.

Egy többgépes rendszer mellett - amit szintén létre kell hozni - az alábbi indokok hozhatók fel:

- a feladat jól dekomponálható viszonylag önálló, egymással csak laza kapcsolatban álló részekre, és a részfeladatok a jelenlegi Rlo gépekkel megoldhatók.
- egyes részrendszerek korábban kifejlesztésre kerültek és a többgépes rendszerbe adaptálhatók;
- a több - szükség esetén önállóan is müködőképes - számitógép lehetőséget ad a megbizhatóság fokozására.

2. Konfiguráció

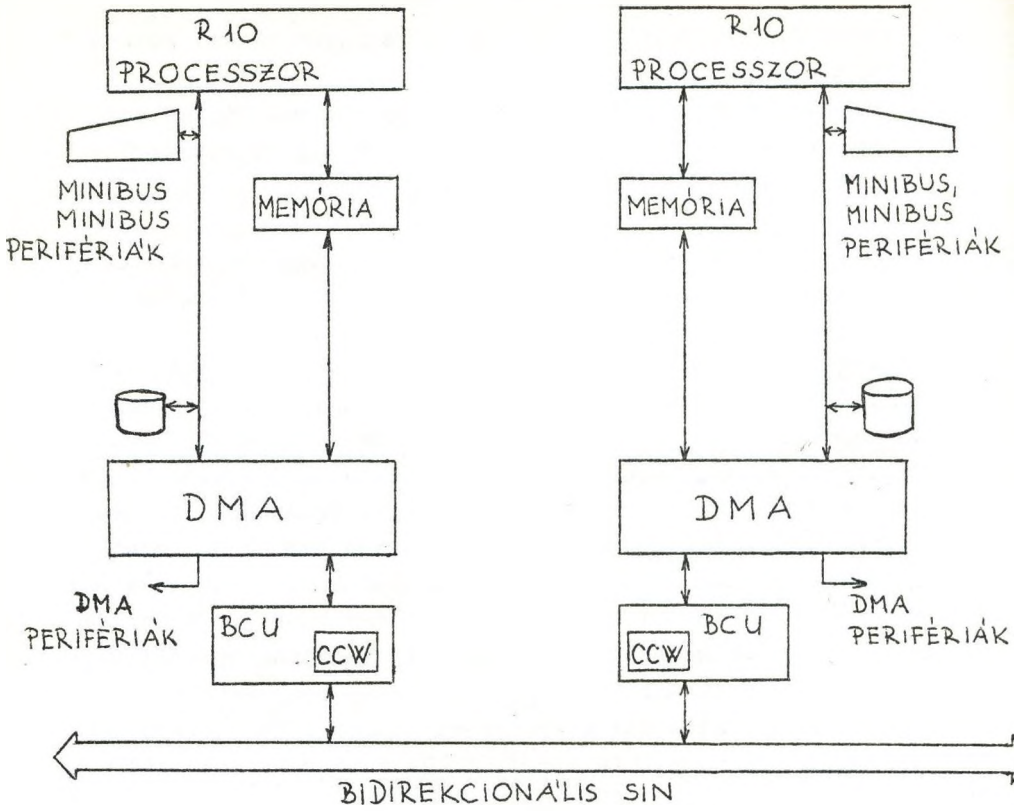
A többgépes kapcsolat megvalósitásához hardware fejlesztésre volt szükség. A nagy adattömegek és a real-time igények nagy sávszélességü összekötést tesznek szükségessé. /Pl.: közös használatú memória, valódi, multiprocesszoros rendszer/. Az Rlo korlátozott cimtartománya csatorna-csatorna összekötés mellett szól, mig a háttértárakon történő közbenső adattárolás felveti a közös hozzáférésü diszken keresztül történő csatolást. Nagysebességü csatorna-csatorna összeköttetés került kifejlesztésre, mert:

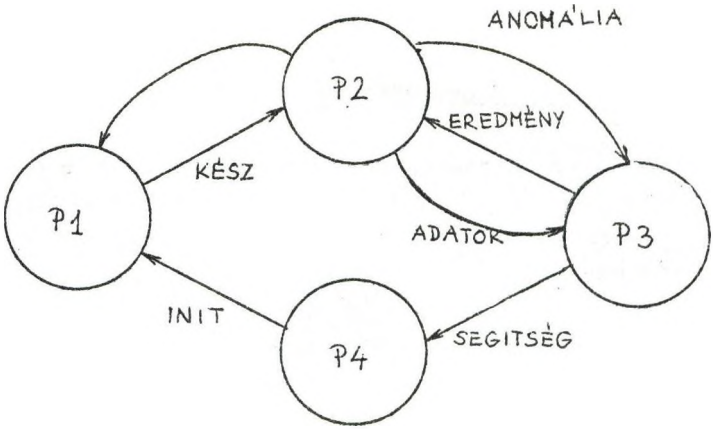
- kettőnél több számitógép összekötésére van szükség;
- az adatcserén kivül vezérlési és szinkronizálási feladatokat is meg kell oldani;
- rendelkezésre áll egy speciális fejlesztésü direkt memória hozzáférést biztositó egység /DMA/, amely az összekötéskifejlesztésének alapját képezte.

A feladatmegosztáson alapuló, homogén, többgépes rendszer konfigurációja az 1. ábrán látható. Maximum 4 Rlo számitógép kapcsolható össze egy nagysebességü, bidirekcionális sin segitségével. Valamennyi gép el van látva egy direkt memória hozzáférést biztositó berendezéssel /DMA/, amely lehetővé teszi adatblokkok cseréjét a speciális fejlesztésü DMA perifériák /pl. diszk, mágnesszalag, speciális tömbprocesszor, stb./ és a memória között. Ha a DMA egy sinvezérlőn /BUS control unit - BCU/ keresztül a közös sinre csatlakozik, az adat transzfer az összekötött gépek memóriái között történhet. A tényleges transzfert a hardware önállóan bonyolitja le, felhasználva a sinvezérlőben elhelyezett csatorna vezérlő szót /channel control word - CCW/, amely a szokásos vezérlési paramétereket tartalmazza: a transzfer irányát, az átviendő szavak számát és a memória cimet. A DMA transzfer inditásakor, ill. befejezésekor szükséges müveletek a MINIBUS segitségével végezhetők el. A MINIBUS-hoz a szokásos perifériák csatlakoztathatók. A rendszer megbizhatóságát az összekötő sin megbizhatósága erősen befolyásolja, ezért az összekötés kialakitásánál döntő szempont volt az egyszerüség.

3. A kommunikációs mechanizmus

A kommunikációs eszközöknek speciális célú rendszert /rendszereket/ kell támogatniuk. A szeizmikus mérésadatgyüjtést, és általában az alkalmazási feladatokat, együttmüködő processzek halmazaként tekintjük [2] . A processzek együttmüködésüket a processz kommunikációs mechanizmus /Interprocess Communication Mechanism-IPCM/ eszközeivel valósitják meg. A kommunikációs mechanizmussal szemben az alkalmazások az


R10
PROCESSZOR
MINIBUS
MINIBUS
PERIFÉRIÁK
MEMÓRIA
DMA
DMA
PERIFÉRIÁK
BCU
CCW
R10
PROCESSZOR
MEMÓRIA
MINIBUS,
MINIBUS
PERIFÉRIÁK
DMA
BCU
CCW
DMA
PERIFÉRIÁK
BIDIREKCIONÁLIS SIN


ANOMÁLIA
P2
EREDMÉNY
KÉSZ
P1
ADATOK
P3
INIT
P4
SEGITSÉG

alábbi általános követelményeket támasztják:

- támogassa nagy adatblokkok átvitelét /kB nagyságrend/;
- adjon eszközöket szinkronizálási feladatok megoldására;
- bár a processz populáció statikus /processzek dinamikus kreálására és törlésére nem kerül sor/ a software rekonfigurálásának lehetőségét biztositani kell.

A hardware-software környezetet két irányból korlátozza a lehetőségeket:

- a kommunikációs mechanizmust kisgépen kell megvalósitani;
- biztositani kell a kompatibilitást az Rlo RTDM, ill. RTDM bázisú, monitoraival. [15].

3.1 Tervezési alapelvek

A processz kommunikáció megvalósitásnak lehetőségei évek óta az érdeklődés előterében állnak mind a centralizált, [2], [3], [4], [5], [6], mind pedig a decentralizált, osztott intelligenciájú rendszerekben [7], [8], [9], [10], [11], [12], [13], [14]. Figyelembevéve a kisgépes környezetet, csak az alapvető eszközöket valósitottuk meg, a megvalósitás azonban az alkalmazási követelmények által megengedett mértékig általános. Az eszközök alkalmasak magasabb szintü funkcionális protokollok támogatására.[7], [8], [13].

A kommunikációs mechanizmus megtervezésekor az alábbi alapvető döntéseket hoztuk:

- kapcsolat-orientált kommunikáció.
Ahhoz, hogy a processzek kommunikálni tudjanak, kommunikációs utat kell kiépiteni [10], [12]. Ha a kommunikációs út kiépitése és az adatátvitel egyetlen lépésben történik, és a kapcsolat csak egyetlen üzenet átküldésének idejére marad fenn, akkor a kommunikáció üzenet-orientált [9]. Kapcsolat-orientált kommunikáció esetén a kommunikációs utat előbb létre kell hozni egy inicializálási fázisban, a tényleges üzenetközvetités csak ez után következhet. A kapcsolat időben állandó, a rendszer statikusabb. [7], [8], [13], [14]. A támogatni kivánt alkalmazásokban a processz populáció statikus, igy az állandó jellegü kapcsolat nem jelent hátrányt.

Ugyanakkor az egyes transzferek inditásakor szolgáltatandó információ mennyisége kevesebb, mert a kommunikációs út kiépitéséhez szükséges adatokat iniciáláskor már megadtuk. A MUSCLE rendszerben a kommunikáció kapcsolat-orientált, az információcsere logikai csatornákon keresztül történik. A logikai csatorna olyan névvel ellátott objektum, amely két processzt köt össze és egyirányú információ átvitelt tesz lehetővé a két processz között. A logikai csatornák neve globális a rendszerre nézve, a név egyediségét az alkalmazási rendszer tervezésekor kell biztositani. Amint a 2. ábrán látható, egy processz egyidejüleg több logikai csatornán kommunikálhat.

- globális és lokális nevek.
A logikai csatornák globális nevei és a processzek környezetére lokális I/O kapuk /port, Rlo terminológiában: operációs cimke/ összekötése inicializáláskor vagy futtatáskor történhet /statikus, ill. dinamikus asszignáció/. A megnyitott logikai csatornát a lokális I/O kapu azonositja.

- egységes hozzáférés.
A lokális I/O kapuk és a fizikai I/O eszközök egymáshoz rendelése ugyancsak iniciáláskor vagy futtatáskor történhet. A szekvenciális perifériák, a file-kezelőn keresztül hozzáférhető file-ok és a logikai csatornák kezelése az RTDM által megengedett mértékig egységes. [12], [13] . Azok a müveletek, amelyek egy adott periféria tipusra nincsenek értelmezve, transzparensek. Az IPCM megfelelő kialakitása biztositja, hogy transzfer egy logikai csatornán azonos módon bonyolódjon le, függetlenül attól, hogy a kommunikáló processzek fizikailag azonos vagy különböző számitógépen vannak. Az egységes hozzáférés előnyei belövéskor nyilvánvalóak, ugyanakkor a lehetőség jól kihasználható fail-safe müködés biztositására.

- pufferkezelés.
Tekintettel arra, hogy mind az egy egységként átviendő információ - üzenet - mérete, mind pedig a fizikai összekötés sávszélessége nagy, a transzfer közvetlenül a kommunikáló processzek területéről, ill. területére történik. Az IPCM tehát nem pufferel, nincs szükség az üzenetek csomagokba történő szétbontására, ill. összeállitására.

3.2 Kommunikációs szolgáltatások

A kommunikációs mechanizmus az alábbi eszközöket valósitja meg:

- információátvitel kezdeményezése.
  Az információátvitel kezdeményezésére alapvetően két parancs áll rendelkezésre:

  READ /operációs cimke, puffercim, szószám/
  WRITE /operációs cimke, puffercim, szószám/

  A transzfer akkor kerül végrehajtásra, ha a már megnyitott logikai csatornára illeszkedő parancspár került kiadásra. Ujabb átvitel csak akkor kezdeményezhető, ha az előző transzfer már befejeződött.

- eseménykezelés.
  Egy logikai csatorna nyitáskor adatcsatornának vagy eseménycsatornának deklarálható. Az eseménycsatorna szinkronizálásra használható, adat-transzferre nem kerül sor, bár a bekövetkezett esemény egy 7 bites információval kvalifikálható. Az eseménycsatorna kezelésére két parancs áll rendelkezésre:

  SEND EVENT /operációs cimke, esemény-kvalifikáló/
  RECEIVE EVENT /operációs cimke, esemény-kvalifikáló/

- transzfervég.
  Egy átvitelt kezdeményező parancsot kiadó processz az RTDM standard

  WAIT /operációs cimke, status/

  parancsának felhasználásával, várhat egy adott logikai csatornán inditott transzfer befejezodésére. A status visszajelzést tartalmaz a hardware és a softwaré által felfedett anomáliákról, ill. a normális müködésről. Hiba esetén az IPCM nem ismétel, az adott alkalmazástól függő célszerü stratégia kialakitása a processzek feladata.

- szimultán várakozás.
  Valamely erőforrást kezelő processz flexibilisen tudja kiszolgálni a beérkező igényeket, ha egyidejüleg több logikai csatornán kezdeményez transzfert, és szimultán várakozik arra, hogy valamelyik logikai csatornán a transzfer be-

fejeződjön. A szimultán várakozás megvalósitása az RTDM

WAITLIST /operációs cimke-status lista/

parancsával történik.

- feltételes transzfer.

Szeizmikus alkalmazásoknál a ciklikusan egymásután következő mérések elkezdését feltétlenül biztositani kell, még azon az áron is, hogy a folyamatban lévő mérés feldolgozása abbamarad. Hasonló megfontolások más alkalmazásokra is érvényesek. A feltételes transzfer két parancscsal kezdeményezhető:

TEST AND READ /operációs cimke, puffercim, szószám/

TEST AND WRITE /operációs cimke, puffercim, szószám/

A parancsok müködése azonos a READ, ill. WRITE parancsok müködésével, ha a kommunikáló partner processz az illeszkedő WRITE, ill. READ parancsot már kiadta - tehát a transzfer azonnal végrehajtható és csak a transzfer tényleges befejeződését kell kivárni. Egyebként a logikai csatorna reset-elődik és erről a partner processz értesitést kap. Az illeszkedő parancsokat az alábbi táblázat mutatja:

	R	TR	W	TW
R			X	X
TR			X	
W	X	X		
TW	X			

3.3 Logikai csatorna nyitása és zárása

Kapcsolat-orientált rendszerekben az adatforgalom megindítása előtt ki kell épiteni a kapcsolatot a kommunikálni kivánó processzek között. Ha a kapcsolatra nincs többé szükség, akkor meg kell szüntetni azt. A MUSCLE az alábbi szolgáltatásokat nyújtja:

- logikai csatorna megnyitása.

A kapcsolat kiépitésére és igy a logikai csatorna megnyitására az

OPEN /operációs cimke, logikai csatornanév, nyitási mód/

parancs szolgál. /A nyitási mód az átvitel lebonyolitására megengedett parancsot definiálja, ezáltal megadja a

transzferirányt és a castornát adat-, ill. eseménycsatornának minősiti./

Általános esetben a parancs hatására az IPCM ellenőrzi a globális-lokális névleképzést, feljegyzi a szükséges paramétereket, majd megkeresi azt a processzort, amely az illeszkedő OPEN parancsot kiadó processzt futtatja. Ha ilyen processzor található, és a nyitási módok illeszkednek, akkor a nyitás folyamata befejeződik. Egyébként az IPCM vár arra, hogy az illeszkedő OPEN parancs hatására a keresés újrainduljon. A keresési eljárás a globális nevek dinamikus leképzését és ezáltal a software rekonfiguráció lehetőségét biztositja.

A keresési folyamat lebonylitására előre definiált, ún. rendszer csatornák állnak rendelkezésre, a keresés a normál kommunikációs eszközökkel történik. Egy logikai csatorna tehát akkor kerül nyitott állapotba, ha mindkét processz kiadta az illeszkedő paraméterekkel ellátott OPEN parancsot. A rendszer visszautasitja azokat a parancsokat, amelyek nem nyitott logikai csatornára vonatkoznak.

- logikai csatorna zárása.
A logikai csatornák zárása a
CLOSE /operációs cimke/
parancs hatására történik. A logikai csatornára vonatkozó helyi bejegyzés törlődik, és a partner processzt kiszolgáló IPCM a zárásról értesitést kap. Logikai csatorna akkor kerül zárt állapotba, ha a kommunikáló partnerek mindegyike kiadta a CLOSE parancsot.

- kényszerzárás.
Ha egy processz exitál vagy abortálódik, az kommunikációs machanizmus gondoskodik a processz valamennyi nyitott logikai csatornájának lezárásáról.

4. A kommunikációs eszközök használata

A fejezetben néhány példa segitségével mutatjuk be a kommunikációs eszközök használatát. Az egyszerüség kedvéért ALGOL-szerü notációt alkalmazunk, a logikai csatornákra nevükkel

hivatkozunk, és csak a példa kapcsán lényeges paramétereket tüntetjük fel.

- transzfer már megnyitott logikai csatornán

```
        P1                                  P2
        .                                   .
        .                                   .
repeat write /csat/;                repeat read /csat/;
        .                                   .
        .                                   .
      wait /csat, status/                 wait /csat, status/
until status ≠ paritáshiba;         until status ≠ paritáshiba;
        .                                   .
```

- az eseménycsatorna használata

```
        P1                                  P2
        .                                   .
        .                                   .
   send event /ecsat,esemény/;         receive event /ecsat,esemény/
        .                                   .
                                            .
                                    wait /ecsat, status/;
                                    case esemény of
                                            .
                                            .
                                    i: begin ... end
                                    end;
```

- felhasználói deadlock

```
        P1                                  P2
        .                                   .
    write /csat1/;                      write /csat2/;
    wait /csat1, status1/;              wait /csat2, status1/;
        .                                   .
    read /csat2/;                       read /csat1/;
    wait /csat2, status2/;              wait /csat1, status1/;
        .                                   .
```

Minthogy a MUSCLE célrendszerek támogatására készült, általános deadlock detektáló és elkerülő eszközöket nem fejlesztettünk, mert ez a rendszer hatékonyságának csökkenéséhez vezetne. Az alkalmazási rendszert deadlock mentésre kell tervezni pl. a logikai csatornák vagy a várakozások

sorrendezett kezelésével, WAITLIST használatával vagy más módon. Ha a P2 processz müködését kissé megváltoztatjuk, az utoljára bemutatott példa deadlock mentéssé válik:

```
        P2
        :
    write /csat2/.
    read /csat1/;
        :
    wait /csat1, status1/;
        :
    wait /csat2, status2/;
        :
```

5. Implementálás

A MUSCLE kompatibilis az Rlo RTDM bázisú monitoraival [15]. A kommunikációs mechanizmus szolgáltatásai a monitor I/O rendszerén keresztül érhetők el, ez biztositja a logikai csatornák és fizikai perifériák egységes kezelését. A kommunikációs mechanizmust magát /IPCM/, amely

- elvégzi a parancsok ellenőrzését;
- multiplexeli a logikai csatornákat a fizikai összekötést megvalósitó sinre;
- visszajelzést ad a müveletek befejezéséről;
- gondoskodik az átvitelek lebonyolitásáról;
- létrehozza, ill. megszünteti a logikai csatornákat;

speciálisan kiképzett handlerként valósitottuk meg.

A konfiguráció valamennyi számitógépe tartalmazza az IPCM egy azonos példányát, együttmüködésük biztositja a helyes információcserét. [16].

A továbbfejlesztés során olyan lehetőségek megvalósitására kerülhet sor, mint a hozzáférés biztositása kitüntetett erőforrásokhoz /pl. konzolirógép, sornyomtató/ valamennyi processz számára, vagy a rendszerbetöltés megoldása valamennyi számitógépbe, egyetlen rendszerdiszkről.

6. Köszönetnyilvánitás

A szerzők köszönetüket fejezik ki Horváth Józsefnek és Békéssy Péternek a MUSCLE tervezésekor nyújtott segitségükért.

Irodalomjegyzék

1. Joseph, E.C.: Innovations in Heterogeneous and Homogeneous Distributed-Function Architectures.
Computer, March 1974. pp. 17-24

2. Dijkstra, E.W.: Cooperating sequential processes
Technological University Eindhoven.
EWD123 Sep. 1965.

3. Habermann, A.N.: Synchronization of Communicating Processes
CACM, June 1972. pp. 171-176

4. Brinch Hansen, P.: The Nucleus of a Multiprogramming System
CACM, Apr. 197o. pp. 238-242

5. Spier, M.J.; Organick, E.I.: The MULTICS Interprocess Communication Facility
2nd ACM Symposium on Operating Systems Principles
Princeton, Oct. 1969. pp. 83-91.

6. Balzer, R.M.: PORTS-A Method for Dynamic Interprogram Communication and Job Control
Proc. 1971. SJCC, pp. 485-49o

7. Carr, S.C.; Crocker, S.D.; Cerf, V.G.: HOST-HOST Communication Protocol in the ARPA Network
Proc. 197o. SJCC, pp. 589-597

8. Crocker, S.D.; Haefner, J.F.; Metcalfe, R.M.; Postel, J.B.: Function-oriented Protocols for the ARPA Computer Network
Proc. 1972. SJCC, pp. 271-279

9. Walden, D.C.: A System for Inter-Process Communication in a Resource Sharing Computer Network
CACM, Apr. 1972. pp. 221-23o.

10. Metcalfe, R.M. : Strategies for Inter-Process Communication in a Distributed Computer System
Proc. Symp. on Computer Comm. Networks and Teletraffic. Polytechnic Institute of Brooklyn, Apr. 1972. pp. 519-526

11. Akkoyunlu, E.; Bernstein, A.J.; Schantz, R.: Inter-Process Communication Facilities for a Network Operating System
Computer, June, 1974, pp. 46-55

12. Bernstein, A.J.; Ekanadham, K.: Inter-Process Communication in a Network
INFOTECH Network Systems Software, 1975, pp. 413-433

13. Retz, D.L.: Operating System Design Considerations for the Packet Switching Environment
Proc. 1975 NCC, pp. 155-160

14. Maekawa, M.: Interprocess Communication in a Highly Diversified Distributed System.
Proc. IFIP 77, North Holland, pp. 149-154

15. Horváth, J.: Real-time monitorok geofizikai feldolgozások támogatására.
Benyújtva a Programozási Rendszerek'78 konferenciára.

16. Simonfai, L.: Deadlock Problems in Multicomputer Interconnection System
Előadás a IV. Visegrádi Téli Iskolán.

ADATFELDOLGOZÓ ANSI-COBOL PROGRAMOK GENERÁLÁSA PROLOG NYELVEN

Láng Oszkárné
Számitástechnikai Koordinációs Intézet

A PROLOG nyelvet a Számitástechnikai Koordinációs Intézetnél 1977-ben kezdtük használni; a számitástechnika gyakorlati feladatainak automatizálására 1978-ban kezdtük alkalmazni, első alkalommal primér adatellenőrzést végző programok generálásában.

A számitógépes munkák egyik igen munkaigényes része az adatfeldolgozó programrendszerek megirása és tesztelése. Ezekben a rendszerekben több olyan feladat van, amely rutinszerü munkát kiván, kevés gondolkodással és sok irással, de az eltérő adatformátumok, táblarajzok stb. miatt minden programrendszernél uj program/ok/ megirását igényli. Az ilyen munkákat a gyakorlott programozók - éppen a rutinjelleg miatt - már nem szivesen csinálják, a kezdő programozók pedig a szükségesnél több személyi és gépi ráforditással végzik el. Ilyen feladat többek között a primer adatellenőrzés, melyre mindenütt szükség van, ahol adatrögzités utján keletkezett inputot kell feldolgozni. Itt mindig gondoskodni kell arról, hogy elirás, adatkihagyás vagy hibás lyukasztás folytán megengedhetetlen értékek ne kerüljenek be az adatállományba. Ennek a feladatnak az automatizálására irtam egy PROLOG nyelvü programot. Ez a program dialóg üzemmódban futtatható, s a terminálról beadott paraméterek alapján komplett, szintaktikailag és szemantikailag hibátlan ANSI COBOL nyelvü programot ir.

A PROLOG nyelvü program elinditása után a következő paramétereket kell terminálról beadni:

- programnév /a készitendő ANSI-COBOL program neve/
- betüjelzés annak meghatározására, hogy az ANSI-COBOL program a feldolgozás során minden tételt listázzon, vagy csak a hibásakat
- a különböző felépitésü rekordok száma
- betü- vagy számjelzés, amelytől függően az ANSI-COBOL program az input file-jából egységes, vagy változó hosszuságu rekordokat vár
- az ANSI-COBOL program inputjának minden elemi adatához: az adat hossza a hozzátartozó ellenőrzési szempontokkal. Ellenőrzési szempontként megadható numerikus, vagy alfabetikus vizsgálat, vagy konkrét érték/ek/re történő vizsgálat. Számokat tartalmazó adatmezőknél értékhatárokat is meg lehet adni. Az ellenőrzési szempontokkal lehet megadni azt is, hogy - több eltérő felépitésü rekord esetében - mely adatmező/k/ tartalma alapján lehet kiválasztani a rekordtipust.

A beadott paramétereket a program formai szempontból ellenőrzi, ha valamelyik nem felel meg a követelményeknek, akkor hibajelzést ad és ujra kéri a paramétert.

A felsorolt paraméterek beadásával egyidejüleg a PROLOG nyelvü program kijelöl egy file-területet, és ebbe irja az ANSI-COBOL nyelvü forrásprogramot.

Az ANSI-COBOL nyelvü program tulajdonságai:

- egy input és egy output szekvenciális lemezfile-t kezel
- további outputja a hibalista /sornyomtatón/
- az adatok nevét a PROLOG nyelvü program generálja a rekordtipus sorszámának és a rekordon belüli adatsorszámnak a felhasználásával /hányadik rekordtipusnak hányadik adata/
- A rekordok felépitését a terminálról beadott adathosszok határozzák meg. Az input file esetében a rekord egy egységes hosszra kiegészülhet /paramétertől függően/.

- A hibalistán minden oldal legfelső sorában egy fejsor jelenik meg, amelynek tartalma:
  programnév, HIBALISTA felirat, napi dátum, folyamatos lapszám.

- A hibalistán a rekord minden adata megjelenik az input rekord adatainak sorrendjében, egymástól 2-2 üres pozicióval elválasztva. A hibás adatok alatt "*" megjelölés van.

Mivel -tapasztalatom szerint - egy software termék alkalmazásától gyakran elriasztja a felhasználókat a tul sok és bonyolult paraméterezés, ezért a program megirásánál arra törekedtem, hogy kezelése egyszerü, könnyen elsajátitható legyen. Ennek érdekében viszont le kellett mondanom néhány olyan ellenőrzésről, amely viszonylag ritkábban fordul elő, de a programgenerátor kezelését megnehezitené. Nincs például beépitve olyan ellenőrzés lehetősége, hogy "ha A adat értéke x, akkor B adat értéke y kell, hogy legyen". A viszonylag ritka, vagy egészen speciális ellenőrzések egyszerü módositásként beépithetők az ANSI-COBOL forrásprogramba. Az ilyen kiegészitések végrehajtásának megkönnyitésére igyekeztem a generált programot ugy felépiteni, hogy abban egy kevéssé gyakorlott programozó is könnyen kiismerhesse magát. Ez a felépités a következő:

1. A program indulásakor szükséges teendők /file-ok nyitása, kezdőértékek beállitása stb./

2. Programvezérlés, mely az input file végének eléréséig ciklikusan ismétlődik:

   - beolvassa a soronkövetkező rekordot
   - ha az input file-nak több rekordtipusa van, eldönti, hogy az éppen beolvasott rekord melyik tipus
   - felhivja a /rekordtipusnak megfelelő/ feldolgozási rutint.

3. Feldolgozási rutin
   - végrehajtja az adatellenőrzéseket, és eltárolja az esetleges hibajelzéseket

- ha van a rekordban hibás adat, vagy ha ugy intézkedtünk, hogy minden beolvasott tétel nyomtatásra kerüljön, felhivja a listázó rutint
- ha az input rekord hibátlan, kiirja /változatlan formában/ az output file-ra.

4. Listázó rutin
   - a rekord valamennyi adatát kinyomtatja
   - ha valamelyik adatra hibajelzés van, azt is kinyomtatja
   - számolja az egy lapra kiirt sorokat és szükség esetén felhivja a lapváltó rutint.

5. Lapváltó rutin
   - lapot vált, és kiirja a lista fejsorát

6. Programfutás befejezése

A feldolgozási és listázó rutin /3. és 4./ annyiszor és olyan sorrendben fordul elő, ahogy az input file különböző rekordtipusai vannak meghatározva.

A fentiekben leirt PROLOG nyelvü programgenerátor ez év március végén készült el, igy ezen ismertetés irásakor még nem tudok széleskörü használatáról beszámolni, de a Számitógép Laboratórium Általános Alkalmazások Osztályán áprilisban indult munkáknál már sikerrel alkalmaztuk. Továbbá elkezdtem kidolgozni egy olyan változatát, mely a terminál helyett egy szekvenciális lemezfile-ból olvassa be a paramétereket; ebben az esetben az adatmezőknek nemcsak a hosszát, hanem a nevét is meg kell adni. Ez bizonyos többletmunkát jelent, de előnye, hogy

- a generált adatnevek helyett az adat tartalmára utaló nevek alkalmazása megkönnyiti a programban való tájékozódást

- a PROLOG program inditása előtt felül lehet vizsgálni a paramétereket, igy pillanatnyi figyelmetlenség nem veszélyezteti a sikeres futást

- a terminál foglalás ideje lényegesen rövidebb

- a file-ban tárolt paramétereket a tervezés alatt álló további PROLOG nyelvü programgenerátorok is felhasználhatják az azonos felépitésü file-oknál.

A már elkészült programgenerátor - ugy vélem - biztató kezdet ahhoz, hogy az adatfeldolgozó számitógépes rendszerek sok irásmunkát igénylő programozását jelentős mértékben egyszerüsitsük. Terveink között szerepel, hogy hasonló elvek alapján megkiséreljük megvalósitani további PROLOG nyelvü programgenerátorok elkészitését a következő célok megoldására:

- listázás
  tételesen, vagy összevontan, tetszés szerinti szamban összegfokozatok képzésével

- karbantartás
  adatállomány kiegészitése uj adatokkal, ill. a bentlévő adatok módositása, törlése

- szétválogatás
  adatállomány szétirása több file-ra, adattartalom alapján.

Az ilyen jellegü programok generálás utján történő előállitása nemcsak a programozók munkáját könnyitené meg, hanem a programrendszerek elkészitésének időtartamát is jelentősen megrövidithetné.
Budapest, 1978. április 24.

ADATFELDOLGOZÁSI RENDSZEREK R-20-RA TÖRTÉNŐ KONVERZIÓJÁNAK PROGRAMOZÁSI TAPASZTALATAI

Légár János - Radvánszki László
MÜM

Intézetünk a Munkaügyi Minisztérium Számitástechnikai Intézete. Alaptevékenységünk közé tartozik az oktatás, kutatás és a szolgáltatás. A szolgáltatás döntően két terület felé irányul:

- Munkaügyi Minisztérium igényeinek kielégitése
  ide tartoznak a munkaüggyel szorosan összefüggő területek /bér, jövedelem, létszám, szabályozórendszer, stb./
- vállalatok igényeinek kielégitése
  döntő mértékben adatfeldolgozással kapcsolatos területek /számlázási rendszerek, elszámolási rendszerek, vertikumok közti termelési, értékesitési kapcsolatok, lineáris programozás, hálótervezés, stb./

A feladatok elvégzésére a következő eszközbázis áll rendelkezésre:

- ICL 1905/E
- R-20
- Honeywell-Bull 6000 ÁSzSz keretében
- Redifon-Seecheck

A minisztériumi munkák döntő többsége az ICL és Honeywell számitógépeken, mig a vállalatok felé történő szolgáltatások az ICL és R-20-as gépeken történnek.

A következőkben csak az R-20-as adatfeldolgozó rendszerek software oldalával kivánunk foglalkozni, melyek eltérnek az eddigi gyakorlatunktól.

- Redifon ismertetése

Részletesebben kivánjuk bemutatni a Redifon kisszámitógépet, mert tudomásunk szerint e gép lehetőségei kevésbé ismertek.
Ez a kisszámitógép adatelőkészitési és kisebb volumenü programozási munkák végzésére, valamint mágnesszalagos konverziók /7 csatornáról 9 csatornára és vissza/ végzésére szolgál. Nagyon jól használható programok tárolására, javitására és természetesen futtatására.
A számitóközpontok egyik legnagyobb gondja a nagymennyiségü adatok rögzitése, a feldolgozáshoz szükséges előkészitése, ellenőrzése, javitása. A hagyományos adatelőkészitési közismert hátrányai közül ki kell emelni azt, hogy hibamentes adatelőkészitést gyakorlatilag nem lehet elérni, ami a javitási ciklusok közbeiktatását vonja maga után, s ezzel jelentősen megnő az adatelőkészités tényleges időigénye.
A MüM SZÁMTI esetében a tipikus problémákat a több gépre történő adatelőkészités és programlyukasztás még tovább bőviti.
A Redifon kisszámitógép ezeket a problémákat viszonylag könnyen, gyorsan és jó hatásfokkal tudja kielégiteni.
Seecheck operációs rendszerrel rendelkezik, ami un. több feladatos virtuális operációs rendszer. A tárban egy nagyon kisméretü residens felügyelő modul található, mely megvalósitja a virtuális tárkezelést. A programok lapozott formában kerülnek be az operativ tárba, egy lap mérete 240 byte. A tárban mindig csak az aktuális lap van fizikailag jelen, mig a ténylegesen lekötött tárméret dinamikusan változik.
Alkalmas input, vagy mezőszintü, rekordszintü, batch szintü és output szintü programozás elvégzésére.

Ezen feladatok végzése mellett el tudja látni a programok felvitelét, forrásnyelvi szerkesztését, karbantartását és különböző gépek felé szétosztását.
Előadásunkban igyekszünk bemutatni az ICL és R-20 számitógépek által nyujtott rendszerbeli különbségeket, melyekre a teamek tervezésénél, valamint a munkák végzésénél okvetlenül figyelemmel kell lenni.

- Teamek tervezésének problémái:

1. Nagyobb emberigény

Tervezési gyakorlatunkban problémát jelentett és még ma is időnként jelent a szükséges emberigény meghatározása. Az ICL-lel összehasonlitva 1-2 emberrel több kell R-20-as projectek megvalósitásához. Ezt a tényt a következőkkel lehet magyarázni:

- kisebb, de több programra kell a rendszert bontani

Ezt részben a gépi meghibásodások gyakorisága, másrészt a lassubb müveletvégzés indokolja.

- a tesztelések azonos nagyságu és nehézségi foku programoknál kétszer hosszabbak az R-20 esetében
- koncentráltabb és több munkát igényel a team és a project irányitása

2. Munkamegosztás

Az ICL-es gyakorlat szerint a team tagjai feladatukat határidőre nagy intenzitásu munkavégzéssel tudták teljesiteni. Általános gyakorlat szerint lehetőség szerint több munkában nem dolgoztak. Az R-20-nál ez megfordult az előbbiekben elmondottak alapján. Kisebb intenzitással, több emberrel, több teamben dolgoznak egyidejüleg.

3. Határidők, futtatások

A DOS operációs rendszerben nem multizunk, igy értelemszerüen a programok átfutási ideje hosszabb. Napi két fordulónál több nem igen lehetséges, gyakorta még ez sem teljesen biztos. Ezért a határidők megállapitásánál ezt feltétlenül figyelembe kell venni. A szellemi és gépidő-tervezés aránya a szellemi felé tolódik el, valamint a rendszerek tapasztalataink szerint elég sok un. "utógondozást" igényelnek, ami persze az esetek jelentős részében apróbb belenyulásokat jelent.
Saját R-20-unk alapján,de más számitóközpontok véleménye szerint is a rendszeresen ismétlődő rövid időközönkénti /1-2 napos/ futtatás elég sok bizonytalanságot rejt magában.
Általában a futtatások relative sok előmunkálatokat követelnek, többször és alaposan kell ellenőrizni a készült outputokat.

- Az átirással kapcsolatos software problémák

1976-óta, R-20-as gépünk üzembeállitásától, három nagyobb volumenü software adatfeldolgozó rendszert tettünk át ICL-ről R-20-ra. Ezek a következők voltak:

- OMKDK, külső munkavállalók nyilvántartása és bérelszámolása
- Dunai Vasmü, számlázási rendszer
- Izzó, ABC analizis

Mindhárom rendszer évek óta az ICL-en futott jól, megbizhatóan. R-20-ra történő átirását az emlitett vállalatok kértek, hogy az akkortájt megkapott ESZR gépükön futtathassák a saját rendszerüket.

Az emlitett rendszerek közül az első kettőnél a szervezési javaslat átirásával kezdődött a munka. Ennek fő oka, bármilyen furcsán is hangzik, a két gépen a nyomtatható sorok szélességének különbsége volt. A rendszereink táblatervei többségükben kihasználták az ICL gép sornyomtatójának mind a 160 pozicióját és igy sok táblát át kellett tervezni, néhányat pedig két külön menetben elkésziteni. Az R-20-hoz csatolt két sornyomtatónk közül egy 160 pozíciót tud nyomtatni, de ezt a DOS operációs rendszerben nem lehet kihasználni.
Teljesen át kellett irni rendszereink gépi folyamatábráját. Az okok a következők voltak:

- A különleges file szervezési eljárások a DOS-ban nehézkesek, nem hatékonyak és nem utolsósorban nem is hibátlanok. Ezért igyekeztünk mellőzni az indexelt szekvenciális és a random file-okat az átirt rendszereinkben.
- Az R-20 igényli, hogy file-jaink könnyen ujra előállithatók legyenek, ezért a check-pointok száma lényegesen több, mint az eredeti rendszerekben.
- A tulságosan nagy programokat szétszedtük több, kevesebb funkcióju programra. Ez egyrészt védelem a rendszer majdani futtatásakor egy esetleg bekövetkező hiba kihatásai ellen, másrészt takarékosság a gépidővel a program és a rendszerteszt alatt. Gépünk központi egysége sokkal lassubb, mint a nagy gépeké és a programteszt alatt rengeteg gépidőt emésztene fel a nagy programok sokszori forditása. A programok szegmentálását csak kevés esetben alkalmaztuk az első ok miatt.
- Az uj rendszerekben sok listát RPG-vel tudtunk megoldani. Ez a lehetőség az ICL-en nem volt.

- A nagytömegü adatok felvitelének módszere időközben megváltozott. A key-to-disc rendszerünk módositotta az adatfeldolgozás inputját és átvett néhány adatellenőrzési funkciót is.

A már emlitett és a következőkben felsorolandó okok miatt teljesen uj programokat kellett irni

- Fortran háttérbe szorulása, COBOL és PL/I nyelv előtérbe kerülése.
  Mig az ICL-re sok rendszer döntő mértékben Fortranban készült, az R-20-ra a Cobolban és PL/I-ben tudók kerültek elsősorban számitásba. Természetesen ez nemcsak programozási nyelv egyszerü váltását jelenti, hanem a rendszer egész felépitését, összefüggéseit is érinti. A PL/I-et ritkán használtuk, talán a nagy forditási gépidőigénye és a megirt program nehéz tesztelhetősége miatt.
- Listázó Cobol programjaink szinte kivétel nélkül a Riport Writter-t használták. Az ICL-en hasonló funkció nem volt, az uj lehetőség nagyban könnyitette a programozók munkáját.
- Az adatfeldolgozási rendszerek fejlesztésénél ritkán lehet csak egyféle programozási nyelvet alkalmazni, vegyesen használjuk a nyelveket, ami a programozók ismeretétől és a megoldandó feladattól függ elsősorban. Nem volt célszerü és nem is tudtuk mindenhonnan kiküszöbölni a Fortran használatát, hiszen egyrészt a rendszer átszervezése /átirása/ rengeteg problémát vet fel, másrészt a programozók összetétele sem teszi ezt mindig lehetővé.
  Általában a file-kezelés tapasztalataink szerint egyszerübb Cobolban, mig a számitási, nyomtatási rész egyes esetekben Fortranban gyorsabb, könnyebb.

Fortran compilerünk az indexelt szekvenciális és random szervezésü file-okat nem tudja kezelni, ami pedig gyakori a feldolgozások során. Ezért született az az elhatározás, hogy a két programozási nyelvet összekapcsoljuk, mindkettő hivható legyen a másikból szubrutin formában.
Az alkalmazás során kedvező tapasztalatokat szereztünk a két nyelv összekapcsolásáról, a feladatok zökkenőmentesen megoldhatók lettek.
Nehezebb dolgunk volt a Cobol - PL/1 programokkal. Csak többszöri próbálkozás után, Assembler nyelvü szegmens közbeiktatásával sikerült. Közvetlenül azért nem kapcsolható össze, mert a PL/1 egy plusz funkciót vár a DSA táblától, amit Cobol programból nem lehet kielégiteni, csak Assemblerből pl. az IJKSZCN szubrutin által.

A programok tesztelése során az addigi gyakorlatunkhoz képest uj módszereket választottunk.

- Minden elkészitendő rendszerhez hozzárendeltünk privát könyvtárfile-okat. A futtatható programokat azonnal katalogizáljuk az abszolut modul könyvtárba, mert ezzel egy-egy forditást esetleg megtakarithatunk. Nehézséget okoz, hogy a PL/I forditó nem tud közvetlenül a forráskönyvtárhoz fordulni.
- Kihasználjuk a kis számitógépen történő programfelvitel lehetőségét.

Az átirt rendszereink nagy tömegü törzsadatállománnyal dolgoztak az ICL gépen. Ezek a törzsadatok csak mágneses periférián voltak tárolva, naprakészitve. A vállalatok nem tudták volna az előállitásukhoz szükséges bizonylatokat rendelkezésünkre bocsátani. A Redifon kisszámitógép az ada-

tok fizikai és logikai konverziójára tökéletesen megfelelt. A 7 csatornás mágnesszalagból 9 csatornásat készitett, 6 bites karakterekből 8 bites byte-okat, az ICL szabvány rekordformátumból DOS által elfogadhatót. A teljesség kedvéért meg kell jegyezni, hogy 6000 rekordos állományt, az emlitetteken kivül bonyolult adatkonverziót is végezve, 8 óra alatt másolt át.

- Következtetések

A rendszereket soha nem lehet egy az egyben átvinni más tipusu számitógépre. Ennek a felsoroltakon kivül oka még az is, hogy a vállalatok életében az eltelt idő mindig több változást hoz és a számitástechnikai ismereteink is jelentősen bővülnek. Ezeket is figyelembe vettük az átirás folyamán, s igy az elkészült rendszereink szinvonala, gépidőigénye a lassubb és kisebb számitógép ellenére változatlan maradt.
A rendszer elkészitése, programozása nagyobb, több energiát igénylő feladat, de a kész rendszer futtatásának paraméterei kedvezően alakulnak.

A BHG-BAN KIFEJLESZTETT PARAMÉTEREZHETŐ TÁBLÁZÓ PROGRAM /PTPWL/ HATÉKONYSÁGA AZ ADATFELDOLGOZÓ RENDSZEREKBEN

Latkoczy Zsuzsanna - Várhegyi Magdolna - Várhegyi István
BHG Hiradástechnikai Vállalat

Mint ismeretes a hazai gépi adatfeldolgozás jellegzetessége, hogy a feldolgozások eredményeit táblázni kell. Mivel az eddigi gyakorlati tapasztalatok azt bizonyitják, hogy a számitástechnika fejlesztése hatékony és eredményes, számolnunk kell azzal a ténnyel, hogy egyre több és nagyobb információ tömeget kell rugalmasan, gyorsan kezelni. A nagytömegü adatállományok feldolgozása mind bonyolultabb feladatok elé állitja a számitástechnikai szakembereket. Minden olyan vállalatnak, amely gépi adatfeldolgozást folytat szemelőtt kell tartania azokat a felhasználói igényeket, amelyek az információ gyors, pontos biztositását kérik.

Az előbb emlitett tények ismeretében merült fel egy olyan rugalmasan kezelhető program elkészitése, amely a felhasználó igényeit adatfeldolgozói területen ki tudja elégiteni és nem kiván nagy programozói munkát. Jelenleg a táblak készitésére az RPG célorientált nyelv van elterjedve, de hozzá kell tenni, hogy nem széleskorben. Nem kivánom részletezni milyen okok azok, amelyek miatt nem terjedt el az RPG, de szeretnék kiemelni néhány általam ismert és elsődlegesnek tartott problémát:

1./ Minden tábla elkészitéséhez egy uj program megirása szükséges. A tábla formájának bármely kisméretü változtatása a program változtatásával jár.

2./ A gyakorta kért információkat táblázó job-kat katalogizálni kell, amelyek a könyvtár kapacitását csökkentik.

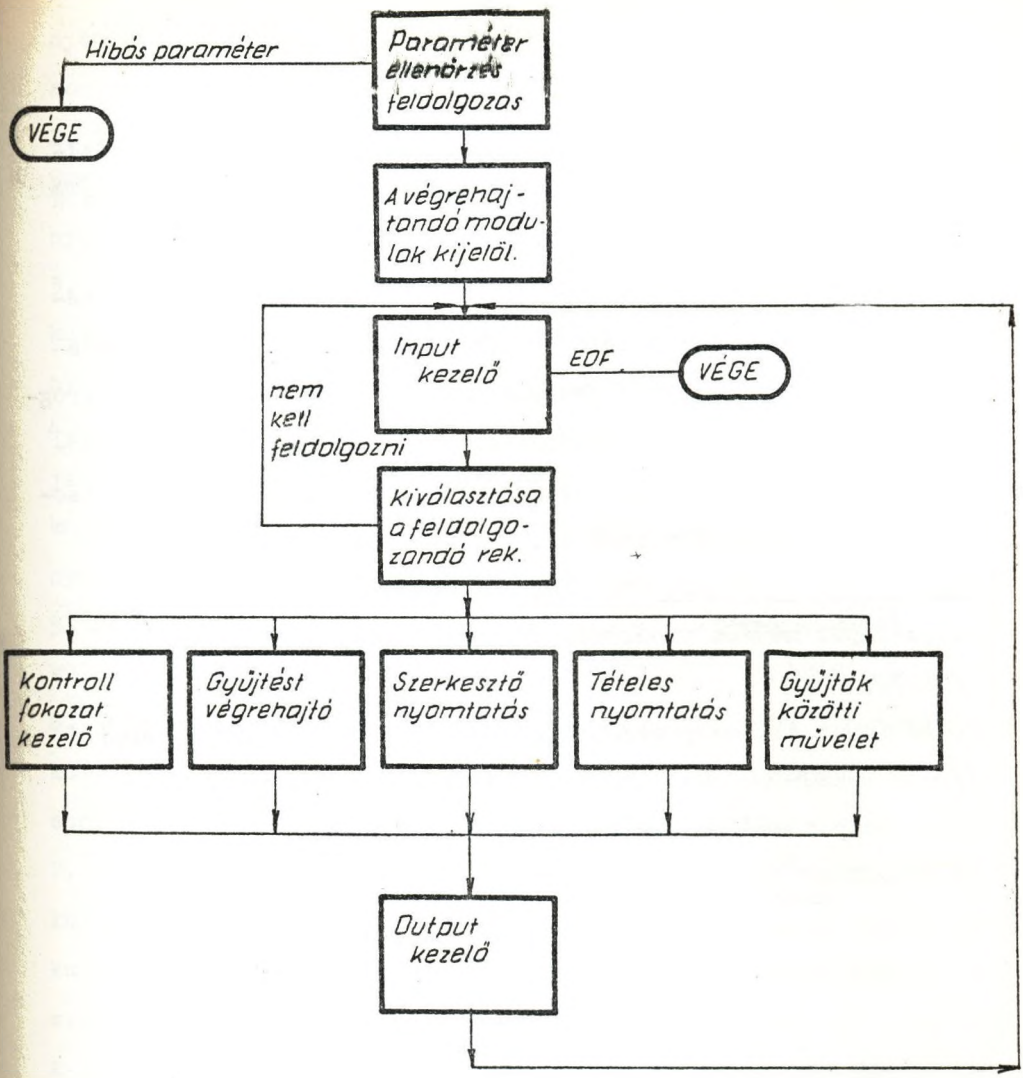
3./ A célorientált nyelvek hatékony kihasználásához behatóan kell ismerni a nyelv lehetőségeit.

Vállalatunknál a software fejlesztési team egy évet dolgozott a probléma megoldásán. Ma már nyugodtan állithatjuk, hogy a feladat első részét sikeresen megoldottuk. A gyakorlati tapasztalatok azt mutatják, hogy a feladatra orientált paraméterezhető programoknak a használata megbizható, gyors és esztétikailag is megfelelő eredményt biztosit.

A program PL/1 nyelv lehetőségeit felhasználva készült és ebből következik, hogy olyan moduláris szerkezetekből épül fel, amelyek önálló funkcionális modulként beépithetők más programokba is. / lsd.l.sz. ábra /

A következő szolgáltatasokat tudja nyujtani a felhasználónak a PTPWL2:

- Választható input/output médium.
- Maximálisan 16 kontrollfokozat változásának figyelése.
- Minden kontrollfokozat szinthez maximálisan 16 gyüjtö rendelhető.
- Tételes karakteres kiiratás, maximum 16 egyseg.
- Tételes szerkesztett numerikus kiiratas. Maximum 16 numerikus adat.
- Gyüjtök közötti müveletvégeztetés.
- Gyüjtö es adott konstans közötti müveletvégeztetés.
- Standard fejlécsor lapszamlálással.
- Tetszőleges felhasznaloi fejlécsor kiiratása.
- Csak adott azonositóju rekordok feldolgozasa.



1. ábra A program szerkezete

A modulok közötti végrehajtási sorrend minden esetben a feladattól függő, a végrehajtási idő legoptimálisabb kihasználása miatt.

- Adott azonositóju rekordok elhagyása a feldolgozásból.
- A feldolgozott rekordok leszámolása.

A vezérlő paramétereknek összesen 13 fajtája van:

A BHG-ban a PrPWL2 bevezetése lehetővé tette, hogy az adatfeldolgozással foglalkozó programozónak nem jelent megterhelést a táblak elkészitése és módositása. A programozónak több ideje és energiája marad más irányu feldolgozások megoldására. A paraméterek formátumát szeretném ismertetni, hogy ezek ismeretében bizonyitani tudjuk a program könnyen kezelhetőségét. A program paramétereit két fő csoportba lehet sorolni, az egyik csoport az input/output médiumra vonatkozó paraméter, a másik csoport a táblavezérlő paraméter.

Input/output médiumra vonatkozó paraméterek:

1. Az input médium tipusának meghatározása:

// INPUT DISK
TAPE

2. Az input állomány méretének meghatározásának paramétere:

// INPUT RECSIZE/nnn/BLKSIZE/nnnn/ kötelező

3. Az output médium tipusának meghatározása:

// OUTPUT TAPE
PRINT

4. Kivánt lapszámtól való táblázás meghatározása:

// OUTPUT PAGE/nnnnnn/

5. Adott azonositóju rekordok feldolgozása:

// OUTPUT YES/nnn//zz/= az adott azonositó

6. Adott azonositóju rekordok elhagyása a feldolgozásból:

// OUTPUT NO/nnn//zz/= az adott azonositó

Táblavezérlő paraméterek:

1. Kontrollfokozat vezérlés "K" kulcsjelü paraméterrel.

kulcsjel,sorszám,elhelyezkedés a rekordban,elhelyezkedés a

nyomtatási mezőben,hossza a kontrollfokozatnak,lapváltás vagy soremelés mutató.

2. Gyüjtő vezérlés "G" kulcsjelü paraméterrel.

kulcsjel,sorszám,elhelyezkedés a rekordban,elhelyezkedés a nyomtatási mezőben,hossz,tizedesek száma.

3. Tételes kiiratás vezérlés "T" kulcsjelü paraméterrel.

kulcsjel,sorszám,elhelyezkedés a rekordban,elhelyezkedés a nyomtatási mezőben,hossz,soremelés mutató.

4. Tételes szerkesztett numerikus kiiratás vezérlés "S" kulcsjelü paraméterrel.

kulcsjel,sorszám,elhelyezkedés a rekordban,elhelyezkedés a nyomtatási mezőben,hossz,tizedesek száma.

5. Felhasználói fejlécvezérlés "F" kulcsjelü paraméterrel.

kulcsjel,sorszám,felhasználói fejlécsor.

6. Gyüjtők közötti müvelet vezérlése "M" kulcsjelü paraméterrel.

kulcsjel,sorszám,eredmény gyüjtő száma,=,első operandus gyüjtő száma,müveleti jel,második operandus gyüjtő száma.

7. Adott konstans és gyüjtő közötti müveletvégzés vezérlése "C" kulcsjelü paraméterrel.

kulcsjel,sorszám,eredménygyüjtő száma,=,első operandus gyüjtő száma,müveleti jel,adott konstans.

A paraméterek közül csak az egyik megadása kötelező a többi szabadon választható. A felsorolt paraméterek segitségével bármely adatfeldolgozás végeredménye táblázható.

A gyakorlati felhasználás bebizonyitotta, hogy a team a fejlesztés első szakaszában képes volt az RPG bonyolult paraméterezését leegyszerüsiteni és a felhasználók igényeit kielégiteni. Az igényelt gyors információszolgáltatás nem képzelhető el az ilyen és

ehhez hasonló egyszerüen vezérelhető program nélkül. Nem szabad elhanyagolni a programozáson nyert időt sem és a könyvtárban a legcélszerübb helykihasználást. A jelenlegi táblázást végző programokkal szembeni nagy előnye, hogy nagyon könnyen kezelhető. Vállalatunknál az előre nyomtatott bizonylatokat kivéve minden táblázat a PTPWL2-vel készül. Mivel napjainkban sok statisztikai kimutatást kell készitení a feladatorientált paraméterezhető program előnye itt mutatkozik meg. Eddig a programozó minden egyes kimutatáshoz uj programot irt, ennek elkészitése és müködésre alkalmassá tétele nagyon időigényes volt. Mivel ezek a kimutatások egyediek ez nagyon leterhelte a programozói gárdát. A PTPWL2 bevezetésével jelentős idő szabadult fel mert a program egyszerü kezelésénél fogva a statisztikák elkészitése nem jelent gondot. A paraméterek megirása nem igényel nagyobb programozói tudast, igy a felhasználó is közelebb kerül a számitógepes adatfeldolgozás megértéséhez.

Most dolgozunk a program egy olyan változatán, amely lehetővé teszi, hogy rendezetlen állomanyokat is képes legyen rendezetten tablázni.

PÁRHUZAMOS MÜKÖDÉSÜ HOMOGÉN SZÁMITÁSI RENDSZEREK TERVEZÉSE, PROGRAMOZÁSA ÉS ALKALMAZÁSA

Legendi Tamás
MTA Matematikai Logikai és Automataelméleti Tanszéki Kutató Csoport

A jelen dolgozatban áttekintést kivánunk adni a kutatócsoportunknál folyó komplex sejtprocesszor kutatásról, amelynek keretén belül matematikai vizsgálatokkal, szimulációs nyelvek fejlesztésével és alkalmazásával, sejtprogramozási és /minimizálást is magában foglaló/ sejt mikroprogramozási technológia létrehozásával, hardware fejlesztéssel és alkalmazási területek vizsgálatával foglalkozunk. Kiemeljük a szimulációs rendszereinkkel /kb. 20.000 soros FORTRAN ill. assembly programokkal/ szerzett kezdeti sejtprogramozási tapasztalatainkat, amelyeket a konkrét alkalmazások irányában szeretnénk kiterjeszteni.

A hagyományos szekvenciális számitógépek igen sokféle alapelemből állnak, sebességük korlátozott.

Gyártástechnológiai szempontból homogén komponensekből álló rendszerek igen előnyösek lennének.
A sebesség növelésére párhuzamos müködésü rendszerek kinálnak lehetőséget.

A gyakorlatban homogén rendszerek egyenlőre nem terjedtek el, különböző párhuzamos müködésü rendszerek /bár sok és nehéz problémával/ azonban már megjelentek.

Eléggé kézenfekvő a párhuzamos müködésü homogén rendszerek gondolata - ennek egyik elméleti modellje a sejtautomata. Az elég sok eredménnyel rendelkező elmélet ellenére sejtprocesszorok eddig nem jelentek meg, aminek fő oka a gyakorlati célokra használható rendszerek óriási /a megvalósithatóságot akadályozó/ mérete volt.

Az LSI technológia megjelenése jelentősen javitotta a helyzetet, egyben fokozódó igényt jelent homogén rendszerekre. A jelenlegi integráltsági szint azonban kb. három nagyságrenddel alatta marad a hagyományos elmélet alapján megvalósitható sejtautomatákhoz szükséges szintnek.

Ezért rendszertechnikai változtatásokra és ujszerü sejttérprogramozási szemléletre van szükség, ha nem akarunk hosszu ideig várni.

Valóban, célkitüzésünk fontos eleme már létező technológiára alapozott sejtprocesszorok létrehozása.

A célkitüzés másik alapvető eleme a gazdaságosság: a hagyományos szekvenciális gépek a funkcionális egységek szintjén ma már igen jól kihasználtak /operációs rendszerek/ a hardware elemi épitőköveinek /kapuk, tárolóelemek/ szintjén azonban rendkivül alacsony a kihasználtság /az egyidejüleg hasznosan müködő elemek aránya igen alacsony/.

A Cellprocessors in computer architecture c. cikk Toffoli alapvető javaslatainak figyelembe vételével és azok messzemenő általánositásával konkrét rendszertechnikai javaslatokat

tartalmaz létező technológiával megvalósitható, /hagyományos sejtautomatákhoz képest/ közepes sebességü /de a hagyományos számitógépeknél gyorsabb/ igen gazdaságos sejtprocesszorok felépitésére.

A javaslat alapvető elemei:

a méretek csökkentésére, a gazdaságosság biztositására és a szükséges alapsoftware mennyiségének jelentős csökkentésére nem önálló, hanem számitási rendszer részeként müködő sejtprocesszort alkalmazzunk;
a homogén alapelemek méretének csökkentésére és a rugalmasság jelentős növelésére mikroprogramozható sejtprocesszorok bevezetése; a homogén alapelemek méretének drasztikus csökkentésére központilag mikroprogramozható kvázisejtek bevezetése /ez a megoldás teszi reálissá a megvalósithatóságot; hátránya, hogy lassul a müködés, azonban a gazdaságosság jelentősen nő/.

ezek az elemek a sejtprogramozási tapasztalatok figyelembevételével /v.ö. a Cellular programming c. cikket/ mostanra bővültek:

a sejtprogramozás /elfogadhatóan gazdaságos/ lehetővé tételéhez a hagyományos elmélettől eltérően térben és időben inhomogén sejtterekre van szükség /a gyártástechnológiai homogenitás megőrzése mellett/ ezt a belső memóriával rendelkező sejtek ill. a bővitett központi mikroprogramtár teszi lehetővé.

Az egész témakör vizsgálatához az egyik első lépés volt szimulátorok kifejlesztése, ezekről több cikkben is beszámoltunk,

pl. az INTERCELLAS - an interactive cellular space simulation language c. cikk világosan kifejti a szimuláció szükségességét, az adott célra történt szimulációs nyelv - fejlesztés és a párbeszédes felhasználás egyes előnyeit.

Altalában is software rendszereink ugyan használhatók általánosabb célokra is /pl. a szimulációs rendszerek biológiai modellezésre, a mikroprogram - átmeneti függvény - minimizációs rendszer Boole függvények minimizálására/ de viszonylag korlátozott módon; alapvetően célorientáltak, a jövendő sejtprocesszorok cross-softwarejét alkotják.

Ehhez a rendszerhez tartozik a TRANSCELL nyelv is /TRANSCELL - a cellular automata transition function definition and minimization language for cellular microprogramming/ amely tulajdonképpen egy speciális felépitésü sejt közepes szintü mikroprogramozási nyelve. A sejt müködése egy billenési lépés során két fázisu - szomszédai állapotait /kisebb bitszámu/ tulajdonság szóra redukálja /tulajdonság-kivonás/ majd ennek alapján határozza meg uj állapotát. Ennek megfelelően a TRANSCELL nyelvben a mikroprogram /átmeneti függvény/ definiálására tulajdonságleiró utasitáscsoport és tulajdonság kombinációk alapján uj állapotot előiró utasitáscsoport szolgál. A minimizálás félautomatikusan történik - a programozó direktivákkal aktivizálhat minimizáló müveleteket. Az elérhető minimizálás jelentős.

A fent emlitett sejten kivül kidolgozásra kerültek egy kétállapotu sejt tervei, SSI/MSI modellek készültek /1-16 sejt/, sejt központi egység és R-10-hez illesztésének a tervei készü-

tek, megindult egy /a Braunschweigi Müegyetemmel együttmüködésben kivitelezésre kerülő/ sejtprocesszor emulátor tervezése, amelynek tapasztalatait a már folyó LSI tervezéshez kívánjuk felhasználni /az emulátor egyben gyakorlati méretü éles feladatok - viszonylag lassu - futtatását is lehetővé tehetné/.

Az egész összetett témán belül sulyponti szerepe van a sejtprogramozásnak, sejtalgoritmusok kidolgozásának, hiszen ezen mulik a tervezett rendszer alkalmazhatósága.

A homogén terek programozására ismeretes hagyományos eredmények /Turing-gép beültetése, hardware szimuláció, stb./ elméleti értéküek ill. rendkivül gazdaságtalanok, alkalmazási célra nem megfelelőek.
Általában kisért a szekvenciális gépek beültetésének a gondolata - ami kényelmes lehet ugyan, de egyáltalán nem hatékony, nem használja ki a sejttér, mint párhuzamos müködésü rendszer lehetőségeit. Igen természetes a javasolt futószalag rendszerü programozási technika: átmeneti időre stabil, futószalaggá összekapcsolt müveletvégző elemeken folyamatos adatáram kerüljön feldolgozásra. Természetesen tömbmüveletek, futószalagon belüli tömbmüveletek ill. több futószalag egyidejü üzemeltetése lehetséges. /az elterjedt terminológiával fogalmazva: data stream/ek/ pipe-line és/vagy array processzálása/.
A javasolt technika mellett is nagyméretü szerkezeteket vagy nagy állapotszámot igényel a homogén sejtekkel való megvalósitás.

A Cellular programming cimü cikk bevezeti és indokolja a térben és időben inhomogén /de gyártástechnológiailag homogén/ tereket; programozási módszert ad kezelésükre és konkrét pél-

dákon /rendezés, összeadás/ mutatja be a futószalag feldolgozás előnyeit.

A javasolt kétállapotu /inhomogén/ sejt univerzalitását Codd ill. Banks eredményére történő visszavezetéssel bizonyitja a cikk.

Az alkalmazhatóság szempontjából igen fontos kérdés, hogy a /különböző szintü/ felhasználók hogyan férnek hozzá a sejtprocesszor által nyujtott szolgáltatásokhoz.

A Cellular programming c. cikk főleg software oldalról tárgyalja a kérdést:
az egyik járható ut assembly majd magasabb szintü sejtnyelvek kialakitása, és ez a megoldás hosszu távon igen fontos és előnyös;
a másik járható ut a sejtprocesszornak az end-user elől történo /részbeni vagy teljes/ takarása, azaz a közvetett /implicit/ használat biztositása - ez a megoldás a használatba való bevezetés során rendkivül fontos, hiszen sokkal kevesebbet kiván a felhasználótól, a javasolt konkrét megoldások /sejtmakrokkal bővitett hagyományos makro assembler, magasszintü nyelvek run-time systemeinek sejtprocesszorral segitett futtatása/ egyértelmüen alátámasztják ezt a megközelitési módot.

Nyilvánvalóan ugyanilyen fontos a hardware architekturába való beillesztés kérdése is /a legegyszerübben I/O egységként csatolható sejtprocesszor hagyományos számitógéphez; alkothatja a cimezhető memória részét is, ez elég drága és jelenleg még

nem reális, de igen rugalmas megoldás; szóbajöhet az architektura egyes részeinek kiváltása sejtprocesszorral, stb./

Feltétlenül egységes módon kell vizsgálni a fenti illesztési kérdéseket, talán a sejtprocesszorok digitális architekturába illesztése elnevezés /amelybe a hardware és a software architekturát is beleértjük/ fedi a legjobban a problémát. Tipikusan felvetődő következménye ennek a szemléletnek pl. az egyes operációs rendszer moduloknak sejtes firmware-rel való kiváltására irányuló javaslat.

Az alapvető csatolási és általános használati lehetőségek vizsgálata mellett az eddigi sejtprogramozási eredményekre alapozva törekszünk általánosan használható sejtalgoritmus könyvtár /vektor és mátrix müveletek, szimbólumtáblából keresés, lexikografikus rendezés, konvertálások, sinus-cosinus számitás, stb./ kialakitására és konkrét alkalmazási területek /adatfeldolgozás, adatbázisból visszakeresés, képfeldolgozás - alakfelismerés-, forditás, mérési adat gyüjtés-feldolgozás, adattömörités stb./ felkutatására és az alkalmazások részletesebb kidolgozására.

A szerzőnek a dolgozatban idézett cikkei:

- Cellular processors in computer architecture.

  Computational Linguistics and Computer Languages, XI. /1977/ 147-167.

- INTERCELLAS - an interactive cellular space simulation language.

  Acta Cybernetica /1977/ Tom 3, Fasc. 3., pp. 261-267.

- Programming of cellular processors.

  "Cellular meeting" Braunschweig, 1977, junius
  Informatik-Bereichte Nr. 7703
  Technische Universität Braunschweig pp. 53-66

- TRANSCELL - a cellular automata transition function and minimization language for cellular microprogramming.

  To appear in: Computational Linguistics and Computer Languages XII.

A fenti cikkek irodalomjegyzéke tartalmazza a terület legfontosabbnak ítélt anyagait; ezen kivül a szerző készségesen rendelkezésre bocsátja az érdeklődőknek a nagyszámu cikket /valamint bibliográfiát és könyvet/ tartalmazó "házikönyvtárának" számitógéppel készülő aktuális katalógusát.

EGY FORMULA MANIPULÁCIÓS RENDSZERRŐL

Lovas Istvánné - Zimányi Magdolna
MTA Központi Fizikai Kutató Intézet
Számitástechnikai Főosztály

A numerikus számitások céljára szolgáló programozási nyelvek megjelenését eléggé számottevő késéssel követte az algebrai formulák szimbolikus kezelésére szolgáló programrendszerek kidolgozása. Az első programok ezen a területen a 60-as évek elején készültek, a 60/70-es évek fordulójára már több nagyobb rendszert fejlesztettek ki /FORMAC, SCHOONSHIP, REDUCE, REDUCE2, SAC-1, ALTRAN/.

A késés oka részint az volt, hogy eleinte az érdeklődés is sokkal inkább a numerikus problémák felé fordult, másrészt viszont hiányoztak a megfelelő eszközök és fontos elvi és módszertani kérdések megoldatlanok voltak.

Az algebrai rendszerek későbbi megjelenését és nem széleskörü elterjedését magyarázza a numerikus és algebrai számitások jellegének különbözősége. Talán a legfontosabb különbségek a következők: amig a numerikus rendszerek jól dokumentáltak és portabilisak, az algebrai rendszerek kevéssé dokumentáltak és csak kevés, speciális gépen alkalmazhatók. A numerikus rendszerek viszonylag kis memória igényüek, a memória és időigényük előre megmondható, meghatározott mennyiségü outputot szolgáltatnak, az algebrai rendszerek nagy memóriát igényelnek, sem a memória, sem az időigényük nem becsülhető, sőt az output mérete sem. A memóriaigény a feladat méretével exponenciálisan vagy exponenciálisnál erősebben növekedhet.

Az elvi és módszertani problémák közül csak a legfontosabbakat emlitjük, hogy érzékeltessük a numerikus és szimbolikus számitások közötti eltéréseket. Egyik ilyen probléma a belső ábrázolás. Általában kanonikus alakban ábrázolják a polinomokat, de a kanonikus alak implementálása is sokféleképpen történhet /pl. a változót lehet explicit vagy implicit ábrázolni/.

További lényeges kérdések: algoritmust találni tetszőleges algebrai kifejezés egyszerüsitésére /általában ad hoc technikával történik az egyszerüsités/, vagy annak a felismerése, hogy egy kifejezés értéke 0-e? A számok ábrázolását célszerü egésznek vagy egész számok hányadosának választani /ami maga után vonja a tetszőleges pontosságu aritmetikát/, hiszen csak igy lehet egzakt módon eldönteni, hogy a kérdéses kifejezés 0-e. Ide tartozik még például a legnagyobb közös osztó számitás, a faktorizáció és az integrálás problémája is.

Egy algebrai manipulációs rendszernél alapvető fontosságu az adatstrukturának a megválasztása. A problémát az jelenti, hogy mig a numerikus számitásoknál pl. FORTRAN nyelvben két szám összeszorzásának eredménye ismét egyetlen szám, addig a szimbolikus számitásoknál két polinom szorzásának eredménye egy polinom, amelyről nem tudjuk előre, hogy hány tagból fog állni. Ebből következik, hogy az algebrai manipulációs programok dinamikus adatbázist igényelnek, amelyben az adatok elérése a fizikai tárolás sorrendjétől függetlenül, az adatok belső összefüggése alapján történik. Ilyen tárológazdálkodást biztositanak a különböző listakezelő nyelvek. A legtöbb rendszer vagy valamilyen listakezelő nyelvre é-

pül /LISP, SLIP stb/ vagy belső, saját listakezelést alkalmaz.

Az algebrai rendszerek megvalósitása különböző módokon történt:

a/ egy numerikus számitásra alkalmas nyelv kibővitése, pl. FORMAC: FORTRAN, illetve PL/I kibővitése makrokkal /preprocesszor dolgozza fel a makrokat a FORTRAN/PL/I forditás előtt/.

b/ assembly nyelven kódolt rendszer, pl. SCHOONSHIP /előnye: kisebb memória igény, nagyobb sebesség, hátránya: nem portabilis.

c/ önálló nyelv forditóprogrammal, illetve interpreterrel /REDUCE2/.

A KFKI R40 számitógépén a Hearn által kidolgozott REDUCE2 nyelvet implementáltuk. Munkánk célja egy, az algebrai kifejezések szimbolikus kezelésére jól használható rendszer megvalósitása volt, elsősorban elméleti fizikai /nagyenergiáju fizika, relativisztikus kvantummechanika, relativitás elmélet stb/, csillagászati, mérnöki /parciális differenciál egyenletek, végtelen sorok, általánositott sajátérték egyenletek/ számitások elvégzésére.

A rendszer viszonylag nagy konfigurációt igényel. A minimális tárigény 300K, de ez nagyon könnyen felmehet 500K-ra, sőt még többre is.

A REDUCE2 a LISP nyelven alapul, a felhasználó felé azonban önálló, az ALGOL-hoz hasonló nyelvként jelenik meg. Lehetőség van LISP nyelven irott programrészek beillesztésére is. A REDUCE2-ben szimbolikusan kiszámolt formu-

lák a FORTRAN fordítóprogram számára átadhatók, azaz REDUCE2 nyelven irt program eredményét felhasználva, egy teljes FORTRAN programot is /szubrutint vagy függvényt/ készithetünk.

A nyelv szerkezete lehetőséget nyujt ciklusok szervezésére, feltételes és összetett utasitások, procedurák irására, bizonyos file kezelő utasitásokat is tartalmaz.

A következőkben röviden összefoglaljuk a rendszer által nyujtott főbb szolgáltatásokat.

- Polinomok és racionális függvények kifejtése és rendezése /inputként sokszorosan zárójelezett kifejezéseket adhatunk meg/.

- Szimbolikus differenciálás. A differenciálás formális szabályait a SIN, COS, LOG elemi függvényeket és azok deriváltjait is ismeri. Szimbolikus exponens is megengedett.

- A felhasználó által definiált függvények differenciálási szabályai is megadhatók.

- Behelyettesités és "pattern matching". Kétféle helyettesitést végezhetünk. A SUB operációval a megadott kifejezésben az adott változó helyébe behelyettesiti a kivánt kifejezést, de nem vizsgálja, hogy a helyettesités után előfordul-e még a helyettesitendő változó. A másik helyettesitési mód: mindaddig elvégzi a helyettesitést minden kifejezésben, amig a helyettesitendő változó előfordul.

- Két polinom legnagyobb közös osztojának kiszámitása.

- Kifejezések automatikus és a felhasználó által irányitott egyszerüsitése.

- Szimbolikus mátrixokkal végzett müveletek /determináns, transzponens, trace számitás/.

- Gamma mátrix algebra, tenzor manipuláció /Dirac mátrixokkal végzett müveletek/.

- Interaktiv lehetőségekkel is rendelkezik /time sharing üzemmód esetén/ ami rendkivül előnyös, hiszen a számitások jellege sokszor megkivánja a heurisztikus eljárásokat.

A fenti felsorolás természetesen nem teljes. Ebből is látható azonban, hogy különös tekintettel a nagyenergiáju fizikai számitásokra, a rendszer nagyon sok lehetőséget nyujt a felhasználó számára.

A REDUCE2 könnyen áthelyezhető rendszer, az IBM 360 és a PDP10-es számitógépeken több verziója is müködik, attól függően, hogy milyen az illető gépen implementált LISP.

A REDUCE2 implementálásával elsősorban a számitógép fontos uj alkalmazási területeken való felhasználását próbáltuk elősegiteni. A rendszer a nem numerikus alkalmazással kapcsolatos problémák tanulmányozására is lehetőséget nyujt.

Irodalom

1 A.C.Hearn: REDUCE2 User's Manual.
Univ. of Utah, 1973.

2 A.C.Hearn: Applications of Symbol Manipulation in Theoretical Physics.
CACM 14. 511 /1971/

3 A.C.Hearn: Computer Solution of Symbolic Problems in Theoretical Physics. Computing as a Language of Physics.
International Atomic Energy Agency.
Vienna 1972.

4 R.Roskies: Algebraic Calculation by computer.
APS Meeting. Salt Lake City, Utah 1974.

5 D.Barton-J.P.Fitch: Review of Algebraic Manipulative Programs.
The Computer Journal 15. 1972.

UJ JOB-ÜTEMEZÉSI RENDSZER KIALAKITÁSA AZ ÁSZSZ HwB 66/60-AS SZÁMITÓGÉPEN

Magyar László - Sztrókay Kálmán
Államigazgatási Számitógépes Szolgálat

A cimben emlitett rendszer kialakitásának igénye az összegyült üzemeltetési tapasztalatok és a várható jövőbeni géphasználat jellemzőinek értékelése alapján került kialakitásra.

Célja kettős: egyrész feltárva a számitógép konfigurációjából adódó szük keresztmetszeteket, azok feloldását a joboknak erőforrás-igényükön alapuló szétválogatásával, továbbá az ütemezési paraméterek változtatásával lehetővé teszi, másrészt a kis erőforrás-igényü jobokat előnyben részesitve gyors átfutásukat biztositja, és ezzel a felhasználókat helyes erőforrás-gazdálkodásra ösztönzi.

Az uj job-ütemezési rendszer megvalósitásának módja a HwB számitógépen GCOS operációs rendszerében ennek tervezői által nyitvahagyott standard interface-en /u.n. exit rutin/ alapuló, az eredeti ütemező müködését felüliró program beépitése volt. E program a jobok beolvasása során - a job-vezérlő kártyákat /periféria-igények, processzor idő, memória, nyomtatási limitek/ elemezve - azokat a rendszer-ütemező különböző /általunk létesitett/ queue-jában helyezi el, egyben meghatározva sürgősségüket /priority/ is. Ez a módszer a joboknak a GCOS eredeti csztályozásánál

differenciáltabb szétválogatását teszi lehetővé, biztositja a feldolgozás jellegétől függő ütemezési statégia kiválasztásának lehetőségét, a hatékony operátori beavatkozást.

A jobok besorolását meghatározó kritériumok kiválasztását, a paraméterek értékeinek beállitását átfutási statisztikák készitése és elemzése segitette a hatékonyság-növekedés és a gépkihasználás fokozása érdekében.

A SZÁMOK RTE RENDSZERE

Majorosné Koós-Hutás Mária, Székely Zoltán
SZÁMOK

1. Általános bevezetés

A program-fejlesztés és oktatás céljait szolgáló RTE /Remote Text Editor/ rendszert a SZÁMOK Software Fejlesztési osztálya hozta létre a hatékonyabb számitógép felhasználás elősegitésére.

1.1 A kifejlesztés alapját képező tényezők

A rendszer kifejlesztése idején a SZÁMOK még nem rendelkezett interaktiv géphasználati lehetőséggel. A hallgatók gyakorlati feladataikat az IBM 37o/145-ös gépen, "batch closed shop" üzemeltetési rendszerben oldották meg. A fordulási idő akkor még több napos is lehetett, s ma sem kisebb általában egy napnál. Igy a hibátlan, s müködő programok elkészitése meglehetősen hosszu ideig tartott, s a hallgatók munkakedvüket is elvesztették a várakozások során. Végül értékes gépidő ment veszendőbe azáltal, hogy a job-ok javitásához a nagy-számitógépet kellett igénybe venni.

Az RTE rendszer, amely a SZÁMOK R-10 gépére készült, a következő problémákat szándékszik tehát megoldani:

- több felhasználó egyidejü számitógép használatát,
- közvetlen számitógép hozzáférést,
- a nagy gép jobb kihasználását,
- a fordulási idő lerövidités ét.

Az RTE létrehozásakor ugyanakkor arra törekedtünk, hogy egy általánosan hasznositható software-rendszert készitsünk, amely az R-10 és egy nagyobb ESZR modell /vagy IBM 36o-as, 37o-es/ közötti kommunikációt, Remote Job feldolgozást tesz lehetővé.
Esetünkben a kapcsolat az R-10 és az IBM 37o/145-ös számitógép között áll fenn.

1.2 Az RTE koncepció lényege

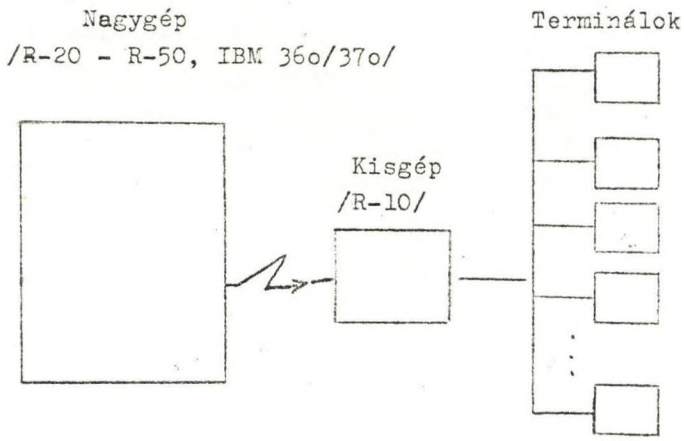
Az ismertetett job feldolgozás hátrányainak kiküszöbölésére az RTE rendszer az alábbi megoldást alkalmazza:

A job-ok feldolgozása továbbra is a nagy-számitógépen történik, de a job input/output nem a nagy-gép amugyis szük keresztmetszetét képező I/O perifériákon valósul meg, hanem adatátviteli vonalakon.
A job-ok előkészitése, összeállitása, javitása az adatátviteli vonal másik végén lévő R-10-es számitógépen történik.

Az R-10 termináljainál /8 db/ ülő programozók közvetlenül a klaviaturáról szerkesztik a job-file-okat, amelyek az adatátviteli rendszer segitségével jutnak el a nagy-számitógépbe.

A nagygép a job-ok feldolgozása után a forditás vagy futás eredményét visszaküldi az R-10 terminálnál ülő felhasználónak. Az eredményt a felhasználó - kivánság szerint - megjelenitheti a képernyőn, kinyomtathatja az R-10 sornyomtatóján vagy más adathordozón rögzitheti.

A terminálnál ülő programozók az eredmény birtokában máris megkezdhetik a következő "fordulót". Az eredeti file-ból a szerkesztő program /Text Editor/ segitségével egy ujabb job-file készül, s ez ismét átkerül a nagyszámitógépbe.



A file-okat az R-10 oldalon mágneslemezen tároljuk; a szerkesztés alatt álló szövegrész a tárban van. /szerkesztő puffer/.

A job-ok feldolgozása tehát két fázisból áll:

a/ a job előkészitéséből /R-10/
b/ a job feldolgozásából /nagy-számitógép/

Mivel egy adott időn belül a felhasználó több géphez-fordulást is realizálhat, a program készités fázisai lényegesen lerövidülnek.

Az RTE rendszer fenti alapfelhasználásán kivül az R-10 oldalon számos más szolgáltatást is nyujt.

2. Az RTE rendszer felépitése

A rendszer tervezésekor bizonyos hardware és software adottságokat, feltételeket figyelembe kellett vennünk, melyek közül a legfontosabb a SZÁMOK-ban rendelkezésre álló konfiguráció volt.

R-10 hardware: CPU: 32 Kszó
Foreground kezelő kártya /4 foreground szint/
DRI-lemez /max. 2 egység/
Sornyomtató, mágnesszalag-egység, kártyaolvasó, lyukszalagolvasó és lyukasztó
Konzolirógép
Display-terminálok /8/
Szinkron-vonal kezelő

software: RTDM operációs rendszer 5. verzió /file-rendszer nélkül elegendő/
DTM adatátviteli monitor

Nagy-számitógép hardware:

Az IBM 37o/145 alapkonfiguráción kivül egy 37o4-es Átvitelkezelő egység /on-line kapcsolat az R-10-zel/, ami a 27o3 vezérlőt emulálja.

software:

DOS/VS POWER/VS operációs rendszer /32-es release/

A rendszer szerkezete

A feladat célkitüzéseiből és a feltételekből következően az alábbiak szerint alakitottuk ki a rendszer szerkezetét:

a/ A rendszer külső és belső funkciói csoportokba gyüjthetők, melyeket "szolgáltatás"-nak nevezünk. Egy szolgáltatás hasonló tevékenységeket végez és/vagy közös adatstrukturái és müveletei vannak. Egy szolgáltatás lehet például a File Rendszer vagy a Szövegszekesztő.

b/ Az egyes szolgáltatásoknak egymástól a lehető legfüggetlenebbeknek kell lenniök, s nem szabad tudniok arról, hogy milyen célra veszik igénybe. Igy bizonyos mértékü általánosság érhető el.

c/ Az egész rendszer az RTDM felügyelete alatt dolgozik, s ahol lehet, az RTDM szolgáltatásait kell közvetlenül használni.

d/ A több-felhasználós rendszer megvalósitásához a "folyamat" /process/ elvét alkalmaztuk. Ez azt jelenti, hogy az egyes felhasználók számára folyamatok müködnek, esetleg ugyanabból a szolgáltatásból egyidejüleg több is /re-entrant kód/. A belső /rendszer/ folyamatokat megkülönböztethetjük a felhasználókat kiszolgáló folyamatoktól.

e/ Az RJE-t egy külső alrendszerként kell létrehozni, amely önmagát vezérli. Az RJE müködés során a job-müködés lemez-file-ból történik, és az eredmény is lemez-file-ba jut vissza az R-10 oldalra.

A nagy-számitógéppel való kommunikáció lebonyolitásához a már meglévő R-10 software-re épitettünk /RTDM 5. verzió, DTM/.

f/ A szolgáltatások funkciói között nincs CPU igényes folyamat, ezért mesterséges időosztásra nincs szükség.

A fentiekben vázolt elvek alapján a rendszer fő részei a következők:

RTEX	adatátviteli alrendszer
RTEC	vezérlő alrendszer
RTEU	felhasználói alrendszer

Az egyes részek részletesebben:

1. A felhasználói alrendszer /RTEU/ szolgáltatásai

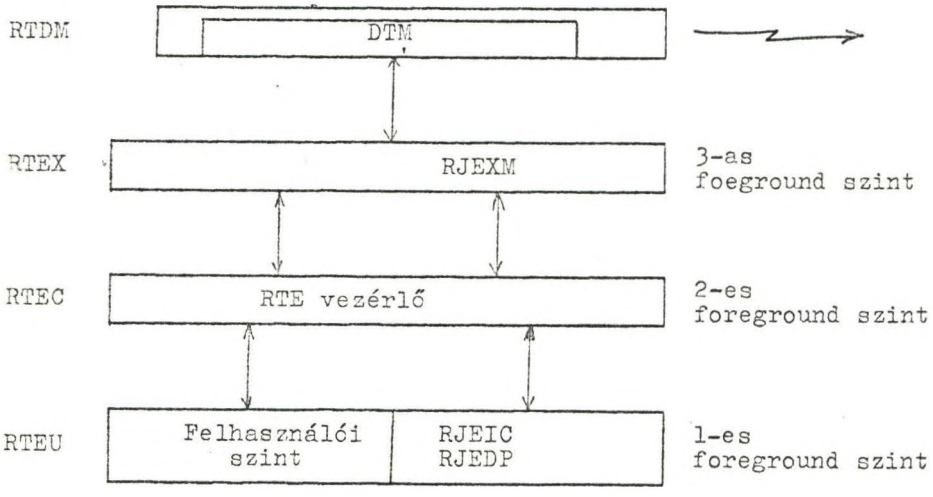
Bejelentkezés	/LOG/
Parancsértelmező	/CLI/
File Rendszer	/FS/
Szövegszerkesztő /text editor/	/QED/
Másoló Rendszer	/COPY/

2. A vezérlő alrendszer /RTEC/ szolgáltatásai

Processor /folyamat/ kezelő	/PS/
Tár kezelő	/MM/
I/O kezelő	/IOM/
Folyamatirányitó	/TC/

3. Az adatátviteli alrendszer /RJE/ szolgáltatásai

RJE felhasználói kommunikáció /RJEIC/
RJE file-rendszer kommunikáció /RJEDP/
RJE átvitelkezelő /RJEXM/



Az RTE rendszer helye

Összegzés

A rendszer 7 emberévnyi munkát igényelt, 1976 őszén indult meg a próbaüzemeltetés. Tapasztalataink kedvezőek. A hallgatók a géppel való közvetlen kapcsolat következtében viszonylag gyorsan képesek elkészíteni egy-egy programot, munkakedvük, lelkesedésük jelentősen megnőtt. Az időközben installált PDP11/70-es számítógép és az erre készült IPR interaktiv programnyelv mellett sem veszitette el jelentőségét, hisz a kereskedelmi fordítóprogramokat továbbra is az RTE-n keresztül lehet kényelmesen elérni.

A DOS-POWER ÜZEMELTETŐ RENDSZER TOVÁBBFEJLESZTÉSE TERÉN ELÉRT EREDMÉNYEK

Merkel Géza
Kohó és Gépipari Szervezési és Számitástechnikai Intézet

Hazánk jelenlegi számitógép-állományának legnagyobb részét az 1973-tól telepitésre került ESZR RJAD (R20,R22,R30,R32, és R 40) tipusu számitógépek alkotják. A számitógéprendszerek a közismert hardware és software adottságok miatt többségben a DOS operációs rendszer felügyelete alatt üzemelnek. A KG ISZSZI 1974. végén installált R 30-as tipusu számitógépére az Intézet Software Fejlesztési Osztályának munkatársai különféle operációs rendszer alternativákat vizsgáltak meg. A vizsgálatok eredményeképpen kiderült, hogy az Intézet adottságainak a POWER II-vel támogatott IBM DOS 26.2 operációs rendszer felel meg mert:

- A különféle ESZR DOS operációs rendszerekből hiányzik több olyan fontos komponens, amely az IBM DOS 26.2-nél rendelkezésre áll.

- A POWER II programcsomag segitségével a DOS alatti multiprogramozás igen hatékonyá tehető (igaz csak két felhasználói programterületen).

- Egy OS (MFT vagy MVT) operációs rendszer tekintettel a 256 Kbyte-os operativ törzskapacitásra és a 6 db ESZ 5052-es tipusu mágneslemezegységre az alkalmazhatóság alsó határán mozog.

- Az intézeti munkák 90 %-a perifériaigényes adatfeldolgozás jellegü, ezért 3-nál több program egyidejü futtatása (multizása) OS operációs rendszer alatt csak igen gazdaságtalanul valósitható meg. 2-3 szoros multiprogramozás esetén pedig a DOS operációs rendszer hatékonyabb az OS-nél.

A POWER II rendszer alkalmazásából a következő előnyök származtak:

- a lassu perifériák (kártyaolvasó, lyukkártya lyukasztó és sornyomtató) kezelését a POWER vette át biztositva a berendezések folyamatos üzemét az eddigi start-stop üzemmóddal ellentétben.

- A rendszer futtatási particióiban (BG és F2) a job-ok átfutási ideje megrövidült az off-line kártyaolvasás és lyukasztás, valamint nyomtatás következtében.

- Lehetővé vált a job-ok un. külső prioritás szerinti ütemezése.

- A lyukasztott és nyomtatott output-ok osztályokba sorolásával könnyebbé vált azok kezelése.

- A multiprogramozás a futtatási particiókban viszonylag kevés lassu perifériával is lehetővé vált. Ezenkivül a POWER partició (F1) lényegében multitask üzemmódot valósitott meg a különféle olvasó és irófeladatok (taskok) szimultán aktivizálásával.

A DOS 26,2 operációs rendszer alkalmazásánál azonban problémát okozott a Job Controll program merevsége, amely elsősorban a fizikai szintü periféria hozzárendelésben és a nem elégséges munkaleirás ellenőrzésben nyilvánult meg. A KG ISZSZI-ben folytatott fejlesztési munkák éppen a hiányosságok kiküszöbölését célozták meg. A fejlettebb DOS változatoknál (pl. DOS VS RELEASE 34) és az OS operációs rendszereknél a problémák szintén megoldottak ezért a fejlesztésnél bizonyos kompatibilitási szempontok is figyelembevételre kerültek. A fejlesztés leglényegesebb alapelve az volt, hogy a módositásokat nem az alapsoftware-nak számitó DOS Job Controll programon, hanem a POWER II rendszeren kell elvégezni, mégpedig ugy, hogy a POWER programhoz saját rutinokat kell készitení. A KG ISZSZI által elvégzett POWER II módositások ilymódon növelik a DOS-POWER II üzemeltető rendszer hatékonyságát azáltal, hogy kapcsolatot létesitenek a POWER II program bizonyos helyein a különféle funkciókat ellátó saját rutinokkal.

E rutinok a POWER II programba két pontban (az 1 sz ábrán A és B-vel jelölt pontok) kerültek beillesztésre.
Az első belépési pont az a hely ahol a POWER a job kártyák információit küldi az operációs rendszer (job controll program) felé a másik pont pedig az ahol az output listázási müveletek inditása történik meg. E két ponton keresztül ilymódon lehetőség van a POWER által kezelt a lassu perifériák felé irányuló teljes adatforgalom felülvizsgálatára és esetleges módositására.

Az 1. sz. ábra a POWER II program fő elemei közötti kapcsolatot szemlélteti. A POWER fő elemei a következők:

- Olvasó feladat-elvégzi az input adatok beolvasását a kártyaolvasóról és az input munkasorban történő letárolását.

- Végrehajtó feladat elvégzi az input munkasorból történő adatkiolvasásokat és a batch particiók aktivizálását,

- Output kiirófeladat - elvégzi az output munkasorokból az adatok lyukasztását és/vagy nyomtatását.

- Kommunikációs rutin-elvégzi az operátorral a kapcsolattartást,

- Feladatütemező - rutin - elvégzi az egyes POWER feladatok aktivizálását.

- SVC kezelő rutin - elvégzi a DOS supervisorral történő kapcsolattartást.

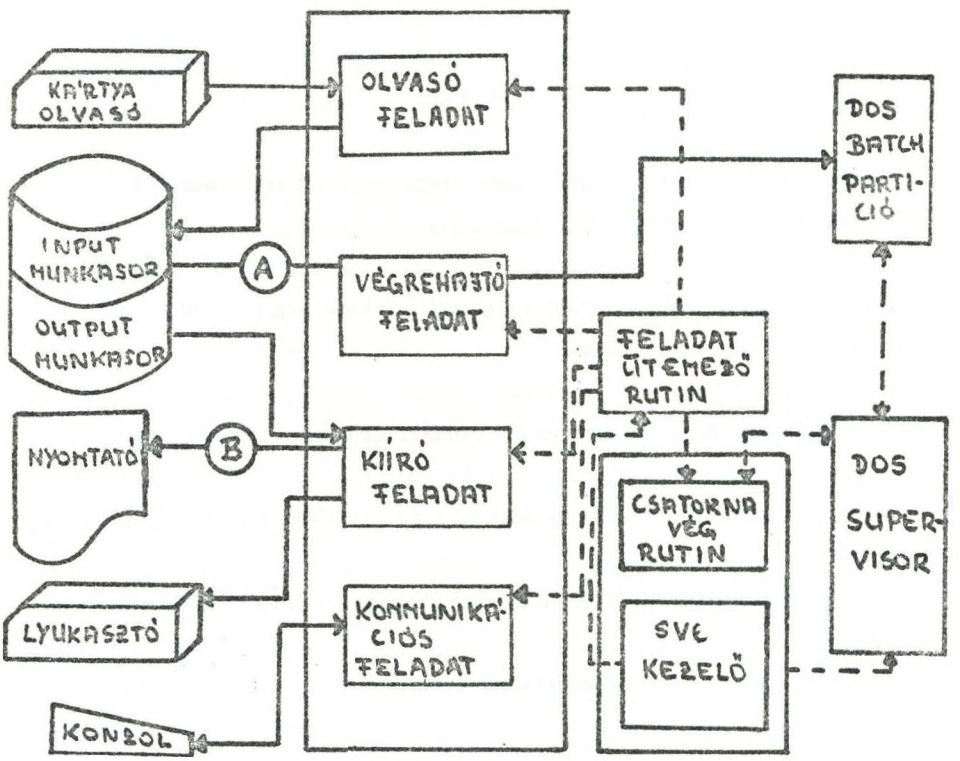
- Csatornavég rutin a Supervisorral együtt lekezeli az I-O megszakitásokat.

Az ábrán az "A" belépési pont (POWEROWN rutin) akkor kerül végrehajtásra mielőtt a végrehajtó feladat az input munkasorból a kártyaformátumokat a DOS futtatási particiók felé továbbitása.

Ebben a felépitési pontban elvileg elvégezhető minden felhasználói módositás a paraméterként átadott 80 byte-os kártyaformátumokon.

Ezek a módositások a következők lehetnek:

- Adatkorrekció (pl. lazább szintaktikai megkötések a Job Controll paramétereinél)
- Job Controll utasitáskészlet' bővités (pl. automatikus periféria hozzárendelések megvalósitása)
- Komplett munkaleirás ellenőrzés (Pl. munkaszám, futásidő-, használható periféria szám ellenőrzés).



1. ábra A POWER II elemek és a felhasználói belépési pontok kapcsolata.

A "B" belépési pontokon a felhasználó akkor kap vezérlést, amikor a kiíró-feladat hozzákerül egy job outputjának kinyomtatásához. Ebből a belépési pontból egy módositott job elválasztólapot készitő rutin (POWERXSP) aktivizálását lehet kérni a rendszertől.
A POWER II rendszeren végrehajtott módositások 4 db POWER makrót (POWER, POWERRES, POWERSUP és POWERDS), valamint 1db POWER tranziens rutint (FGPSPOLH) érintett. Az "A" és "B" belépési pontokból hivott felhasználói rutinok paraméterezhető makrók formájában készültek el ezáltal azok a konkrét számitógép konfigurációra és a különböző felhasználói igényekre adaptálhatók.

A POWEROWN rutin lehetséges szolgáltatásai a következők:

1./ Munkaszám ellenőrzés - A rendszer által elfogadható 5 karakter hosszuságu alfanumerikus munkaszámokat egy szekvenciális szervezésü mágneslemezes file (JOBNFIL) tartalmazza. A munkaszámot a POWER JECL * $$ JOB kártyán a job név mező első 5 karakterében kell megoldani. Illegális munkaszám esetén a teljes POWER job kihagyásra kerül és a conzolon hibaüzenet jelenik meg.

2./ Job osztály ellenőrzés A futtatási particiókban (BG, F2) végrehajtandó munkák un. job osztályokba sorolhatók.

Minden egyes job osztályt valamely futtatási partició és memóriaméret ezenkivül kötött számu mágneses adathordozó (mágnesszalag vagy mágneslemez) jellemzi. A job osztályok partició és perifériaigénye paraméterek formájában közölhető a POWEROWN rutinnal, annak forditása során. A job osztályok A-Z karakterrel azonosithatók. Egy POWER job job osztályát a * $$ JOB kártya job név mezejének 6. karakterében kell specifikálni. Illegális job osztály esetén a teljes POWER job kihagyásra kerül és a consolon hibaüzenet jelenik meg.

Automatikus periféria hozzárendelés. A POWEROWN rutin a mágneses perifériák dinamikus (futásközbeni) hozzárendelését a bővitett ASSGN Job Controll utasitás segitségével hajtja végre. A bővitett ASSGN utasitások a következők lehetnek:

a./ // ASSGN SYSnnn, SYSmmm

azaz, a SYSnnn-logikai cim hozzárendelése a SYSmmm logikai cimmel rendelkező fizikai cimhez. Például ha a SYSmmm hozzárendelés az X'28Ø'fizikai cimhez történt előzőleg, akkor a POWEROWN rutin által továbbitott job kártya a következő:

// ASSGN SYSnnn, X'28Ø' SYSnnn, SYSmmm

b./ // ASSGN SYSnnn, VOL1=TAPE

azaz, a SYSnnn logikai cim hozzárendelése egy olyan mágnesszalagegységhezmelyen egy munkaszalag (TM-al kezdődik) helyezkedik el. Például ha az X'28Ø' cimen egy mágnesszalag helyezkedik el, akkor a POWEROWN által továbbitott kártya a következő:

// ASSGN SYSnnn, X'28Ø' VOL1 = TAPE

c./ // ASSGN SYSnnn, VOL1=DISK

azaz, hozzárendelés egy munkalemezhez. Munkalemeznek tekintendő minden 111111, 222222, 333333 stb. VOL1 cimkével rendelkező lemez.

d./ // ASSGN SYSnnn, VOL1=XXXXXX

azaz, hozzárendelés egy olyan mágnesszalagos vagy mágneslemezes perifériához, melynek a VOL1 cimkéje megegyezik a kártyán az x-ek helyén megadottal. Az azonositás meggyorsitása céljából a mágnesszalag egységek VOL1 cimkéiben speciális karaktert/eket lehet elhelyezni (Pl: TPØØ28, SOFTØ5 stb) A POWEROWN rutin ellenőrzi a szabad egységeken elhelyezkedő mágneses perifériák VOL1 azonositóit. Abban az esetben, ha a kártyán megadott VOL1

azonositó és valamely adathordozó VOL1 cimkéje megegyezik, ugy a hozzárendelés megtörténik. Például ha a VOL1= DISKØ5 cimkéjü lemez a X'193'-as fizikai cimen helyezkedik el akkor a POWEROWN rutin által továbbitott ASSGN kártya a következő lesz:

// ASSGN SYSnnn, X'193' VOL1=DISKØ5

e./ // ASSGN SYSnnn, FILEID =44 karakteres azonositó, azaz, hozzárendelés egy olyan perifériához, melyen egy olyan file helyezkedik el, amelynek a 44 karakteres file azonositója megegyezik a kártyán megadottal. A rutin keresést végez az egyes mágneslemezek cimketáblázataiban (VTOC) a megadott file azonositó szerint. Egyezés esetén a hozzárendelés megtörténik, mig ellenkező esetben a rutin üzenetet küld az operátornak, melyben a kivánt lemezcsomag felhelyezését kéri.

- Adathordozó használat ellenőrzés. A POWEROWN rutin minden egyes DOS job step futtatása előtt megszámolja a lekötött (hozzárendelt) mágnesszalag és mágneslemez egységeket és ellenőrzi, hogy a POWER job job osztályának megfelelnek-e. Ha több mágneses periféria lekötése történt meg mint ami a job osztály részére meg volt határozva a job step futtatása meghiusul.

- Automatikus lemezterület kijelölés A POWEROWN rutinnak ez a funkciója a DOS Job Controll program egyik legnagyobb hiányosságát kompenzálja azáltal, hogy automatikus lemezterület (sáv vagy cilinder) kijelölést valósit meg. Például a :

```
// EXTENT SYSØØ5,,,,TRK=5Ø
// EXTENT SYSØØ6,,,,CYL=15
```

"hamis" Job Controll utasitásokat a rutin a:

```
// EXTENT SYSØØ5,,,,1211,5Ø  TRK=5Ø
// EXTENT SYSØØ6,,,,15ØØ,15Ø  CYL=15
```

Job Controll utasitásokra konvertálja át miután megfelelő számu szabad sávot és cilindert keresett a SYSØØ5, illetve SYSØØ6 logikai cimekkel meghatározott lemezcsomagokon.

A POWEROWN makró lehetővé teszi, hogy azt a legkülönbözöbb felhasználási környezetekre (számitógépkonfigurációra) generálják. A generált rutin maximális mérete 6 Kbyte, amelyet természetesen a POWER partició méretének kijelölésénél számitásba kell venni.:

A POWERXSP rutin lehetséges szolgáltatásai a következők.

- Nagybetüs job szeparátor lap készitése. A job szeparátor lapon 4 db 12x12-es betüméretü információ jelenik meg, amely tartalmazza:
  - A DOS JOB nevét.
  - A futtatási particiót és job osztályt
  - A futás dátumát
  - A futás időpontját.

Ez a funkció a POWER makró JOBSEP=EXT, paraméterével aktivizálható, memóriaigénye 5 Kbyte.

- Felhasználói job szeparátor lap készitése.
  A felhasználónak a saját job szeparátor lapját specifikáló konstans adatokat egy POWNSEP nevü makróban kell megadni kötött szabályok szerint. E makró valamint a POWER makró JOBSEP=USR paraméterének együttes hatására a felhasználó által kivánt job szeparátor lap nyomtatása lehetséges.

Irodalomjegyzék:

1./ IBM S/360 DOS POWER II GH20-0737-3
2./ IBM DOS 26.2 SYSTEM PROGRAMMERS GUIDE
3./ Merkel Géza: POWERXSP, POWEROWN
belépési pontok POWER II rendszerbe.1978. Programfejlesztési dokumentáció
4./ Kovács Zoltán: DOS supervisor és közhasznu rutinok.1978. programfejlesztési dokumentáció

MÁGNESSZALAG NYILVÁNTARTÓ ÉS HIBASTATISZTIKÁT KEZELŐ RENDSZER

Mitterer Walter
FÜTI

Vállalatunknál komoly problémát jelentett a mágnesszalagok nyilvántartása illetve kezelése, a filevédelmek megtartása mellett. Ugyanis egyrészt minden változó adatot rá kellett vezetni az illető szalag raktári kartonjára, ami nagy adminisztrációs munkát jelentett, továbbá a kezelése is nehézkes volt, mivel pl. szükség lett volna anyagok szerinti kigyüjtésre, tehát más csoportositásra mint amiben ezen raktári kartonok voltak.

Kellett tehát egy olyan rendszer, amellyel csökken az adminisztrativ munka, gyorsan visszakereshetők egy adott szalag adatai, illetve egy szalag tartalma alapján a szalag száma, továbbá "tetszőlegesen" visszanyerhetőek a szalagok jellemző adatai.
El is készült egy programcsomag, mely nagyjából megfelelt a célnak. Hibái a következők voltak:

a./ nem szüntette meg a manuális munkát /továbbra is be kellett irni az adatok változtatását egy kódlapra, mely lyukasztásra került s erről végezte egy program a karbantartást/

b./ nem volt naprakész állományunk, mivel havonta egyszer történt karbantartás.

c./ Az adatrögzitésnél léphettek fel hibák, - ellyukasztások-, melyeknek csak egy részét lehetett kiküszöbölni.

Tehát egy jobb megoldást kellett keresnünk.

Eközben kezdtünk foglalkozni a Mágnesszalag Hibastatisztikát készitő rendszerrel. Javitottunk benne, megváltoztattuk a kiiratást, s eközben merült fel az ötlet, hogy próbáljuk meg az ESTV file-t felhasználni egy fejlettebb mágnesszalag nyilvántartási rendszerhez. Ezen file azoknak a szalagoknak a hibastatisztikai adatait tartalmazza, amelyekre OPEN parancs lett kiadva. Ezen adatok a következők:

- a felirás ill. olvasás ideje /nap, óra...sec/; mely fizikai egységen volt a szalag; a hibák számát hibatipusonként, a SIO-k számát és a VOLUME-ot. /egyes rendszerek a blokkhosszat is tartalmazzák/.

A teljes ESTV rekordban 20 byte szabad hely van. Ide kellett beiratnunk az illető VOLUME-n szalag file cimkéjét, a file sorszámát, a kötet sorszámát és a file törlésének dátumát. Ezt OPEN időben végeztük el. /OPEN tranziensek kibővitése/.

A hagyományos hibastatiszikai rendszerben, ettől az ESTV-FILE-ról telitődése esetén "VOLUME" szerinti listát lehetett kérni /hozzáford. szerinti listát/.

A hardware osztály ezeket az információkat a mágnesszalagegységek hibáinak feltárására használja fel. Pontosabban csak használná, mivel a listán az egységekre vonatkozó hibák szét vannak szórva. Használhatóbb lenne egy olyan táblázat, mely egységenként mutatja, hogy pl. a 280-as egységre egy adott időszakban összesen hány hiba esik, adott SIO szám mellett. Ezért "VOLUME" szerinti listát nem készitettünk, hanem csak egy ilyen összevont táblázatot.

Az ESTV file kb. 1 hét alatt telik meg. Tehát az előbb emlitett összevont lista max. 7 napos időszakra terjed ki. Hardware osztályunknak egy olyan kérése volt, hogy szeretnének ilyen tipusu információt kapni, de nagyobb, 2-4 hetes időszakról. Ezt ugy oldottuk meg, hogy az ESTV file első 5 rekordját felhasználtuk ezen adatok tárolására. Ehhez viszont át kellett irni a supervisor TEBV blokkját, hogy ezen rekorodkat ne irja fölül a rendszer. A könnyebb hivatkozás miatt ezt az 5 rekordot ESTS file-nak nevezetm, bár nem alkotnak külön file-t.

Igy tehát a mágnesszalagok nyilvántartása mellett ezeket a hibastatisztikai funkciókat is el kellett látni.
Az ezen szempontok szerint a kialakitott rendszer öt részből áll.

1./ Az első az ESTV-FILE creálása. Ez az ESTS file miatt szükséges, mivel ezen rekordok más felépitésüek, s nem hexadecimális nullára kell initelni őket.

2./ A második az alap mágnesszalag törzsállományt /nyilvántartási törzsállományt/ létrehozó program. Ez a régi nyilvántartási rendszer törzsállományából készíti el az uj törzsállományt.

3./ A harmadik program csak hibastatisztikai funkciót lát el. Feladata az ESTS file kiiratása.

   a./ kiirja, hogy a lekérdezés idejéig az egyes egységekre hány hiba történt hibatipusonként, illetve hány SIO parancs lett kiadva rá.

   b./ a másik részben százalékos kiértékelést ad, amely egységenként megadja a 1ØØ db SIO-ra eső hibák számát.

4./ A negyedik, a tulajdonképpeni fő rész végzi el az update-olást, a törzsállomány és az ESTV file között.

A törzsállomány egy rekordja a következő felépitésü: tartalmazza

- a szalag nyilvántartási számát /törzsszám/
- a file cimkéjét /HDR1/
- a file felirásoakor keletkezett hibák számát hibatipusonként
- a szalagon addig keletkező összes hibák számát
- a file fairása közben kiadott SIO-k számát
- a szalagra addig kiadott összes SIO-k számát
- a file sorszámát a szalagon
- a kötet sorszámát
- azon egység fizikai cimét, amelyen a felirás történt
- a felirás dátumát
- a törlés dátumát
- a szalagra kiadott összes OPEN-ek számát
- munkaszámot.

Az update-olás a következőképpen történik:

- Ha olyan szalag információi érkeznek, amelyek már szerepelnek egyszer a törzsállományban, akkor két verzió végrehajtása lehetséges:

  a./ ha outputként szerepel a szalag, akkor update-olásra kerül a teljes rekord a nyilv. tart.-i szám kivételével.

  b./ input esetén a felirás és törlés dátuma; továbbá a feliráskor keletkező hibák száma, a feliráskor kiadott SIO-k száma; azon egység fiz. cime, melyen a felirás történt változatlan marad.

Ha a szalag inf.-i nem szerepelnek a törzsállományban, akkor uj rekordként felveszi.

5./ Az ötödik program a NOLABEL-es szalagokat veszi be a törzsállományba, kártya inputból.
Végül az utolsó részhez tartoznak a listázó programok, melyek a különböző információkat adják az osztályoknak. /folyamatelőkészités, raktár, gépkezelők/.
Ezek a következők lehetnek:

- teljes lista az állományról
  tartalma: VOLUME
  file sorszám /max. 9/
  munkaszámot /az adott file mely anyaghoz tartozik/
  a cimkét
  a kötet sorszámát
  a felirás dátumát
  a törlés dátumát
  az aktuális file létrehozásakor keletkező hibák számát
  a CUU-t /mely egys.-en történt a felirás/
  egyéb megjegyzéseket / TOROLHETO'/
- munkaszámos lista
  tartalmazza munkaszámok szerint a teljes lista adatait.
- törölhető szalagok listája
- VOLUME szerinti keresés lehetősége, illetve cimke szerinti VOLUME keresése
- információ a szalagok állapotáról
  tartalmazza: VOLUME
  a szalagra kiadott össz. SIO-t
  a szalagra kiadott össz. OPEN-ek számát
  az előfordult összes hibák számát.

Végül meg kell emlitenem a rendszer továbbfejlesztési lehetőségét, mely már folyamatban van.

A feltételezés szerint a VOLUME tartalmazná a szalag nagyságára illetve százalékos értékére vonatkozó információt. Erre a VOLUME első két karakterét hasz-

nálnánk fel. Itt az átállási idő elég nagy, igy kb. 1 év mulva várhatunk ilyen tipusu értékes információkat. Ennek fő jelentősége egyrészt abban van, hogy ebből reálisabban lehetne kiszámlázni a felhasználó vállalatok felé a mágnesszalag bérleti dijakat, másrészt jobb információt kapnánk szalagállományunkról.

Befejezésként a rendszer elsődleges előnyeként minősitjük, hogy sikerült automatizálnunk és igy eredményként ON-LINE nyilvántartási rendszert létrehoznunk. Legnagyobb problémákkal akkor álltunk szemben, amikor OPEN időben kellett beavatkoznunk az operációs rendszer alapvető funkcióiba.
R-40-es számitógéppel és DOS-POWER operációs rendszer felügyelete mellett dolgozunk.

AZ R10-ES MM-RENDSZER FEJLESZTÉSE SORÁN SZERZETT TAPASZTALATOK

Molnár Máté
Számitógépalkalmazási Kutató Intézet

Jelen előadásban az MM-rendszer négyéves történetét /1974-1978/ és a rendszer fejlesztése során szerzett tapasztalatokat szeretnénk bemutatni. Ennek érdekében először vázlatosan magát a rendszert ismertetjük /részletesebben lásd /1/, /2/ és az intézetünkben beszerezhető dokumentációkat/. Ezután az R10-es realizáció szerkezetéről lesz szó, majd beszámolunk a rendszer további fejlesztéseiről és a rendszer használata során szerzett tapasztalatokról /a rendszer értékelése/.

1. Az MM-rendszer vázlatos ismertetése

Az MM-rendszer arra a tényre épül, hogy az információrendszerek implementálása során elkészitett programok sok közös vagy hasonló elemet tartalmaznak. Ennek egyik oka az, hogy a programok által megoldott feladatok tekintélyes része bizonyos tipikus osztályokba sorolható. Igy beszélhetünk pl. hibaszürő, aktualizáló, tablózó, stb. programokról. A másik ok abban áll, hogy a több szekvenciális fájlt feldolgozó programok számára lehet találni egy olyan olvasási és feldolgozási algoritmust, amely az esetek nagy részében alkalmazható, és alkalmazásával a program szerkezete egyszerübbé válik.

Az MM-rendszerben ezeknek a közös elemeknek vázalgoritmusok, un. tipusprogramok felelnek meg. Természetesen a konkrét feladatok esetében ezeket a vázalgoritmusokat ki kell egésziteni különböző paraméterekkel és felhasználói algoritmusokkal. Ezen paraméterek és kisebb algoritmusok megadása egy nyelven, az AP /Adaptáló Paraméterező/ nyelven történik. Az AP-nyelvü szöveget AP-fájl-

nak nevezzük. Az AP-fájlból egy forditóprogram /az APTRA/ készit gépi modult, amely futás előtt összeszerkesztődik a tipusprogram és a futtató rendszer moduljaival.

A fentiekből következik, hogy az AP-nyelvnek két követelményt kell teljesitenie. Egyrészt alkalmasnak kell lennie arra, hogy segitségével különböző paramétereket lehessen megadni a tipusprogramok számára. Másrészt alkalmasnak kell lennie arra is, hogy vele algoritmusokat irjunk le. Az AP-nyelv, illetve az AP-fájl szerkezete biztositja ezeknek a követelményeknek a teljesülését.

Az AP-fájl szakaszokból áll. Az alábbiakban ismertetjük, hogy milyen szakaszok fordulhatnak elő egy AP-fájlban.

a/ D-szakasz, a ki- és bemenő fájlok és rekordjaik leirására;
b/ T-szakasz, a készitendő tabló leirására;
c/ L- és M-szakasz /logikai szakaszok/, különböző logikai feltételek leirására;
d/ P-szakasz, értékadások és aritmetikai müveletek leirására;
e/ S-szakasz, könyvtári eljárások hivására;
f/ K-szakasz, egyéb paraméterek megadására.

A szakaszok AP-rekordokból állnak. Az AP-rekordoknak a szakaszon belül számazonositójuk van. Az AP-fájl tetszőleges rekordjára egyértelmüen hivatkozhatunk az őt tartalmazó szakasz azonositójának és a rekord szakaszon belüli azonositójának megadásával, pl. DØ3 jelenti a D-szakasz Ø3-as azonositójú rekordját. A továbbiakban ezt az azonositót AP-rekordazonositónak nevezzük.

Ez a szerkezet biztositja a tipusprogramok számára a paraméterek egyértelmü leszedését, ugyanis az egyes tipusprogramok a különböző paramétereket meghatározott azonositójú AP-rekordokból várják. Igy pl. az UD5 nevü tipusprogram azon kulcs leirását, amely szerint a bemenő fájlok rendezve vannak, a KØ1 azonositójú AP-rekordból várja.

Felhasználói algoritmusok megadása AP-rekordazonositókból álló sorozatok segitségével történik. Ugyanis ilyenkor a sorozatból hivott AP-rekordok müveletek leirását tartalmazzák, amelyek végrehajtódnak az adott sorrendben. Lehetőség van uj sorozatok hivására /K-rekordokon keresz-

tül/, elágazások létrehozására /logikai rekordok segitségével/, ciklusok képzésére, stb. Az AP-rekordokban megadható müveletek: kimenő fájlra rekord irása /D-rekordokban/, sor összeállitása és kiiratása tablóra /T-rekordokban/, értékadások, aritmetikai müveletek /P-rekordokban/ és a rendszer által tartalmazott vagy a felhasználók által ASSEMBLER nyelven irt eljárások hivása /S-rekordokban/. Az AP-rekordazonositókból álló sorozatok által megadott felhasználói algoritmusok a tipusprogramok futásának bizonyos pontjaiban hajtódnak végre. A megfeleltetés ezen belépési pontok és az AP-fájlban lévő különböző rekordazonositó-sorozatok között általában úgy jön létre, hogy az egyes pontokhoz K-rekordok azonositói vannak hozzárendelve: ezekben a pontokban az adott azonósitójú K-rekordban lévő sorozat hajtódik végre. Igy pl. az OM1 tipusprogramban a program futásának az elején a K10 azonositójú rekordban lévő sorozat, a futás végén pedig a K99 azonositójú rekordban lévő sorozat hajtódik végre. A program futása közben a KØ1, KØ2, ill. KØ3 azonósitójú rekordok valamelyikében lévő sorozat hajtódik végre olvasási ciklusonként, az éppen beolvasott rekordtól függően.

2. Az MM-rendszer R10-es realizációjának szerkezete

Az alábbiakban az MM-rendszer R10-es realizációjának jelenlegi /legutolsó fejlesztés utáni/ szerkezetét ismertetjük.

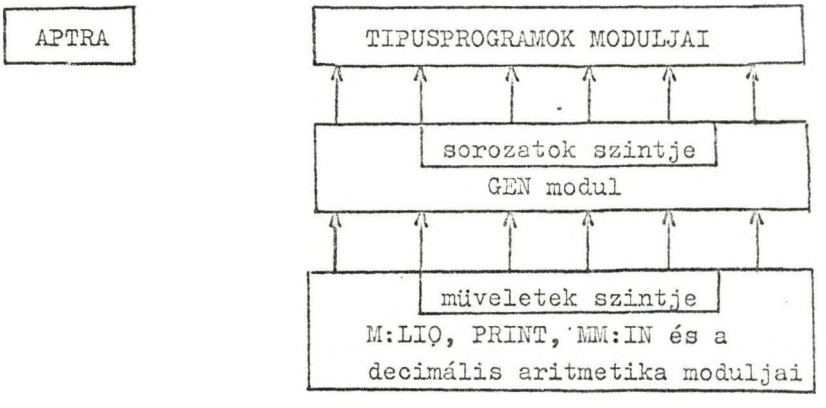
Az előzőekből látszik, hogy az MM-rendszert realizáló szoftvernek a következő feltételeket kell kielégitenie:

- tartalmaznia kell egy forditóprogramot az AP-fájlok lefordítására;
- tartalmaznia kell a tipusprogramok vázalgoritmusait;
- tartalmaznia kell az AP-rekordazonositó-sorozatok végrehajtására szolgáló mechanizmust.

Ez utóbbi mechanizmust megvalósitó modulokat két csoportra oszthatjuk. Az első csoportba azok a modulok tartoznak, amelyek maguknak a sorozatoknak a végrehajtásával foglalkoznak /sorozatok szintje/. A második csoportba pedig azok a modulok tartoznak, amelyek a sorozatokban hivatko-

zott AP-rekordokban lévő müveletek végrehajtását végzik /értékadások, I/O, tablósorok szerkesztése és nyomtatása, logikai állitások kiértékelése stb. az egyes müveletek szintje/.

Az alábbi ábra az R10-es MM-szoftver szerkezetét mutatja.



A továbbiakban röviden ismertetjük az egyes programok, illetve modulok funkcióját.

APTRA: forditóprogram az AP-fájlok forditására. CDL nyelven iródott. A forditás eredménye egy BT modul.

GEN modul: vezérli az AP-rekordazonositó-sorozatok végrehajtását. Ez a modul hivja a megfelelő müveleteket végrehajtó modulokat, továbbá kezel egy vermet, amelybe uj sorozatra való áttéréskor a visszatérési cimek bejegyződnek. Rekurziv modul.

M:LIO modul: az I/O müveleteket végző modul. Olvassa a bemenő fájlok soronkövetkező rekordjait, illetve elvégzi a rekordok kiirását a kimenő fájlokra.

PRINT modul: a tablósorok összeállitását és kinyomtatását végzi. Ez a modul végzi a többfokozatú összegek képzését és a fokozatváltások ellenőrzését is.

MM:IN modul: interpreter a P-rekordokban lévő értékadások, illetve aritmetikai müveletek végrehajtására és a logikai rekordokban lévő állitások kiértékelésére. Ez a modul hivja a decimális aritmetika moduljait.

3. A rendszer története /1974-1978/

Az MM-rendszer R10-es realizációja 1974-ben készült el, nemzetközi bevizsgálása pedig 1975 elején történt meg. A rendszer ekkor három tipusprogramot tartalmazott. A TT3 tipusprogram egy bemenő fájl rekordjait osztályozta paraméterként /logikai rekordokban/ adott feltételek szerint. Vele lehetett elvégezni elsődleges adathordozón /lyukkártyán vagy lyukszalagon/ lévő fájlok rekordjainak hibaszürését, fájl tablózását, stb. Az UD2 tipusprogram egy törzsfájl aktualizálását végezte változásfájl alapján. A harmadik tipusprogram /BR1/ gépipari jellegü vállalatoknál előforduló speciális feladat, a darabjegyzéklebontás elvégzésére volt alkalmas. Ez a készlet még 1975 folyamán kiegészitődött a CR1 tipusprogrammal, amely képes volt három bemenő fájl rekordjai közti párossági helyzetek megállapitására.

A rendszer ezen első készletének használata során a következő hiányosságokat észleltük:

- Az UD2 tipusprogram által tartalmazott /törzsfájlok aktualizálására szolgáló/ algoritmus nem volt "telitalálat", a feladatosztályra igazán jellemző algoritmus. Aktualizálási algoritmusunk tisztulása után, még 1975-ben elkészült az UD3 tipusprogram, amely már egy olyan algoritmust tartalmazott, amely egyrészt könnyen leirható és megjegyezhető, másrészt a feladatosztályra is jellemző. Ezen az algoritmuson /apróbb módositásoktól eltekintve/ azóta se kellett változtatnunk.
- Az előbbinél sulyosabb hiányosság volt, hogy az MM-rendszer első készletében a felhasználói algoritmusok megadására csak korlátozott lehetőség volt és az őket leiró AP-rekordazonositó-sorozatok végrehajtásának módja tipusprogramonként eltérő volt.
- A rendszer tipusprogramjai nem fedték le az összes, gyakorlatban előforduló feladatot.

E két utóbbi hiányosság megszüntetésére készült el az MM-rendszer második készlete. A második készlet létrejöttét célzó fejlesztés eredményeképpen az AP-nyelv alkalmassá vált bonyolult algoritmusok leirására és az

AP-rekordazonositó-sorozatok végrehajtása is egységessé vált. A második készlet tartalmaz egy új tipusprogramot is /OM1/, amelynek vázalgoritmusa egy olyan általános olvasási algoritmus, amely a több szekvenciális fájlt feldolgozó programokban jól használható.

Azt mondhatjuk, hogy az MM-rendszer második készletében lettek kijavitva a tipizálás túlsúlyozásából származó hibák, mert a második készlet adta meg a lehetőséget a tipikus vázalgoritmusok megfelelő felhasználói algoritmusokkal való kiegészitésére, továbbá itt jelent meg egy olyan "tipusprogram", amely nem egy speciális feladatosztály számára készült, hanem általában a több szekvenciális fájlt feldolgozó feladatok megoldására.

Jelenleg két olyan fejlesztés folyik, amely a második készlet logikai szintjét érinti. Ezek közül az első: a rendszerbe beépitjük az indexelt-szekvenciális fájlok kezelését.Eddig a rendszer csak szekvenciális fájlokat tudott kezelni. /A DIFKE nevü eljáráscsomag segitségével lehetett ugyan nem szekvenciális fájlokat is használni, de ezt nem tekinthettük kielégitő és végleges megoldásnak./

A második fejlesztési munkánk egy GS nevü /General Sequential File Treating/ tipusprogram létrehozására irányul. A tipusprogram az OM1 olvasási algoritmusa szerint olvassa a rekordokat a maximálisan három bemenő fájlról, de egy-egy rekord beolvasása után a felhasználó által paraméterként megadott relációk fennállásától függően hajtódnak végre különböző /AP-rekordazonositó-sorozatokkal leirt/ felhasználói algoritmusok. A paraméterként megadott relációk vonatkozhatnak arra, hogy az aktuális rekord kulcsával van-e azonos kulcsú rekord a többi fájlban /párosság/, illetve hogy az azonos kulcsú rekordokból álló csoportra nézve az éppen beolvasott rekord első, közbülső vagy utolsó-e.

A jelenleg folyó harmadik fejlesztési munkát az tette szükségessé, hogy az MM-rendszer R10-es realizációjának elkészülte /1974/ óta a gép alapszoftvere fejlődött: uj monitorok és fájlkezelő-rendszerek jelentek meg. Elkerülhetetlenné vált ezen uj szoftver-elemek beépitése a rend-

szerbe. Ennek eredménye az lesz, hogy az MM-rendszer fájlkezelése kompatibilis lesz a többi R10-es rendszer fájlkezelésével, sőt mágnesszalagos fájlok vonatkozásában még a nagyobb ESZR gépek fájlkezelésével is.

Közben 1977-ben létrejött az a keret is, amelyen belül az R10-es MM-rendszer jelenleg is terjesztődik. Ez a SZTAK /Számitógépes Tipusrendszereket Alkalmazók Köre/, A SZTAK-ba belépő vállalatok számára intézetünk biztositja az MM-rendszer /és a hozzátartozó dokumentáció/ használatát, segitséget nyújt a rendszer üzembe állitásánál, tanfolyamokat tart a rendszer használatáról, továbbá konzultációs segitséget is nyújt. Jelenleg a SZTAK-nak 9 tagja van.

4. A rendszer értékelése

Az alábbiakban az MM-rendszert és R10-es realizációját szeretnénk értékelni. Értékelésünknek kétségtelenül hibája, hogy a rendszer fejlesztői végzik. Azonban intézetünk nem csak fejlesztője, de felhasználója is a rendszernek, továbbá közvetitjük a SZTAK-tagoknak hozzánk eljutott véleményeit is. Értékelésünk ki fog terjedni a rendszer fizikai jellemzőire /futási idők, helyfoglalás/, a tipusprogramok és az AP-nyelv /kényelmes/ használhatóságára és a felhasználói dokumentációra.

Eddigi tapasztalataink szerint a tipusprogramok futási ideje erősen összefügg azzal, hogy az őket paraméterező AP-fájl hány aritmetikai müvelet végrehajtását kéri. Ennek az az oka, hogy az R10 számitógép nem rendelkezik hardver decimális aritmetikával, az aritmetikai müveleteket programmodulokkal kell kiváltani. Ezen a területen lényeges javulás tapasztalható az R12 számitógépen, ugyanis a mult év folyamán a rendszerbe beépitettük az R12-es hardver aritmetika használatát. Eddigi tapasztalataink szerint a sok aritmetikai müveletet tartalmazó feladatok esetében, R12-számitógépet használva a futási idő töredékére csökken.

A rendszer helyfoglalásával kapcsolatban azt mondhatjuk, hogy egy nagy AP-fájl összeszerkesztve valamely tipusprogrammal és a futtató rendszer moduljaival, kb. 25

Kbájtot foglal el a memóriából. Programmodulok átlapolására /overlay/ csak korlátozott lehetőségek vannak. Feltétlenül negativumként kell említeni, hogy a rendszerhez tartozó segédprogramok /utility-k/ és az APTRA forditóprogram a futtatható programok könyvtárában /EP/ nagy helyet foglalnak el. Ennek oka részben az, hogy a segédprogramok egy része típusprogramok és könyvtári eljárások felhasználásával, AP-nyelven készült és igy az összeszerkesztett program a futtató rendszer olyan moduljait is tartalmazza, amelyekre az adott programnál nincs szükség. Ezen a téren a rendszernek még "tartalékai" vannak, és a most folyó fejlesztések eredményeképpen is a rendszerhez tartozó EP-könyvtár mérete várhatóan csökkeni fog.

A típusprogramok alkalmazhatóságával kapcsolatban a /véleményünk szerint/ két legsikerültebbel, az UD5 és OM1 típusprogramokkal szerzett tapasztalatokról számolunk be. Megjegyezzük, hogy ez a két típusprogram elegendő csaknem az összes, gyakorlatban előforduló feladat megoldására /kivéve olyan speciális feladatokat, mint pl. a darabjegyzék-lebontás/.

Az UD5 típusprogram törzsfájlok változásfájl alapján történő aktualizálására készült. Vázalgoritmusa segitségével, minimális AP-nyelvü paraméterezési munkával /gyakorlatilag komolyabb felhasználói algoritmusok nélkül/ a legtöbb ilyen jellegü feladat megoldható. A típusprogramnak külön előnye, hogy a kezdő szervezők és programozók számára nehezen rendszerbe szedhető aktualizálási és hibahelyzetekre belépési pontokat tartalmaz, és igy az aktualizálási müvelet tervezőjének és programozójának mintegy a kezét vezeti.

Az OM1 típusprogram maximálisan három, rendezett bemenő fájlt tud feldolgozni. A típusprogram ugy tekinti a bemenő fájlokat, mintha egyetlen fájlba lennének összerendezve /merge/, de az egyes rekordok beolvasása után különböző felhasználói algoritmusokat hajt végre, attól függően, hogy a beolvasott rekord melyik fájlról való. Ezen egyszerü vázalgoritmus segitségével igen bonyolult logikájú feladatok megoldhatók kisméretü és áttekinthe-

tő AP-fájl megirásával. Igaz, hogy a tipusprogram ilyen jellegü használatához "érezni" kell a merge-szerü olvasási algoritmust /a készülő GS tipusprogram segitségével ezek a feladatok még egyszerübben megoldhatók lesznek/.

Az AP-nyelv szerkezetéből következik, hogy benne az algoritmusok megadása nem közvetlenül az utasitások egymásután irásával történik, hiszen értékadásokra, feltételes ugrásokra, I/O müveletekre csak azon AP-rekordok azonositóival hivatkozhatunk, amelyekben a müvelet részletesen le van irva. Azt is mondhatnánk, hogy az AP-rekordok megadásával egy absztrakt gépet létesitünk, amelynek utasitásai az AP-rekordazonositók, az utasitások szemantikája pedig az egyes AP-rekordokban jellegüknek megfelelő nyelven /az értékadások nyelvén, a tablósorok leirásának nyelvén, az I/O müveletek nyelvén, a logikai állitások nyelvén/ van megadva. Ez, továbbá az a tény, hogy rövid AP-rekordazonositó-sorozatokkal kényelmes dolgozni, lehetővé teszi /sőt kényszerit arra/, hogy a felhasználói algoritmusok szintekre bontva, strukturáltan legyenek megadva és leirva.

A következőkben az AP-nyelv két hátrányos tulajdonságára mutatunk rá. Az első annak a következménye, hogy egy algoritmus leirásánál egy müveletre csak az őt leiró AP-rekord azonositóján keresztül lehet hivatkozni, azaz nem lehet a müveletet az adott helyen közvetlenül leirni. Ennek az a következménye, hogy lokális jelentőségü értékadások /kezdőérték, flag beállitása, ciklusváltozó növelése, stb./ és feltételes ugrások külön AP-rekordok megirását követelik meg. Ez néha egy AP-fájlon belül az AP-rekordok elszaporodásához vezet.

A másik hátrányos tulajdonságnak az AP-rekordok azonositóinak kötött szintaktikája az oka /szakaszazonositó + rekordazonositó/. Ez megakadályozza, hogy az AP-rekordnak a bennük leirt müveletet jellemző neveket adhassunk. Ha erre lehetőség lenne, az AP-fájl még inkább "olvashatóbbá" válna.

Az MM-rendszer felhasználói dokumentációjával kapcsolatban /néhány más kisebb és könnyen javitható hiányos-

ság mellett/, arra a problémára mutatunk rá, hogy a tipusprogramok vázalgoritmusainak leirására nincs egységes eszközünk vagy módszerünk. A probléma megoldásaként szóbajöhetne egy kötött dokumentáció-szerkezet vagy akár egy formális leirási eszköz /nyelv/ is. Természetesen ha ez utóbbi területen eredményt tudnánk elérni, az használható lenne az MM-rendszeren kivül is, az információrendszereket megvalósitó programok specifikációjának és fejlesztői dokumentációjának elkészitésében is.

Az MM-rendszert és a SZTAK-ot részletesebben ismertetik:

/1/ Siklaky István és munkaközössége: A Management Modul Rendszer. Közgazdasági és Jogi Könyvkiadó 1976.

/2/ Molnár Máté: Az MM-rendszer és R10-es relációja. Információ-Elektronika 1978/2.

/3/ Siklaky István: Megalakult a Számitógépes Tipusrendszer Alkalmazók Köre. Számvitel és Ügyviteltechnika 1978. április.

PL/1 COMPILER AZ ICL SYSTEM 4-RE

Müller Henrik
Magyar Hiradástechnikai Egyesülés Számitástechnikai
és Szervezési Központja

Nem volt PL/1 forditóprogram az ICL System 4-es gépére. Az operációs rendszer tervezői gondoltak a PL/1-re, mert adtak nyelvkódot és azonositót a forditóprogram számára, de nem készitették el a forditót.

Az R-es gépek terjedésével egyre népszerübb Magyarországon a PL/1 nyelv. Ezért elhatároztuk, hogy PL/1 forditót készitünk a System 4-re. Fő szempont volt, hogy az R-es és a System 4-es PL/1 a lehető legnagyobb mértékben kompatibilis legyen. Ugyanakkor azt is figyelembe kellett venni, hogy a COBOL-t használó programozókat csak akkor lehet PL/1-re tanitani, ha a PL/1 forditó kényelmesebb, és több lehetőséget nyújt, mint a COBOL.

A forditó tervezésekor az R-es gépeken futó PL/I-D compilerből indultunk ki. Ennek mouláris felépitése lehetővé tette a lépésenként történő adaptálást. A munkát két fő fázisra bontottuk:

1/ Készüljön olyan forditó, mely System 4-en fordit, de az outputja R-es gépeken tud futni /kereszt-forditó/

2/ Készüljön olyan forditó, mely System 4-en fordit, és az outputja is System 4-en futtatható

A PL/1 nyelv gepfüggetlen részeit sikerült teljesen R-kompatibilisan megvalósitani. Az I/O rendszert, a megszakitási rend-

szert, a job controlt, az overlay technikát és a több modulos programozást viszont át kellett tervezni.

Fájlkezelés

A System 4 logikai fájlkezelő rendszere sok szempontból fejlettebb az R-DOS-nál. Elmaradt a MEDIUM opció, helyére periférianevek kerültek:

CARD

PTAPE

PRINTER

TAPE

DISC

INWELL

Növeltük a default opciók és attributumok körét, pl. az INDEXED implikálja a DIRECT attributumot, ez meg a DISC opciót.
A CLOSE utasitásban megengedtük mindazon kulcsszavak használatát, melyek a System 4-en léteznek:

UNLOAD

REOPEN

REREAD

REWIND

NOREWIND

RELSE

TRUNC

A Standard Input fájlt a J operációs rendszer // PARAM adataira helyeztük, ezáltal lehetővé tettük magasszintü nyelvből a futási paraméterek elérését.

A Standard Output fájl újradefiniálással 160 karakteresre bővithető. PRINT attribútumú fájlok mágneses közegre helyezve

blokkolhatók. Lassú perifériákra alapfeltételezés a BUFFERS 2 . Indexelt fájlok esetén lehetőség van rekord törlésére REWRITE utasitással, ha a rekord első karaktere X'FF'.

Megszakitáskezelés

A megszakitáskezelés új rutinjai több információt adnak a hibáról, mint az R-es gépeken. Több új SIGNAL-kód született, és külön könyvtári szubrutinnal ezek elérhetővé válnak a programozó számára is.

Job Control

A job összeállitást olyan módon alakitottuk ki, hogy a PL/1 fordító a többi fordítóhoz hasonlóan a Trials rendszer alatt fusson. Előrelépést jelent az, hogy a // COMPILE kártyán is megadhatunk fordításvezérlő opciókat.

Overlay használat

Az overlay technika a System 4 composer-lehetőségeinek figyelembevételével került kialakitásra. Minden overlay-t kétjegyü decimális szám azonosit. Egy programban max. 99 overlay lehet.

Több modulos programok

Adatok átadása egyik modulból a másikba EXTERNAL változókkal jöhet létre. De a System 4-es Usercode assembler nem tud common data blokkot generálni, csak common areat. Igy PL/1 és Usercode kapcsolatban egyszerübb a paraméterrel történő adatátadás. A Usercode modulok kényelmes programozhatósága kedvéért három új makró készült:

PNTRY	Usercode modul belépési pontja
PCALL	Usercode-ból ezzel hivható PL/1 rutin
PEXIT	Usercode modul kilépési pontja

Ezek teljesen megoldják a PL/1 interface gondját, akár PL/1 - Usercode, akár Usercode - PL/1 a hivási irány. MAIN modul egyaránt lehet Usercode vagy PL/1.

A forditó felépitése

A forditó száznál több modulból áll, háromszintü overlay szerkezettel. Az I/O szervezés a ROOT szegmensben van, ugyanide került egy elég jelentős méretü ön-diagnosztizáló rutin is. A forditó hibája esetén ez post mortem jellegü dumpot készit, és kiirja a három diszkes munkafájl tartalmát is. E rutinnak köszönhető, hogy a forditó mindössze egy év munkájával készült el.

PROGRAMRENDSZEREK ÜZEMELTETÉSÉNEK KÉRDÉSEI

Nagy Elemér - Gál György
JATE Kibernetikai Laboratórium

Ebben az előadásban először a batch környezetben üzemeltetett programrendszerekkel szemben támasztott különböző igényekkel és ezek kielégitésének módjával foglalkozunk. Ezután kisérletet teszünk néhány általános szempont és összefüggés feltárására, amelyek alapját képezhetik a hatékony üzemeltetés eszköztára pontosabb meghatározásának. Végül konkrét programrendszer kapcsán vázlatosan ismertetjük a konkrét követelményeket és az üzemeltetési megoldást. A leirtakban azokat az általános tapasztalatokat és felismeréseket próbáljuk rögziteni, amelyeket egy viszonylag nagy - sok programból álló és nagytömegü adatot felhasználó - közel két éve naponta futó adatfeldolgozási rendszer üzemeltetése során nyertünk.

1. Programrendszer üzemeltetés, hatékonysági szempontok

A batch üzemben egy programrendszer - üzemeltetése során - rendszeresen együtt futó, valamely feladat megoldására szolgáló programsorozatként tekinthető. Ez a meghatározás független attól, hogy milyen gépen, milyen operációs rendszerben üzemel a rendszer, mi a megoldandó feladat jellege, független a tárolt adatállományok /fileok/ szervezésétől; különálló adatfileokat, vagy integrált adatbázist kezelő, felhasználó programokról van-e szó.

Megj.: Az interaktiv rendszereket itt eltérő tulajdonságaik miatt nem tárgyaljuk. Rendszerint az ilyen rendszereknek is van batch üzemben müködő része.

Annak meghatározására, hogy a hatékonyság általában mit jelent nem vállalkozunk. A továbbiakban a "hatékonyság" szót a szokásos értelemben használjuk és megvizsgáljuk

néhány összetevőjét az üzemeltetésben résztvevő három csoport - szervezet - igényeinek tükrében.

A felhasználó - vagyis a futási eredményeket hasznosító szervezet, intézmény - a rendszer specifikációjának megfelelő inputra meghatározott időn belül formailag és tartalmilag helyes outputot vár. A felhasználó hatékonysági szempontjai a fenti igények minél nagyobb százalékban való kielégítése, esetenként a "meghatározott idő" rövidítése és a géphasználatért fizetendő összeg csökkentése.

A programrendszert üzemeltetők, akik esetenként vagy a felhasználó, vagy a gépet üzemeltető szervezethez tartoznak - igénylik a "closed shop" futtatás lehetősége mellett is azt, hogy lehetőleg egyszeri géphezfordulással - biztonságosan - jussanak az eredményekhez. Fontos, hogy a rendszer lefutása követhető legyen, a rendszer komponensei minél kevesebb speciális karbantartást igényeljenek, háttérgépen való futtatáskor minél kevesebb üzemeltetői beavatkozásra legyen szükség.

Multiprogramozott gépi környezetben a gépüzemeltetés a programrendszer hatékonyságát mint a teljes üzemeltetési rendszer hatékonyságának egy tényezőjét vizsgálja. Igy a programrendszer üzemeltetőinek az alábbi szempontokat kell figyelembe venni.

- A programrendszer futtatásához a szükséges méretü tár és az input-output eszközök biztosithatók legyenek, ugyanakkor csak annyi erőforrást foglaljon le, mint amennyi feltétlenül szükséges. Ennek a szempontnak főként kisebb kiépitettségü gépek ill. nagy rendszerek esetén van különleges jelentősége.
- A programsorozat lehetőleg jól illeszkedjen a batchben általában előtte vagy utána futtatott más rendszerek alapkövetelményeihez, futtatható legyen az operációs rendszer általánosan használt variánsával.
- Az üzemeltetett gép paramétereinek megfelelően a központi egység és az autonom I/O csatornák működését igénylő müveletek aránya lehetőleg egyenletes gépterhelést biztositson
- A programsorozat átfutásának gyorsitása érdekében a programok közötti - eltérő adathordozó igény stb.

miatt fellépő - "holt idő"-t az egyes lépések tervezett összehangolása csökkentse.

- A futtatás minél kevesebb gépkezelői beavatkozást /futtatásvezérlést, adathordozó cserét, konzolkommunikációt/ igényeljen.
- A gép - az operációs rendszer - üzemeltetése során előforduló hardware, software hibák esetén legyen lehetőség a programsorozat ujrainditására a már elkészült részek ujbóli végrehajtása nélkül.
- A programrendszernek a számitásba jövő, nem szükségképpen azonos paraméterekkel rendelkező háttérgépen való üzemeltetése is lehetőleg hatékony rendszerüzemelést engedjen meg. Ilyen helyzet saját gépen is előállhat részleges rendszerkiesés esetén.

Az itt felsorolt követelmények közül a különböző környezeti feltételektől függően más és más válhat elsődleges fontosságuvá. A környezeti feltételek változásával a változó követelményekhez kell igazodnia az üzemeltetett programrendszernek is.

A programrendszernek egy sor a fent emlitett követelményekkel kapcsolatos tulajdonsága már az egyes programok elkészitésekor rögzitődik. Ilyen a programok szemantikus helyessége, az egyes programok minimális konfigurációja, a CPU - I/O müveletek aránya, a programok eredményei ellenőrizhetőségének lehetőségei, stb. Az üzemeltetés során dől el, hogy mennyire lett hatékony a rendszer a tervezői és programozói munka eredményeként.

Mi most nem azzal foglalkozunk, hogy a programrendszer tervezése, létrehozása, tesztelése, változtatása során hogyan lehet a fenti követelményeket kielégiteni, hanem az emlitett fázisok többségén már tuljutott rendszer üzemeltetésének eszközeit vizsgáljuk. Ezek alapjainak tisztázásával szeretnénk éppen azt elérni, hogy az üzemeltetést megelőző fázisok, főként a tervezés során már előre figyelembe vehetők legyenek.

2.1. Feldolgozási rendszer, programelem, feldolgozási feladat

Egy feldolgozási rendszer - összetevőit tekintve - két fő részből áll, a programelemekből és a programelemek által manipulált tárolt adatokból. Az alábbiakban megvilágitjuk hogy milyen értelemben használjuk a programelem és az adatok fogalmát.

Egy adott - szokásos - programozói feladatnál a programozó számára az egység a program /eljárás/. Egy programnak viszont lehet több különböző funkciója, amelyekben néha semmilyen közös vonás nincs azon kivül, hogy ugyanazon programnak különböző paraméterezésü futtatásaként nyerhetjük. Egyszerü példa erre az esetre egy olyan program, amely két különböző tárolási módu /esetleg különböző rekordszerkezetü/ fileból készit listát. /Különböző rekordszerkezet esetén a közös mezőkkel dolgozik/.
Azt, hogy a lehetséges két file közül melyik lesz a tényleges input file, valamely futáskori paraméter /UPSI byte, operátori válasz, paraméterkártya stb./ rögziti le. Ilyen értelemben ez a program potencialitás, amely két lehetséges funkciót - programelemet - foglal magába. A potencialitás és a ténylegesség további szinteken is tárgyalható, ezzel nem foglalkozunk itt részletesen, csak azt emlitjük meg, hogy egy-egy programelem is potencialitás abban az értelemben, hogy e tényleges lefutás /a program nyoma/ függ az input file rekordjainak számától, az esetleges törölt /érvénytelen/ rekordok számától és előfordulási helyétől. Ezek ismeretében már egyértelmü a program logikai nyoma, ami még mindig potencialitás abban az értelemben, hogy a tényleges lefutást további tényezők is befolyásolhatják, például különböző sulyossági foku berendezés hibák /read check/.

Ebből a gondolatmenetből következik, hogy a program irásakor rögzitődik a lehetséges funkciók - programelemek - halmaza. A program előkészitése egy meghatározott végrehajtáshoz /futtatáshoz/ egy programelem kijelölését jelenti.

Megj.: A programelem tényleges fizikai nyoma kisebb-nagyobb mértékben véletlentől is függő és csak akkor azonos a logikai nyommal, ha minden a futáskor használt eszköz biztosan előirásszerüen müködik.

A továbbiakban programelemen értünk egy olyan önálló egységet, amelynek meghatározott komponensszámu, tipusu, szerkezetü és szervezettségü input adatrendszere van, s az adott feltételeket kielégitő input esetén a programok végrehajtásakor nagy valószinüséggel olyan outputot /adatrendszert/ kapunk, amelynek komponensszáma, tipusa, szerkezete és szervezettsége ismert.

A programelemekről /gyakorlati véletlen eseményektől eltekintve/ feltehetjük, hogy azonos inputra mindig azonos outputot adnak.

Nézzük most meg, hogy mit jelent egy feldolgozási rendszer egy feldolgozási feladata.

Ha P a feldolgozási rendszer programelemeinek a halmaza, akkor ki kell választani a feladat végrehajtásához szükséges programelemeket és meg kell határozni a végrehajtási sorrendet. Tetszőleges feldolgozási rendszer és feldolgozási feladat esetén nyilván sem exisztencia, sem unicitás nem biztositott.

Ha a feladat megoldható P elemeinek segitségével, akkor minden megoldás esetén a feladat szemantikája és a programelemek tulajdonságai együttesen meghatározzák P-nek egy - a feladat megoldásához felhasználandó - P' részhalmazát és egy R relációt a P' részhalmazon, az alábbiak szerint: $a,b \in P'$ esetén aRb ha b csak az a lefutása után futhat.

Az R reláció nyilván tranzitiv és minden értelmes esetben érvényes, hogy a R b $\Rightarrow \neg$b R a ; a szimmetria ui. az un. halálos ölelést jelentené.

A P'-n az R relációt irányitott gráfként ábrázolva / a-ból b-be mutasson nyil, ha a R b/ hurokmentes gráfot kapunk, amely nem biztos, hogy összefüggő. Zárjuk le ezt a gráfot egy kezdő és egy záró elemmel a következő módon:

A kezdő elemből mutasson lezáró él minden olyan csucsba, amelybe nem mutat él és a záró elembe mutasson lezáró él minden olyan csucsból, ahonnan nem indul él.

/Nyitó és záró elemként üres programelemeket véve a két gráfban leirt tevékenység lényegileg ugyanaz. Az ekviva-

lencia formális definiciója és bizonyitása meghaladja a rendelkezésünkre álló kereteket./
Terjesszük ki az R relációt a lezáró élekre is, azaz a R b jelentse azt, hogy a-ból b-be vezet ut.
Ilyen módon minden feladathoz hozzárendelhető egy hálógráf /network/ modell, ahol a végrehajtás szempontjából mindig teljesülnie kell az alábbi végrehajthatósági alapfeltételnek:

A: a b programelem akkor futhat, ha minden olyan a, amelyre aRb teljesül - már lefutott.

Nyilván, ha minden programelem biztosan jól fut, akkor a feladat megfogalmazása ekvivalens lenne azzal, hogy minden csucsban lévő programelemet végre kell hajtani, mégpedig oly módon, hogy a végrehajtási sorrendre az A feltétel mindig teljesüljön. Ez nyilván megoldható, s általában több lehetséges bejárási sorrend kielégiti a fenti előirást. Tekintettel azonban arra, hogy egy-egy programelem lefutása /még ha feltételezzük is, hogy szemantikusan jó programelemekkel dolgozunk/ nem biztos esemény, igy a hálógráf bejárása sem az.
Maximalitásnak nevezzük azt az igényt, hogy minden olyan programelem elinditása megtörténjen, amelyre az A feltétel teljesül. /Nyilvánvaló, hogy néhány egyszerü hálószerkezettől eltekintve általában nincs maximális tulajdonságu bejárása a hálógráfnak - azaz nincs olyan sorrend, ami megtartja maximalitását attól függetlenül, hogy melyik programelem nem fut le./

Megj.: Minden bejárási sorozathoz kiszámitható a maximalitás valószinüsége az egyes programelemek lefutási valószinüségének ismeretében.

2.2. Feldolgozási feladat programozása

A feldolgozási rendszer, feldolgozási feladat fenti fogalma különböző szintekre alkalmazható. Programelem lehet egy gépi utasitás /mikroprogram utasitás/, lehet valamely programozási nyelv utasitása, de lehet egy - a definicióban emlitett tulajdonságokat kielégitő eljárás, vagy program is. Programozási nyelvek szintjéig bezárólag az általunk feldolgozási feladat-összeállitásnak nevezett tevé-

kenységeket programozásnak nevezik, a programelemeket utasításkészletnek, a programelemek inputjait és outputjait az utasitás operandusainak, vagy paramétereinek. A szokásos programozási eljárás mindig egy linearizálás, azaz az adott nyelv programelem összekapcsolási lehetőségeit figyelembe véve fel kell irni a megfelelő operendusokkal társitott programelemek sorozatát, amely előbb-utóbb lineáris lesz, legkésőbb az utasitások időben egymásutáni végrehajtásakor. /Itt a multiprocessingtől, mint nem általánosan elterjedt lehetőségtől eltekintünk-ugyanigy, hasonló jellemzői miatt a multitaskingtól is./

A fentiekkel analóg módon az adatfeldolgozási rendszerekben egy feldolgozási feladat összeállitása olyan programozást jelent, amelynek utasitáskészletét a feldolgozási rendszerhez tartozó programelemek képezik /az általános célu utilityket mindig ide értjük/, az operandusok a logikai fileok - mint egyetlen dyan memória, amely a programelemek között permanens információt közvetithet; az utasitások összekapcsolási lehetőségét pedig vagy az operációs rendszer, vagy egy saját kezelő rendszer biztositja.

Megj.: Itt a logikai file fogalma nem szükségképpen azonos egy-egy konkrét operációs rendszerbeli file fogalmával. Logikai filenak tekinthetjük például egy integrált adatbázisnak egy olyan részét /rekordhalmazát/ amelyet egy vagy több programelem egységesen kezel.

A fenti analógiának néhány, a programrendszer hatékonyságával összefüggő vonatkozását emeljük ki a következőkben. Ahogyan hosszu futásidejü eljárásoknál megoldandó a checkpoint és restart problémája, ugyanugy felmerül ezek igénye feldolgozási feladatok esetében is. Nyilván itt elsődleges checkpointként a hálógráf olyan csucspontjait kell számitásba venni, amelyekből a feldolgozási feladat ujrainditható.

A gépi programozás hőskorában a programozóknak optimalizálni kellett programjaikat a lefoglalt memóriarekeszek számát illetve a végrehajtott utasitások idejét tekintve. Adatfeldolgozási feladatoknál ma a felhasznált memóriarekeszek számának csökkentése néhányszáz byte vagy szó ere-

jéig nem befolyásolja lényegesen a program tárigényét, hiszen a fordítás és szerkesztés során több tiz K-nyi rendszer rutin kapcsolódik hozzá.
A memóriaoptimalizáción az adatfeldolgozási rendszerekben a programelemekhez tartozó logikai fileok, pontosabban az adathordozók lekötésének optimalizációja felel meg.
A futásidők tekintetében az adatfeldolgozási rendszerek szintjén nem néhány gépi utasitás megtakaritásával érhetünk el hatékonyabb időkihasználást, /főként több százezres másodpercenkénti müveleti sebességnél/ hanem a programelemekben az ennél néhány nagyságrenddel nagyobb végrehajtási idejü I/O müveletek számának optimalizálása mellett a programelemek végrehajtása közötti időintervallum csökkentésével. Ez utóbbi gyakran azonos nagyságrendü egy-egy programelem teljes végrehajtási idejével.

Az előzőekben beláttuk, hogy a feldolgozási rendszerekben a feldolgozási feladatok a hálószerüen összekapcsolódó programelemek, ennek a strukturának egy bejárása, a linearizálás - programozás.

Egy fenti tulajdonságu hálót az R reláció az A feltétel szerint maximálisan be tudunk járni, ha a következő feltételek teljesülnek:

1./ a feldolgozási feladat gráfjának minden csucsában információnk van legalább arról, hogy a többi csucsban lévő programelemek elindultak-e már, illetve ha elindultak lefutottak-e vagy sem.

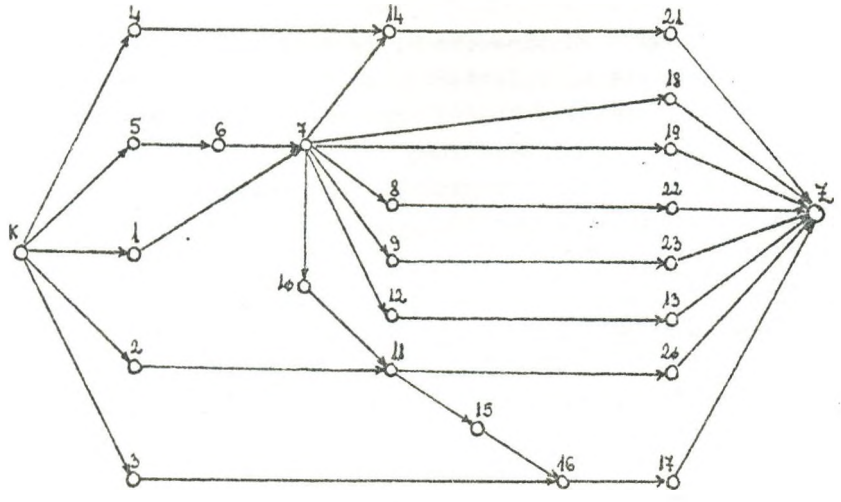
2./ Van egy elágazó utasitástipus a kezelőrendszer szintjén, amelynek predikátumrészében a csucsokban lévő programok lefutására vonatkozó kérdésekből összeállitott konjunktiv kifejezések megengedettek, utasitásrészében pedig a programelemek inditására vonatkozó parancsok.

Megj.: Mélyebb analizissel ennél szükebb elégséges feltételek is adhatók.

Az operációs rendszerek ezen feltételek teljesitéséhez különböző mértékü segitséget adnak. Mi a DOS operációs rendszer - mint a hazai ESZR gépeken egyik legszélesebb körben használt operációs rendszer lehetőségeit vizsgáljuk konkrét programrendszer üzemeltetési megoldásának bemutatásával.

3.1. A VIDIA programrendszer feladata, mérete, hálógráfja

A VIDIA Nagykereskedelmi Vállalat ügyvitelgépesitési és adatfeldolgozási rendszere 1976. január 1. óta müködik üzemszerüen a JATE Kibernetikai Laboratórium R-40 gépén. Tekintettel arra, hogy a rendszer egészének áttekintése meghaladja a rendelkezésünkre álló idő és oldalszám keretet, annak - számunkra - legtanulságosabb részét emeljük ki. Ez a rész a napi rendszer, amely egyetlen ponttól eltekintve /a napi tranzakciók gyüjtése a heti ill. a havi feldolgozáshoz/ önálló rendszernek tekinthető.
A nagykereskedelmi vállalat öt kirendeltségéről /Szeged, Békéscsaba, Baja, Szolnok, Kecskemét/ érkeznek be naponta inputként a különböző árumozgásokat leiró adatszalagok. A tárolt áru és vevő - szállitó törzsadatok felhasználásával el kell készitenl a számla és egyéb bizonylatokat, ezek összesitőjét, a napi forgalmi adatok halmozását a napi jelentés adataihoz, a vevő vállalatok terhére kiállitott inkasszójegyeket. Ezen kivül a tranzakciós file archiválása és a törzsadattárak aktualizálása is napi feladat.
A rendszer méreteit jellemzi, hogy 28000 a különböző árucikkek, 3800 a vevő - szállitó rekordok száma. A napi input tételszám 4500.
A feladat megoldásának az alábbi hálógráf felel meg. Az egyes csucsokban a VIDIA programrendszer programjaiból kiválasztott programelemek állnak.



3.2. A VIDIA programrendszer üzemeltetésével kapcsolatos igények

A rendszer futtatásához a felhasználó kétfajta input adatot szolgáltat. Az árumozgások adatait leiró lyukszalagokat a kirendeltségeken a munkaidő végéig lyukasztják, igy a szállitási idő miatt 21 óráig érkeznek a Laboratóriumba. A törzsadatváltoztatások kártyái 13 óráig elkészülnek. A vállalat másnap reggel 6-kor várja az outputot. Igy a törzsadat módositásokat a napi futtatás során végre kell hajtani, hogy a változott törzsadatu cikkek másnap ellenőrzött adatokkal kerülhessenek a feldolgozásba.
A rendszer üzemeltetéséhez a tényleges futáskor nem áll rendelkezésre futtató személyzet, az operátoroknak egyedül kell biztositaniuk a maximális lefutást.
Az általános gépüzemeltetői igények közül néhánynak esetünkben különleges jelentősége van.

- A gép napközbeni fokozottabb leterheltsége miatt lehetőleg a tervezett időben fusson a rendszer. Ugyanez a szempont más szinten ugy merül fel, hogy ha az opetárorok felkészültek a rendszer futtatására, akkor történjen is meg a maximális lefutás.
- A rendszer futtatásához 3 mágnesszalag- és 2 lemezegységnél ne legyen többre szükség, a programok 60 K-s particióban is futhassanak.

3.3. Az üzemeltetés megoldása

A 2.2. pontban meghatározott 1. feltételt, azt, hogy az egyes programelemek lefutásáról az operátornak legyen információja a végrehajtás minden pontján azzal érjük el, hogy egy-egy rész végrehajtása után

* c. <ell.pont szám> <azonositó>

alaku üzenet jelenik meg konzolon. A következő job step végrehajtása megkezdődik. Az üzenet olyan job stepet azonosit, amelynél hiba esetén a feldolgozást folytatni kell. Ez a jelzés a következő üzenet megjelenéséig érvényes.
A 2. feltétel kielégitéséhez a DOS Job Control-nak azt a tulajdonságát használjuk ki, hogy adat-, program-, hozzárendelési, JCL vagy hardware hiba esetén a job megszakad

és a hátralévő részét a JC átugorja a /& kártyáig, azaz a végrehajtást a hiba előfordulási helyén felfüggeszti. Ezt a funkciót ugy tehetjük hatásossá, ha az egész programrendszer futását vezérlő kártyákat egyetlen jobba tesszük. Az utoljára kijelölt ponttól való ujraindítást az operátornak kell végeznie, mivel a DOS-ban erre más lehetőség nincs. Az érvényes - utolsó üzenetben kapott - azonositót meg kell találnia a jobot vezérlő kártyák egyikén. Innen kell ujrafuttatnia a jobot. A szükséges JOB kártyát illetve a hiba előfordulási helyétől függően esetleg szükséges más vezérlőkártyákat ugyanazon azonositó segitségével találhatja meg az ujrafuttatáshoz leadott kártyacsomagban. Ezekkel az eredeti job maradékát kiegészitve uj, teljes jobot kap, amellyel ujraindithatja a futtatást.
Az ujrainditáskor fellépő időveszteséget csökkenthetjük, ha

- külön jobban elvégezzük az input-output berendezések permanens hozzárendelését és a lefutás után visszaállitjuk;
- felhasználjuk a DOS POWER adta lehetőségeket az összetartozó vezérlőkártyák POWER SLI könyvekbe vitelére. A leadott kártyák száma igy csökken, a keresés egyszerübb lesz és gyorsul.

Az eddigiek során - a változó belső körülményekhez való alkalmazkodással, a felderitett gyenge pontok erősitésével együttjáró programmódositásokkal és a fokozatosan módosuló job összeállitással - az emlitett szoros határidőt 95 %-os biztonsággal sikerült tartani. Az operátorok gyorsan elsajátitották és biztonságosan alkalmazzák az ismétlőpontokról való ujrainditás módszerét. Az üzemeltetői napi tevékenység - néhány esettől eltekintve - két paraméterkártya és a törzsanyagok módositó kártyáinak cseréjéből áll.

Az üzemszerü müködés első időszakában a napi futási idő átlagosan 2 óra volt. Jelenleg - a kb. négyszeresére növekedett input tételszám mellett átlagosan 80 perc a végrehajtási idő. Ezt a gyorsitást és biztonságot az ismertetett futtatási módszer bevezetésével, az egyes program-

elemek adathordozó igényének összehangolásával és a kedvezőbb CPU-I/O arány elérésére a programelemek módositásával értük el.

A fejlettebb operációs rendszerekben, igy az OS-ben is a return kód /user return kód/ beállitási, lekérdezési lehetősége, a feltételtől függő jobstep végrehajtás, valamint az automatikus és feltételes restart funkció a JCL-ben biztositja a rendszerüzemeltetés megkivánt alapelemeit. Az üzemeltetési munkát nagymértékben segitené egy általános, az operációs rendszer és a programelemek közötti vezérlő program, amely a hálógráf, az egyes programelemek erőforrásigénye és lefutási valószinüsége alapján megtervezné és végrehajtaná a kijelölt feladatot. Ilyen eszközre a fejlettebb operációs rendszerekben is szükség lehet a változó körülményekhez való rugalmasabb alkalmazkodás, valamint a nagy figyelmet igénylő szellemi munka gépesitésének érdekében.

Megállapithatjuk, hogy ha a programrendszer tervezése során figyelembe vesszük az első részben tárgyalt követelményeket, illetve megkönnyitjük a programrendszer vezérlését az itt vázolt alapokon, akkor számottevően javithatjuk az elkészülő programrendszer hatékonyságát.

Felhasznált irodalom

1. IBM SYSTEMS REFERENCE LIBRARY
   DOS SYSTEM CONTROL AND SERVICE

2. IBM SYSTEMS REFERENCE LIBRARY
   DOS AND TOS UTILITY PROGRAMS

3. Bölcsföldi József:
   Az OS Job Control nyelve
   Statisztikai Kiadó Vállalat,

4. IBM SYSTEM/360 DOS POWER II.
   PROGRAMLEIRÓ ÉS OPERÁCIÓS KÉZIKÖNYV /FORDITÁS/.

5. VIDIA Programrendszer specifikáció /dokumentáció/

EGY KÓRHÁZI INFORMÁCIÓRENDSZER INPUT-TEVÉKENYSÉGEIT MEGVALÓSITÓ PROGRAMRENDSZER JELLEMZŐI

Pasek Béla - Benedek Szabolcs
Szegedi Orvostudományi Egyetem
Számitástechnikai Központ

A Szegedi Orvostudományi Egyetem Számitástechnikai Központ és az I.sz. Belgyógyászati Klinika közös fejlesztésében a GIN-S-nek nevezett kórházi információrendszer-modell létrehozásán dolgozunk.

Alapvető input perifériaként a VT-340 tipusú displayket jelöltük ki. Ezek müködtetését a rendelkezésre álló R-10 számitógép segitségével interaktiv módon valósitottuk meg. A feladat megoldásához programrendszert készitettünk, amit alaprendszernek nevezünk.

A következőkben a display-ken párbeszédes üzemmódban megvalósitott input-tevékenységek módját, jellemzőit ismertetjük.

A helyi adottságokat és lehetőségeket figyelembe véve /R-10 konfiguráció, más rendszerfejlesztési munka, napi rutin futtatások/az alaprendszer a következő erőforrásokra épült:

- 17 Kbyte központi memória
- 1-4 db VT-340 tipusú display /CLA vonalon/
- 2,5 Mbyte diszkterület
- 1 db mágnesszalagos egység
- 1-4 IT szintet kezelő kártya.

Az alaprendszer müködés közben feltételez egy RTDMFE futtató-monitor jelenlétét.
A GIN-S alaprendszerének futásakor a memória felosztását az 1.sz. ábra szemlélteti.

A MEMÓRIA FELOSZTÁSA

	FELOSZTÁS	NAGYSÁG
TELJES MEMÓRIA		32 KSZÓ
	MONITOR TERÜLET	13 KSZÓ
GIN-SYSTEM	FOREGROUND TERÜLET	9 KSZÓ
	BACKGROUND TERÜLET	10 KSZÓ

A GIN-SYSTEM HELYFOGLALÁSA(4 DB DISPLAY): 8,5 KSZÓ

1. ábra

Az adatközlő személy a GIN-S-ben az adatokat utasitásnak nevezett dialógusokban tudja közölni a számitógéppel a display-k segitségével. Az összes olyan személyt, akinek a GIN-S-ben adatközlési és visszakérdezési joga van, kezelőnek nevezzük.

Az utasitás végrehajtásán a kezelő és a számitógép közötti adatkommunikáció /dialógus/ folymatát értjük. A párbeszéd lebonyolitása közben - az utasitás végrehajtása alatt - megjelenő /vetitett/ display-képernyő tartalmakat képeknek nevezzük.
Azokban az utasitásokban, amelyekben betegre vonatkozó információk kerülnek feldolgozásra, a beteg kijelölését a munkaszám biztositja. A betegre a kezelők a munkaszámhoz hasonlóan a kórterem-ágyszám /5 jegyü decimális szám, első három jegye a kórtermet, utolsó két jegye a kórtermen belüli ágyszámot jelöli/ megadásával is hivatkozhatnak.
A kezelők által a display-n megadott

$$\&XX\ YYY\left[\sqcup\left\{\begin{matrix}ZZZ\\ \\ WWWWW\end{matrix}\right\}\right]\ ETX$$

alakú karakter-sorozatot utasitáshivásnak nevezzük, ahol:

- & jelzi, hogy utasitás hivása következik
- XX karakterek helyén megadott betük a kezelő titkos kódja
- YYY karakterek helyén megadott három betü az utasitás neve
- ZZZ és WWWWW karakterek helyére a munkaszám, illetve kórterem-ágyszám kerülhet.

A munkaszámot és a kórterem-ágyszámot az utasitás paraméterének nevezzük.

Az utasitás hivásának szabályából látszik, hogy a GIN-S-ben paraméteres és paraméter nélküli utasitások is lehetnek.

A display-eknek azt az állapotát, amikor utasitás hivása következhet alapállapotnak nevezzük.

Azt a karaktersorozatot, amelyet a kezelő az utasitás végrehajtása alatt egy kép megjelenése után az ETX gomb lenyomásáig ad meg, válasznak nevezzük.

Az alaprendszer jellemzői, müködéséről szerzett tapasztalatok és a továbbfejlesztés lehetőségei:

A GIN-S fejlesztése alrendszerek egymás utáni létrehozásával történik. Eddig három alrendszer, kb. 25 utasitását - összesen 650 kép - készitettük el. Ezek az utasitások idáig napi rutin feladatokat még nem látnak el, de tesztelésük során az alaprendszert több mint 200 órán keresztül /2-3 display egyidejü felhasználásával/ müködtettük.

Programrendszerünk az adatátviteli és kezelői hibákat észleli, és mindig vissza tud térni a hibát megelőző állapotba vagy az alapállapotba. Ez teszi lehetővé rendszerünk biztonságos müködését.

Az utasitás végrehajtása alatt egy képre adott válasz feldolgozásához és a következő kép megjelenitéséhez átlagosan 1500 gépi utasitás végrehajtása és 5 diszkhez fordulás szükséges.

Az alaprendszer jelenleg 4 display kezelésére képes. Ujabb display üzembeállitása egyetlen modul átirását /bővitését/ és 1 Kbyte központi memóriaigény növekedést jelent.

Az előzőekben ismertett erőforrásokkal rendelkező R-10 konfigurációra az alaprendszer módositás nélkül átvihető. Olyan diszkre való áttérés esetén, ahol az FMS-10 filekezelő nem használható a diszk-kezelő modul átirása után az alaprendszer üzemképes.

Az aszinkron vonalon történő átvitelt megvalósitó modul lecserélésével programrendszerünk más display-s rendszerekre is alkalmazható.

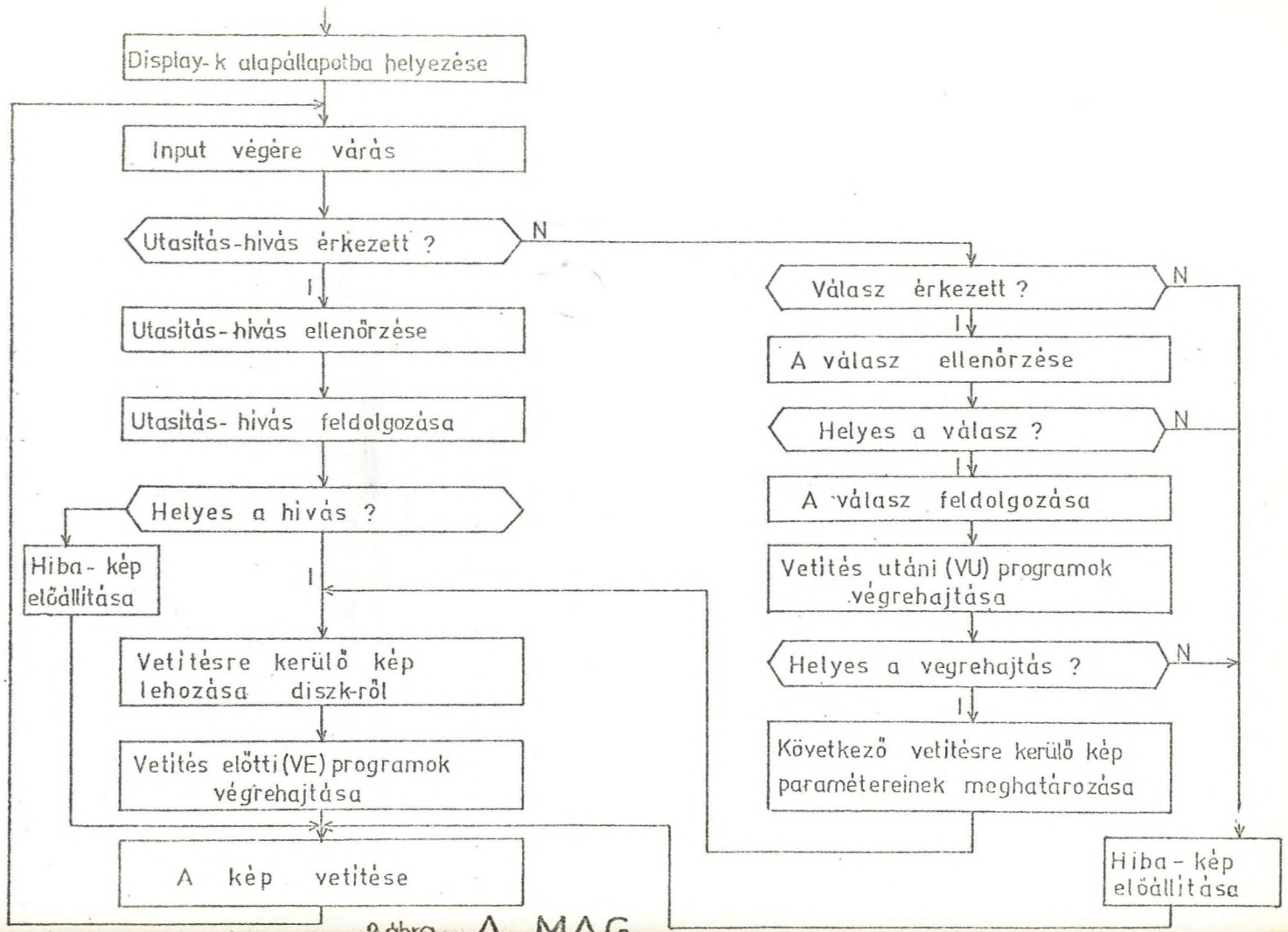
A rendszerben egy új utasitás felépitésének és képeinek meghatározásával mindig definiált az adatáramlás folyamata is, amely elemi tevékenységek sorozatára bontható. Ezen tevékenységeket valósitják meg a vetités előtt /VE/

és a vetités után /VU/ végrehajtásra kerülő programok. Feladatuk általában egy-egy utasitáscsoporthoz tartozik - nem általánosak - igy beépitésük a rendszer állandó részébe /MAG/ nem látszott célszerünek.
Az alaprendszer moduláris felépitése biztositja, hogy a rendszer állandó részéhez a VE és a VU programokból az igényeknek megfelelő /az utasitások által meghatározott/ minimális programrendszer újraszerkesztéssel előállitható legyen.

A MAG müködési elvét a 2.sz. ábra szemlélteti.
A rendszer inditásakor a következő paramétereket kéri:
- napi dátum
- aktiv display kijelölése.
Ezen input adatok feldolgozása után inditja a rendszert az aktivnak jelölt display-ken, a display-ket alapállapotba helyezi, azaz a rendszer utasitás hivását várja. Az utasitás hivását követően /az ETX karakter leütése után/ a rendszer ellenőrzi, hogy utasitáshivás történt-e.
Amennyiben nem, a rendszer újra alapállapotba kerül.

Utasitáshiváskor ellenőrzi a hivás helyességét, és amennyiben helyes volt, megkezdődhet az utasitás végrehajtása, azaz az első kép vetitése.
A megjelenitett képre adott helyes válasz esetén - a kép jellege és az előző válaszok által meghatározott - a következő kép kiirása és a bejegyzett válasz tárolása a feladat. Az utasitáshoz tartozó utolsó kép vetitése után a rendszer ismét alapállapotba kerül.
A rendszer jelenleg 4 display szimultán müködését biztositja, kitüntetett display nincs. A CLA vonalon on-line üzemmódban minden display-n leütött karakter, függetlenül a végrehajtástól, átvitelre kerül.

Megoldottuk bizonyos, csak ténylegesen végrehajtásra szánt karakterek szürését. A rendszer lehetőséget nyújt on-line üzemmódban a display-kép javitására és a hivó modul számára megadja a display-n közölt érvényes karakterek számát.


Display-k alapállapotba helyezése
Input végére várás
Utasítás-hívás érkezett ?
N
I
Utasítás-hívás ellenőrzése
Utasítás-hívás feldolgozása
Helyes a hívás ?
Hiba-kép előállítása
I
Vetítésre kerülő kép lehozása diszk-ről
Vetítés előtti (VE) programok végrehajtása
A kép vetítése
Válasz érkezett ?
N
I
A válasz ellenőrzése
Helyes a válasz ?
N
I
A válasz feldolgozása
Vetítés utáni (VU) programok végrehajtása
Helyes a végrehajtás ?
N
I
Következő vetítésre kerülő kép paramétereinek meghatározása
Hiba-kép előállítása

2.ábra A MAG

A kép vetitésére algoritmust készitettünk, mivel CLA vonalon nem adható egyszerre egy egész képernyő adattartalma outputra.
A display-n megadott utasitáshivást ellenőrizzük, hogy szintaktikusan és szemantikusan helyesek-e /3.sz. ábra/.

Egy külön program-modul végzi a vetített képre adott válasz szintaktikus és szemantikus ellenőrzését, és a válaszok konvertálását belső ábrázolási kódba. A kód kialakitásakor a tömörséget tartottuk szem előtt, mivel a GIN-S az input adatok döntő többségét ily módon tárolja.

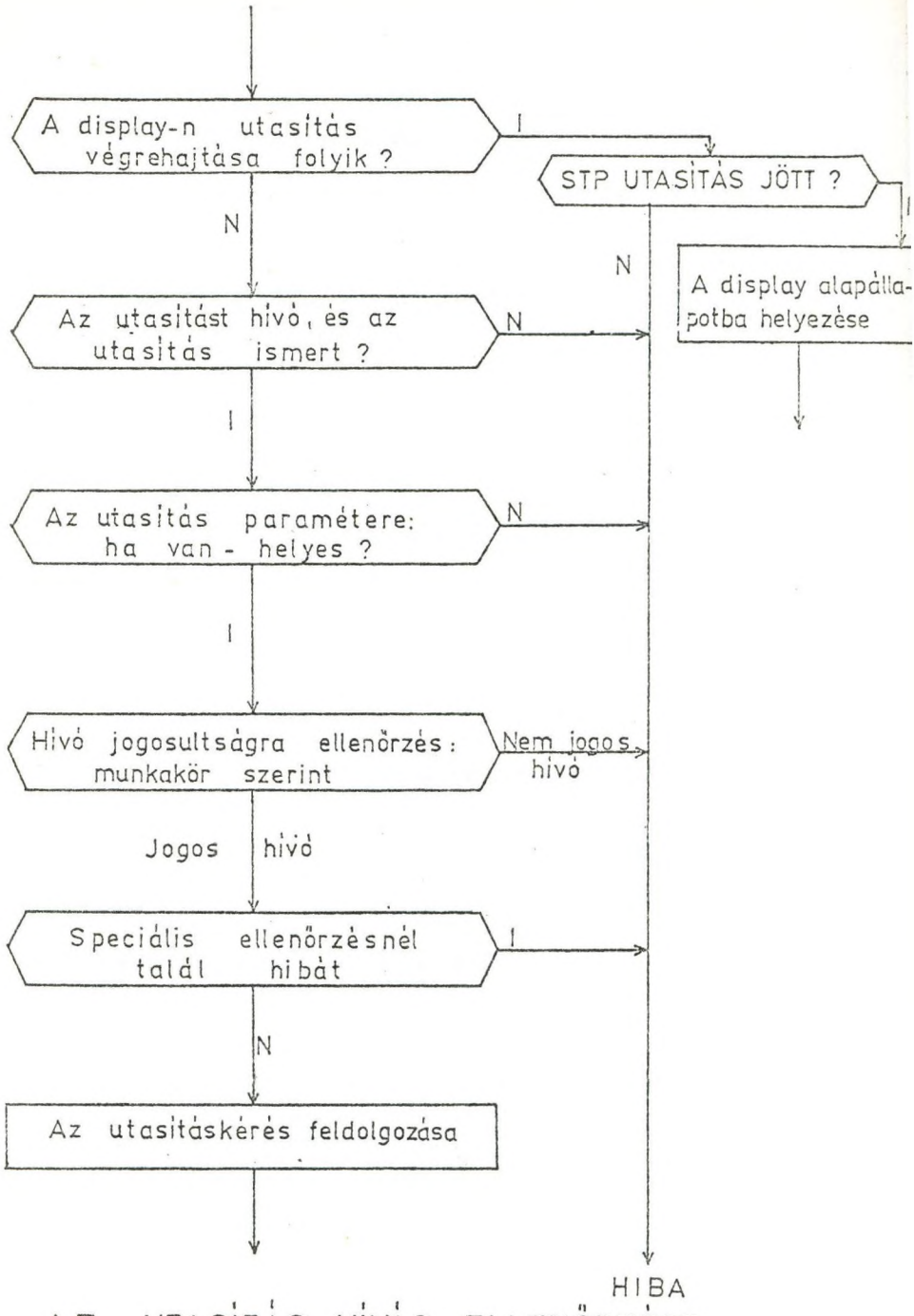
A rendszer biztositja - ahol erre szükség van - a képre adott válasz összegyüjtését belső tárban és rendszer file-okban.
A diszkes file-okhoz fordulást FMS-10 filekezelő segitségével valósitjuk meg.

A fejlesztés és bővités könnyitése érdekében olyan modulokat épitettünk a rendszerbe, amelyek megoldják bármilyen jellegü hiba egységes kijelzését, lehetőséget biztositanak a rendszer futás közben történő tesztelésére, és a rendszer követésére.

A programrendszer kevés programozási munkától / a meglévő VE és VU programokkal nem megvalósitható tevékenységek megoldásától/ eltekintve teljes.

A további fejlesztésben a felhasználók által kért /előzetes felmérés alapján rögzitett/ utasitások kialakitása a feladat.

A közlemény anyaga az Eü.M. 4-13-0201-03-0/GY számú "Számitástechnikai módszerek, rendszerek, berendezések fejlesztése, adaptálása az orvostudományban és az egészségügyben" c. tárcaszintü kutatási főirányhoz minisztériumi szinten kiemelten elfogadott "Számitástechnikai és matematikai módszerek alkalmazása az orvostudományban és az egészségügyben" c. témában végzett kutatómunka alapján készült.



AZ UTASÍTÁS- HÍVÁS ELLENŐRZÉSE

3. ábra

DOKTOR - EGY PROGRAMHIBA DIAGNOSZTIZÁLÓ RENDSZER

Pomper János
Csepel Müvek Fémmüve, Szervezési Osztály

1. Bevezetés

Egy 1975 -ben számitóközpontokban elvégzett felmérés szerint a programozók munkaidejüknek közel 50 % -át a programokban felmerülő hibák okainak megkeresésére forditják. A tapasztalat szerint ennek az időnek viszonylag kis százaléka forditódik a programkészités során elkövetett szintaktikai hibák kijavitására, a legtöbb időt és energiát a programok próbafuttatása során vagy a futás közben előforduló hibák okainak megkeresése és kijavitása igényli. Különösen igy van ez akkor, ha a számitógép ezekről a hibákról a program készitési módjától - a forrásnyelvtől - teljesen idegen módon, sőt a forrásnyelvü programozó előtt gyakran érthetetlen rendszerben, pl. hibakód és abszolut cim vagy dump formájában közöl információt. A hiba megkeresésének legfáradságosabb és leghosszabb ideig tartó része arra irányul, hogy a hibának a számitógép által közölt formájából megkeressük a forrásnyelvü programnak azt a részét, esetleg utasitását, amelyik felelős lehet a futási hibáért. E "visszaforditási" munka megkönnyitése érdekében a compilerek biztositanak bizonyos hibakeresési lehetőségeket.

A CSM-ben jelenleg üzemelő adatfeldolgozási rendszerek programjainak tulnyomó többsége COBOL nyelven készül vagy készült. Ugyanakkor viszont a számitógéphez / ICL

SYSTEM 4-52 / tartozó compilerek közül éppen a COBOL az, amely a legkevésbé van ellátva hibakeresési eljárásokkal. A rendelkezésre álló eljárások /ON, TRACE, EXHIBIT, DEBUG/ pedig csak a forrásnyelvü programok módositása árán használhatók, de ezek kevés, vagy nem a célra irányuló, vagy éppen túlzott bősége miatt használhatatlan információt nyújtanak.

A fenti problémák érzékelése vetette fel azt a gondolatot, hogy szükség lenne egy COBOL nyelvre orientált hibakereső programra, amely képes a programozónak a hibakeresés ismétlődő és fáradságos "visszaforditási" munkáját megkönnyiteni, a hiba okát és helyét a forrásnyelvü programozó számára érthetőbb formában közölni.

Ezt a feladatot végzi el a DOKTOR, a Dumpolási Okok Keresésnek Tipizált On-line Rendszere. Használalata rendkivül egyszerü, a forrásnyelvü program módositása nélkül beépithető. Csak akkor lép müködésbe, ha a program futása közben hiba fordul elő: - a hibáról információt ad, azután /ha lehet/ folytatódik a felhasználói program futása,
- egyébként észrevétlen marad.
Egy hiba okának és helyének megkereséséhez egy másodpercnél rövidebb időre van szükség. Ha figyelembe vesszük, hogy azonos mennyiségü és értékü információ előkeresése egy jól képzett programozónak legalább negyedóráig tart, feltéve, hogy minden szükséges kiinduló információ a bir-

tokában van, akkor a használatának gazdaságossága pusztán a hibakeresésre forditott idők pénzegyenértékéből is nyilvánvaló, nem beszélve arról, hogy mennyi gépidőt, papirt és fárasztó programozói munkát takarit meg.

2. A hibakeresés stratégiája

A felhasználói programok készitésének folyamatára is érvényes az a számitástechnikában általánosan elfogadott alapelv, hogy az esetleges hibákat a folyamat legkorábbi szakaszában ki kell szürni. Ez a törekvés nyilvánul meg abban a tendenciában is, hogy a modern compilerek a szintaxis vizsgálata mellett egyre több szemantikus hibára is rámutatnak. Ennek az irányzatnak azonban határt szab az a tény, hogy a compilerek a program jóságát csak önmagában vizsgálják, de nem képesek előre jelezni a program és az általa feldolgozott adatok kölcsönhatása folytán fellépő struktúrák hibáit.

A programok készitésének folyamata két szakaszra bontható: Az első szakasz a programok helyes szintaktikai formájának kialakitása, melyben a compiler segit a programozónak a hibák kijavitásában; a második szakasz a programok próbaadatokkal történő "belövése", melyben az operációs rendszer ad jelzést a hibákról.

Mai számitógépeinknél a hibákról adott jelzések minő-

sége és formája azonban a programkészités két szakaszában érdekes ellentétet mutat.

Az első szakaszban a szintaktikus hibákról a compilerek a forrásnyelvü programozó által könnyen érthető formában közölnek információt, minőségileg pedig a hiba helyét és tipusát pontosan megadják - ezért a javitást egyszerüvé teszik. A második szakaszban viszont a végrehajtási hibákról az operációs rendszer által közölt adatok szinte mindig kódolt formájúak, minőségileg pedig a hiba tipusát általános jelleggel, helyét pedig legfeljebb hexadecimális cimként adják meg - ezért a forrásnyelvü programozó számára a javitást alig könnyitik meg.

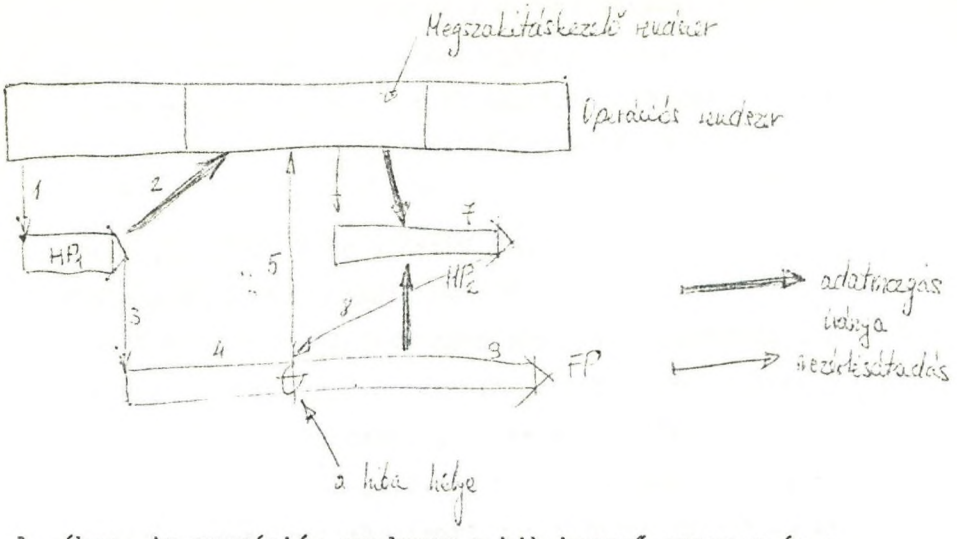
Ez a kettősség vetette fel azt a gondolatot, hogy a végrehajtási hibákról közölt információk értékét sokszorosára növelné - és ezáltal a hibakeresésre forditott erőfeszitést töredékére csökkentené - egy olyan program, amely az operációs rendszer által adott hibaüzenetet a forrásnyelvhez közelálló formaban adja meg. Ez az igény azt jelenti, hogy mig a programkészités első szakaszában létrejövő forrásnyelv → gépi kód transzformáció mutatja ki a szintaktikus hibákat, addig a második szakaszban egy ellenkező irányú kód → forrásnyelvü hibaüzenet transzformációra van szükségünk, amely a végrehajtási hibát mutatja ki.

Ezt az ellenkező irányú transzformációt végző hibakereső programnak szüksége van arra, hogy hozzáférjen a hibás állapotú kódhoz,és az operációs rendszer által meghatározott kódolt hibaüzenethez.

Ennek biztositására kézenfekvő a gondolat, hogy a hibakereső programot /a továbbiakban HP/ a felhasználói programmal /továbbiakban FP/ együtt töltsük a memóriába, és az FP futási hibája esetén a HP-t a gép megszakitáskezelő rendszerébe épitett hivás útján aktivizáljuk. Az FP-hoz a HP-t úgy kell hozzáépiteni, hogy ez az FP-ban semmiféle változtatást ne tegyen szükségessé. Ez legegyszerübben olyan módon valósitható meg, hogy a LINKAGE EDITOR köti össze a HP-t és az FP-t úgy, hogy a HP a főprogram, a FP pedig ennek szubrutinja legyen. Ilyen szerkezet mellett a kapcsolattartáshoz szükséges minden eljárást a HP-ba épithetünk be, és igy az FP-ben semmiféle változtatásra nincs szükség. Az igy egybeépitett FP-HP kapcsolat az 1.sz. ábrán látható.

3. A hiba helyének meghatározása

A fenti alapelvekre épülő HP müködése az 1.sz. ábrán látható módon két fázisra bomlik. Az első fázisban kezdőértékeket ad és előkésziti az operációs rendszer megszakitáskezelő részét arra, hogy végrehajtási hiba



1. ábra. Az operációs rendszer,a hibakereső program és a felhasználói program kapcsolata

1 a futás kezdete

2 a hibakereső program előkésziti a megszakitáskezelő rendszert

3 a FP futásának kezdete

4 az FP futása

5 a hiba fellépésekor a vezérlést átveszi az operációs rendszer

6 a megszakitáskezelő rendszer átadja a vezérlést a HP második szakaszának

7 a HP az FP és a hibakódok alapján elvégzi a hibakeresést

8 ha a hiba tipusa megengedi, a FP folytathatja a feldolgozást, ha nem, akkor a HP terminál.

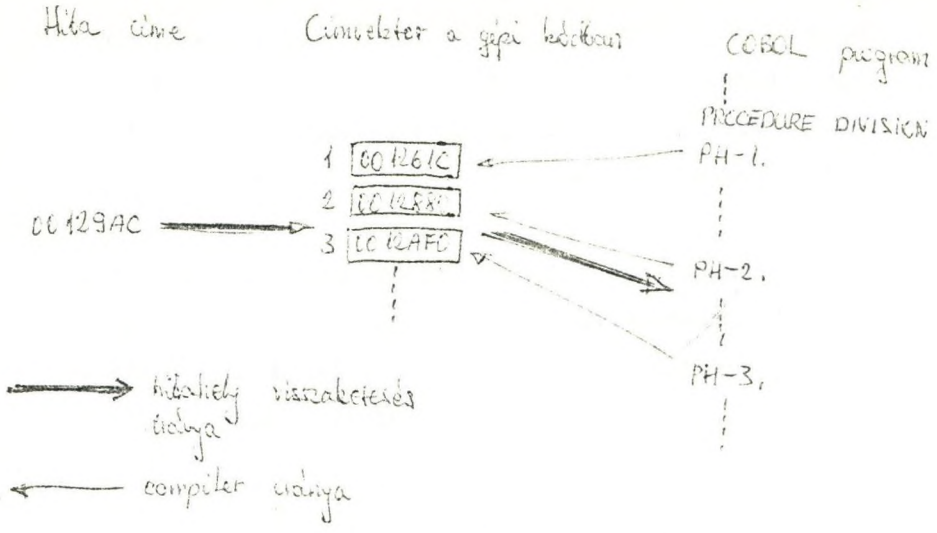
9 a felhasználói program folytatja a futást

esetén aktivizálja a második fázist. A második fázisban a HP az operációs rendszertől átveszi a kódolt formájú hibaüzenetet és a hiba helyét, majd a hibás állapotú FP módszeres analizisével ezt forrásnyelvü formára alakitja át.

A hibakeresési eljárás legizgalmasabb része a hiba helyének forrásnyelvü szintü megadása. Erre az a tény nyújt lehetőséget, hogy a COBOL compiler által készitett gépi kód nagyon pontosan meghatározott szabályok szerint épül fel.

A hiba helyének megadására az egyik jó lehetőség a hibás forrásnyelvü utasitást tartalmazó paragrafus sorszámának meghatározása. A leirás sorrendjében egymást követő paragrafusok kezdőpontjainak cimét ugyanis a gépi kód egy vektorban tárolja, igy a feladat csak az, hogy ezen a cimvektoron végighaladva meghatározzuk annak a legnagyobb paragrafuscimnek a sorszámát, amely a hiba ciménél kisebb. Ez a sorszám máris megadja a hibás paragrafus számát /2. ábra/.

Ezt a közelitő jellegü információt tovább finomithatjuk, ha megvizsgáljuk azokat a szabályszerüségeket, amelyeket követ a forditóprogram a forrásnyelvü utasitás gépi kódra történő átalakitásakor. Ezek a szabályszerüségek egy-egy kódminta segitségével irhatók le.



2. ábra. A hibás paragrafus számának meghatározása

A kódmintáknak olyan változó hosszuságu bytesorozatokat tekintünk, amelyekben meghatározott helyeken fix értékek vannak, más helyeken esetenként változó értékek szerepelhetnek. A compiler ugyanis meghatározott szabályok szerint ilyen kódmintát választ ki saját mintakészletéből akkor, amikor egy forrásnyelvü utasitást gépi kódra alakit át. Ezt a mintát az adatneveknek megfelelő cimekkel és hosszakkal feltölti és igy illeszti be a gépi kódsorozatba.

A "visszaforditáshoz" tehát a gépi kódot zavaró adatokkal /cimekkel és hosszakkal/ feltöltött kódmintának kell tekintenünk, és a feladat egyszerüen az, hogy az aktuális gépi kódból kiszürjük a zavaró adatokat, és az igy adódó mintát összehasonlitsuk a forrásnyelvü utasitásokat reprezentáló mintakészlet elemeivel. Ha a "zavarszürés" minősége egy kritériumrendszer szerint megfelelő /pl. az

aktuális kód teljesen megegyezik valamelyik kódmintával/, akkor a kódmintához tartozó forrásnyelvü utasitást meghatároztuk.

A fenti eljárást legjobban egy példa világitja meg. Legyen pl. az aktuális gépi kód hexadecimális alakja:

```
..... F2 23 2380 401C                    PACK
      F9 21 2380 23A3                    CP
      47 70 D01C .....                   BC
```

és az IF utasitás egyik kódmintája a készletben

```
      F2 00 2000 0000
      F9 00 2000 2000
      47 00 D000
```

ahol a "0" a zavaró adatok helyét jelzi.

A mintafelismerés abból áll, hogy az aktuális gépi kódból a kódminta szerint zavaró adatokat tartalmazó részeket első közelitésben teljesen figyelmen kivül hagyjuk, és igy a kódmintával összehasonlitjuk. Ha teljes egyezés áll fenn, akkor az aktuális kód egy szakasza megegyezik a kódmintával, azaz megtaláltuk a kódmintához tartozó forrásnyelvü utasitást. Ezt egy stack-ben elhelyezzük, és a gépi kód következő szakaszának vizsgálatára térünk rá. Ha szóbajövő minták közül egyik sem egyezik meg az aktualis gépi kóddal, akkor az egyezés mértékét egy kritériumrendszer szerint pontszámmal minősitjük. Minél nagyobb fokú az egyezés, annál több pontot adunk a kódmintának /pl. teljes egyezés = 1000 pont/, és közülök azt választjuk ki, amelyik a legtöbb pontot kapta.

Ez az eljárás jól alkalmazható, de kevéssé hatékony. Egyrészt azért, mert a szokásos COBOL utasitástipusok mindegyikéhez 3-6 különböző kódminta tartozik, igy a teljes mintakészlettel való összehasonlitás és értékelés hosszabb időt vesz igénybe, másrészt azért, mert az összes kódminta viszonylag nagy memóriaterületet vesz igénybe, és ez, éppen a legnagyobb méretü felhasználói programoknál, korlátozza a gépi hibakeresés lehetőségét.

Az első nehézség a szintaxis-vezérelt compilerek elvéhez nagyon hasonlóan felépülő gépikód-vezérelt mintafelismerő eljárással rendkivül hatékonyan leküzdhető. A kódminták számának csökkentése pedig azáltal válik lehetségessé, hogy bennük nagyon sokszor ismétlődő elemek vannak: egy jól választott jelrendszerrel le kell irni a mintákat alkotó elemeket, és egy kis mintagenerátor a leirás alapján összeállitja az éppen kivánt mintát.

Miután a hiba helyét és okát forrásnyelvi szinten meghatároztuk, az eredményt a felhasználói program nyomtatási képének megzavarása nélkül külön lapra nyomtathatjuk. Ezután a hiba tipusától függően a HP vagy megszakitja a feldolgozást és terminál, vagy visszatér a felhasználói programba, és a hibás utasitást követő utasitásnál továbbfolytatja a feldolgozást.

4. Implementáció és a tapasztalatok

A fenti elvekre épülő hibakereső programot az ESZR gépekhez nagyon hasonló ICL SYSTEM-4 számitógép COBOL nyelven készülő programjaihoz fejlesztettük ki. A DOKTOR egy év óta próbaüzemszerüen használatban van. A vele szerzett tapasztalatok rendkivül kedvezőek. Egyszerű hibakeresésre és az eredmény nyomtatására forditott idő 1 másodpercnél kevesebb, a program memóriaigénye 8 K byte.

A DOKTOR az azonos utasitáskészlettel dolgozó más tipusú számitógépekre könnyen implementálható. Az implementációs munka súlyponti része az új compiler mintakészletének felvétele. Ezzel a mintakészlettel kell a jelenlegi mintakészletet felváltani és a hozzátartozó mintagenerátort elkésziteni.

A mellékletben bemutatott minta-hibaüzenet a felhasználói programozó számára könnyen érthető. Bár összehasonlitó vizsgálatokat nem végeztünk a DOKTOR használatával elért programozói munkamegtakaritásról, tapasztalatunk szerint a programozók könnyen és gyorsan megszokták használatát, a hibakeresésre forditott idő pedig az esetek 80 %-ában a korábbi töredékére csökkent le.

```
********************************************** DOKTOR **********************************************

   ERROR NO: 1                          CLIOoPER403P,00001                          77/06/28  15:58:32

   ERROR :        TYPE : DATA ERROR
                  CLASS: RECOVERABLE
   A  LOCATION :  ABSOLUT ADDRESS:   0384CE     ILC+CC+MASK :  E0
                  POSITION :  035880 + 001E90 + 000098 + 06   (LOAD ADR + CSECT ADR + RELATIV ADR + LENGTH)  IN COBOL MAIN
                  PARAGR. NUMBER IN ERROR :    9
                  SUPPOSED SOURCE STATEMENT SEQUENCE :  PARAGR  INDEXS  *** IF     ***
                  ASSEMBLER STATEMENT IN ERROR :  F9 11 2380 2390  (CP  )
                  DATA-1 :   003F              FROM LITERALS
                  DATA-2 :   0000              FROM LITERALS
```

JOB ACCOUNTING /"JA"/ RENDSZER HELYE A FÜTI ÜZEMELTETÉSI KÖRNYEZETÉBEN

Rammacher Tamás - Urvölgyi Tamás
Fővárosi Épitőipari Üzemgazdasági
és Ügyviteltechnikai Iroda

Az ismertetésre kerülő rendszer a Fővárosi Épitőipari Üzemgazdasági és Ügyviteltechnikai Iroda R-40-es számitógépén - melynek legfontosabb konfigurációs ismérvei a következők: 512 Kbyte operativ tárkapacitás, 1 multiplexor és 3 szelektor csatorna, 2 kártyaolvasó, 2 sornyomtató, 7,25 Mbyte-os lemezcsomagokkal müködő 12 mágneslemezegység, 8 mágnesszalagegység - 1977. augusztusától üzemszerüen müködik.

Néhány szót arról, hogy az 1977. évi KSH rendeleten túl, amely előirja a JA rendszer alkalmazását, miért láttuk célszerünek az elszámolási információk számitógéppel történő feldolgozásának bevezetését. Vállalatunk számitógépes bérirodaként müködik, igy a gépidővel való hatékony gazdálkodás szinte létkérdésnek tekinthető. A DOS/POWER rendszer, mellyel számitógépünket üzemeltetjük, a multiprogramozási környezet, az egyes feladatok, programok paramétereinek és fontos hatékonysági mutatóinak manuális úton történő meghatározását már szinte lehetetlenné teszik. Mivel a rendszergenerálási procedura során - csak egy paraméter megoldásával - lehetőség van arra, hogy igényeljük a DOS-tól az elszámolási információk készitését és átadását - tehát jelentős adminisztrativ munkától szabadulunk meg -, éltünk a lehetőséggel, és igy "csak" az információk feldolgozása maradt a mi feladatunk. Tekintettel arra, hogy a DOS a JOB utasitásban levő jobnév, ill. ha igény van rá, az ugyanebben az utasitásban lévő 16 byte-os felhasználói információk rovat alapján készíti információit, rendkivül sok múlott azon, hogyan sikerül megtalálni a legmegfelelőbb kitöltési rendszert, és miként lehet megteremteni a rendszer bevezetéséhez nélkülözhetetlen, rendkivül fegylmezett üzemeltetési környezetet.

Ez utóbbi, egyáltalán nem problémamentes időszakot érdemes alaposabban szemügyre venni.
Az elszámolási információknak kétszinten történő felhasználását irányoztuk elő a rendszer tervezésekor, az egyik az irányitás szintü, a másik a gazdasági szinten történő alkalmazás.

A sürgősebb megvalósitást igénylő alkalmazás az irányitás szintü volt, hiszen a már emlitett üzemeltetési módszerek alkalmazásának következtében igen kevés információ volt a számitógép hatékonysági mutatóiról, a gépidőfelhasználások tipusainak időszükségletéről és egy sor egyéb fontos paraméterről, ugyanakkor biztositani kellett a vezetők számára nélkülözhetetlen visszacsatolásokat. Ugy döntöttünk, hogy az elszámolási információk készitéséhez csak a jobnév 8 pozicióját használjuk fel, és az un. felhasználói 16 byte-ot tartalékoljuk.
A következő lépés a jobnév kitöltés szabályainak definiálása volt, melyhez rögziteni kellett a számitógépes feldolgozások és a gépidőfelhasználások valamennyi tipusát, a vállalatunkat munkával ellátó ügyfeleink pontos nevét és kódszámát, a dolgozószámokat - természetesen azokét, akik egyáltalán a géppel futtatási szinten kapcsolatba kerülhetnek - , fel kellett mérni a vezetőknek a készülő rendszerrel szemben támasztott igényeit, és természetesen mindezeket összevetni a rendelkezésünkre álló lehetőségekkel. Mindent figyelembe véve a jobnév első pozicióját elegendőnek bizonyult az egyes gépidő felhasználások differenciálására. Igy megkülönböztetünk éles futást /üzemszerü feldolgozás/, software jellegü tevékenységet, programozói tesztet, próbafeldolgozásokat, rezsi munkálatokat /amelyek az egyes futásokhoz az üzemeltetés folyamatosságának biztositásához nélkülözhetetlenek/, valamint a külső felhasználóknak a számitóközpontunk operációs rendszerével végzett futtatásait. A jobnév további pozicióit éles futás esetén a vállalati kód, tárgyidő, feldolgozási tipus, más müveleteknél a vállalati kód és a dolgozószám foglalja el.
Meg kellett találni továbbá a megoldást arra, hogyan lehet a géphibákat - melyeket az operációs rendszer nem képes lekezelni - a rendszerfüggetlen hardware-karbantartásokat és teszteket, valamint külső felhasználónak a saját operációs rendszerével

végzett futtatásait a JA rendszer tudomására hozni. A megvalósitás érdekében bevezettünk néhány különleges jobot, melyeket az emlitett tevékenységek megkezdése előtt illetőleg befejezése után vagy géphiba esetén csak az IPL-eljárást követően kell futtatni. Igy a JA rendszer a kezdési és befejezési időpontok különbségéből értesülhet a kérdéses időintervallumakról.

A megvalósitott rendszer által készitett táblázatok a következők:

- "jobonkénti lista" amely változtatások és számitások nélkül készül a mentett elszámolási információkról,
- "napi gépidőfelhasználások részletezése" cimü, amely időtérképnek tekinthető, mivel a tárgynap/ok/ valamennyi, a számitógépen lezajlott eseményét, óra, perc, másodperc dimenziókban összegezve tartalmazza,
- "munkaszámos lista", amely a megrendelőinknek végzett üzemszerü tevékenységeinkről tájékoztat - és végül
- "tesztidőfelhasználások részletezése" cimü, ahol egy bizonyos - paraméterrel megadható - időszak felhasznált tesztgépidejének kiértékelése található meg munkaszám és programozó/k/ bontásban.

A felsorolt problémákon túl azonban mégis a fegyelmezett és körültekintő üzemeltetési környezet megteremtése okozta a legnagyobb nehézségeket. A legkülönbözőtt ösztönző módszerekkel - igazgatói utasitás, prémiumrendszer - próbálták rábirni az üzemeltetéssel közvetlenül foglalkozókat a JA rendszer követelményeinek megfelelő munkamódszerekre, mégis elég hosszú időszak telt el ennek megvalósitásáig.

Ma már azonban elmondhatjuk, hogy szinte valamennyi érdekelt munkatársunk hozzászokott, és fegyelmezetten viszonyul a JA rendszerhez. A segitségével nyert információk igaznak és megbizhatónak bizonyulnak.

Ahhoz, hogy job-elszámolási rendszerünk igazi funkcióját betölthesse, nevezetesen a gépi számlázás alapjainak tekinthető információkat szolgáltassa, komoly feladatok álltak előttünk.

Megoldást kellett találnunk azoknak a problémáknak az áthidalására, amelyek a DOS által szolgáltatott elszámolási adatok nagyon általános, a multikörnyezetes futások időadatainak "nagyvonalú" kezeléséből adódtak. A futási idők, nagyságrendileg legjelentősebb részét az input/output idők adják. A prioritási alapon szervezett multiprogramozási rendszerben egyáltalán nem mindegy az, hogy egy feladat - mi esetünkben valamely felhasználónk feldolgozása - melyik particióban kerül végrehajtásra. A DOS Supervisor tekintet nélkül az egyes particiók tényleges input/output lekötöttségére, egyenlő arányban felosztja az összes input-output időket az aktiv particiók számával.

Ebből következik az az eredmény, hogy egy alacsonyabb prioritású particióban futó feladat idő adatait is megterheli a rendszer, más magasabb prioritású particiók sokszor lényegesen nagyobb input-output időszükségletének részarányával.

Feladatunk az volt, hogy a JA rendszer által korrektül szolgáltatott CPU időt csak azokkal az input-output időkkel növeljük meg, amelyek mono üzemmódban is jelentkeztek volna. Természetesen a CPU idő és a tényleges I/O időkön túl egy feladat végrehajtási idejében még helyet kell, hogy kapjon az operációs rendszer vezérlő funkcióinak időszükséglete. Ezek az utóbbiak tapasztalataink szerint nagyságrendben elenyészőek az I/O időkhöz viszonyitva, igy ezeknél megelégehetünk a Supervisor által szolgáltatott adatokkal.
Egy adott feladat I/O idejének megbecsülésénél, tiszta mono üzemmódot feltételeztünk - multitaskingtól eltekintve - egy aktiv particióval. Feltételrendszerünkben az I/O idő kizárólagosan egy fizikai blokk átvitelezése, egy primitiv csatornaprogram végrehajtása. Ebben a gondolatmenetben egy program I/O ideje; az egyes perifériákra kiadott SIO utasitások száma szorozva egy fizikai blokk átviteli idejével. Természetesen ez utóbbi tényező két faktornak a függvénye: a periféria illetve csatorna tipusának másrészt a blokkméretnek. A perifériák és a cstornák hardware konstansok. A megoldandó feladat az adott program által az egyes perifériákra/ról/ átvitelezett átlagos blokkméret visszanyerése a job végrehajtása után. Mivel ezeket a blokkméret-információkat a Supervisor nem tárolja a JA táblázatokban, nekünk kellett megtennünk.

Mielőtt a blokkméret tárolásának problémájával foglalkoznánk, meg kell említenünk, hogy természetesen megvizsgáltuk az I/O tevékenységeket kisérő egyéb időfaktorokat is /megszakításkezelések, hibafeltárás stb./, de ezek döntő többségükben a rendszer program bontható rezsiidejébe tartoznak, amelyeknek minősitéséről már az előzőekben említést tettünk.

A blokkméret letárolását a futó program végrehajtása közben kell elvégezni. Ezt a feladatot a mi software környezetünkben két helyen láthattuk el. Az első lehetőség a supervisor program, a második a POWER.
A supervisor módositását a szükös főtárterület, valamint az a tény vettette el velünk, hogy a supervisoron belül sok helyen és alapvető logikákba kellett volna belenyúlnunk.
A POWER müködésének egyik alapvető pillére, nevezetesen az SVC kezelés adta meg számunkra a lehetőséget.
Mivel a POWER struktúráját egy-két alapsortware-t érintő hiba elháritása, valamint üzemeltetési környezetünk konformitását befolyásoló fejlesztésünk kapcsán meg kellett ismernünk, kézenfekvőnek tünt a POWER módositása

Átalakitásunk lényge abból állt, hogy a POWER-on belülről, periféria cimek valamint azon belül állományok -CCB-k - szerint leszedtük a blokkméretet és azt bizonyos rendezőelveknek megfelelően letároltuk a supervisor területén.
Job-step végén, amikor a felhasználói JA rutin háttértára menti a JA táblázat információit, a blokkméret adatokat, és még egy-két az adott particióra jellemző idő és egyéb adatot a POWER-ból supervisorba betárolt területről nyeri vissza.
Ennek a fejlesztésnek megvalósitásához több olyan alapsoftware-t érintő megoldást kellett technologizálnunk, amelyek másirányú DOS fejlesztéseink alap moduljai lettek /tárolóvédelem, lemez file-védelem feloldása/.
Ezek után rendelkezésünkre álltak mindazok az információk, amelyek birtokában megkezdhettük a kisérleteket azokkal az elszámolási képletekkel, melyek egy üzemszerüen multiprogramozási környezetet feltételező számitóközpont tényleges ráforditásait vetik össze az árképzési szabályzatban meghatározott normativákkal.

Kialakitandó árainkban lényeges szerep jut a papirköltségnek. A felhasználás mértékének automatizálását szintén a JA rendszer részévé kellett tenni. Az operációs rendszer ezen belül a supervisor által szolgáltatott JA információk elsősorban a nyomtatott sorok - pontosabban a nyumtatókra kiadott SIO-k - vonatkozásában nyujtanak támaszt. Költség és ár oldalról nyilvánvalóan a felhasznált leporelló lapok száma informativ. A POWER rendszer - helyesen - a nyomtatott lapok száma szerint adminisztrál, de mivel a POWER saját elszámolási rendszerét nem használjuk, igy a lapszámlálást a "blokkméret" problémakör keretein belül oldottuk meg.

Most amikor az előadás szövege leadásra kerül, az előző bekezdésben emlitett "kisérletek" fázisában tartunk. Reménhetőleg az előadásban, előszóval már a megvalósulás fázisáról számolhatunk be.

BASIC INTERPRETER A GD80 GC-N

Rattinger Márta
MTA SZTAKI

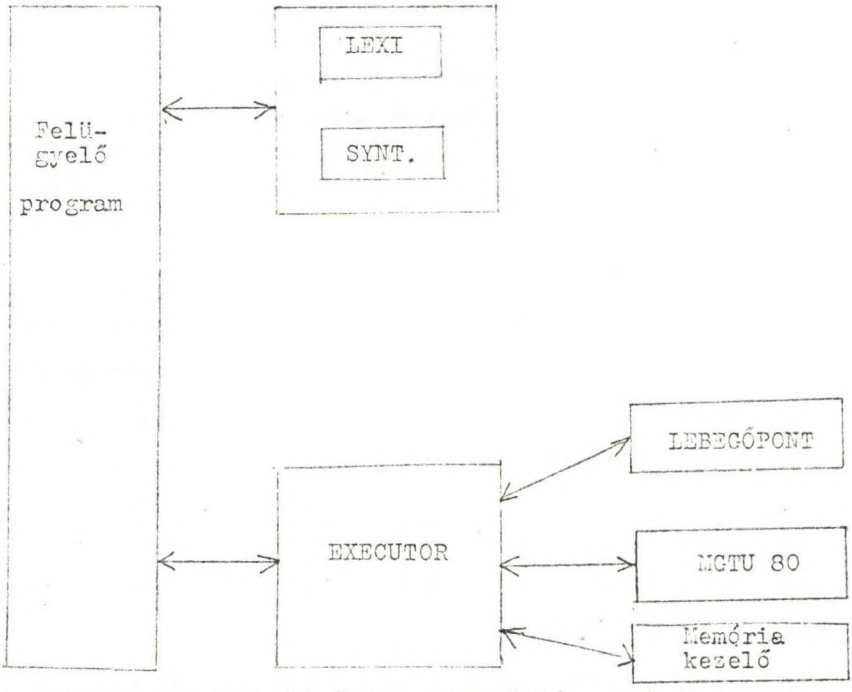
1. Bevezetés

Intézetünkben folyik a GD80 grafikus display család fejlesztése, amely INTEL 8080 mikroprocesszoron alapszik. A család legegyszerübb tagjának, a GD80 GC-nek software-ét egy felhasználás operációs rendszer, BASIC programozási nyelv és az MGTU80 szubrutin csomag alkotja.

A GD80 GC-n megvalósitott BASIC az ECMA szabványt tartalmazza grafikus bővitéssel.
Igy olyan nyelvet nyertünk, amelynek utasitásai könnyen megjegyezhetők, nincs szükség használatakor nagy programozói gyakorlatra, ugyanakkor alkalmas müszaki-tudományos és grafikus szolgáltatásokat igénylő feladatok megoldására.

2. Az interpreter felépitése

A GD80 BASIC felépitését az alábbi ábra mutatja:



Az interpreter funkcionálisan különálló egységeket tartalmaz./A fent vázolt teljes rendszer megvalósitása az alábbiak munkája: Darvas Péter, Déri Gábor, Gallai István, Hosszu Péter, Rattinger Márta./

A lexikális és szintaktikus analizist végző programok, valamint az executor a végrehajtás szempontjából egy szekvenciát alkotnak. Nevezzük ezeket az egységeket boxoknak.

A boxok a felügyelő programhoz és a memória kezelő programhoz kapcsolódnak, az executor pedig használja az MGTU80 és a lebegőpontos szubrutin csomagot.

A felügyelő program feladata a forrásnyelvü sorok bevitele, a BASIC program editálása, az utasitásonkénti végrehajtás vezérlése.

A felügyelő program szervezi az adatátadást a boxok között, ugyancsak ez a program tart kapcsolatot az operációs rendszerrel.

A lexikális analizis bemenő adata egy pointer a forrásnyelvü sorra, outputja egy ugynevezett token sorozat. A token egy lexikális egység, amelyet token osztály és - amikor az értelmes -, az osztályon belüli érték alkot.

A grafikus bővitéshez használtuk az MGTU80 szubrutin csomagot. Az MGTU rutinok használatával jelenithetők meg az ábrák és kezelhetők az input eszközök.

A lexikális analizis befejezése után a szintaktikus boxnak adódik át a vezérlés. A szintaktikus box a bemenő token sorozatot atom sorozattá transzformálja, amely az executor bemenő adata lesz. Az atomok változó hosszuságuak, tipusjelből és értékből állnak.

A felügyelő program az analizis boxokat kétféle módon hivhatja meg. Egy forrásnyelvü sor begépelésekor célszerü azt szintaktikusan ellenorizni, igy a hiba azonnal javitható.

A program futásakor az analizis boxokban az ellenőrzésen kivül végrehajtódnak a számkonverziók, majd a felügyelő program átadja a vezérlést a kódgenerátornak.

Az executor bemenete az utasitásnak megfelelő atom sorozat, amelyen belül az aritmetikai kifejezések forditott lengyel formában adódnak át. Változóra történő hivatkozás esetén az executor a memória kezelő programhoz fordul. Ez a program egy HOAR-féle monitor, kizárólagosan látja el a memóriagazdálkodással kapcsolatos feladatokat. A memória

kezelő egység végzi a tömbök, string változók és a DATA utasitások adatmezői számára szükséges adminisztrációkat.

3. Részletesebb leirás

3.1. A lexikális boxban a BASIC alapszavak és a standard függvénynevek egy prefix felismerő automata végzi. Az eljárás használ egy induló vektort és egy állapot táblát. Az alapszó első betüjének kódja index az induló vektorhoz. Minden vektorelem pointer az állapot táblára. A vektorelem nulla volta azt jelzi, hogy nincs az adott betüvel kezdődő alapszó. Ez esetben a beolvasott betü változót jelöl. Az állapot tábla átmenet sorozatokat tartalmaz a további betük feldolgozásához. Az automata végállapotba jut, ha felismert egy alapszót - ekkor kimenete az alapszó kódja -, vagy ha a már azonositott betük a most beolvasottal együtt nem illeszkednek egyetlen alapszóra sem. Ebben az esetben az automata hibajelzéssel fejezi be működését.

3.2. Az általunk megvalósitott BASIC nyelvben előfordulhatnak sorszám nélküli, ugynevezett immediate utasitások. Az immediate utasitások azonnal végrehajtódnak, ily módon a nyelv használható egyszerü kalkulátorként is. Ezek az utasitások lehetőséget adnak arra, hogy a STOP vagy PAUSE utasitással felfüggesztett program változóinak értékét megvizsgáljuk, módositsuk, a programot adott sorszámtól kezdődően inditsuk tovább.

3.3. A felügyelő programnak - más BASIC rendszerekhez hasonlóan - adhatók BASIC alapszavaktól különböző parancsok. A parancsszavakkal a felhasználó közölheti a programmal kapcsolatos kívánságait. Kérhet a programról listát, megőrizheti azt egy file-on, törölheti a file-ról a korábban kimentett programot, kérheti a memóriában levő program futtatását.
Forrás sorok bevitele közben valamint javításkor a felügyelő program editor szolgáltatást is nyujt.

3.4. A felhasználó által hivható MGTU80 rutinok a BASIC nyelvből CALL NÉV /paraméter lista/ formában érhetők el. A lehetséges szubrutinok nevét és paramétereik számát egy táblázat tartalmazza, amely egyszerüen bővitheto.

3.5. Az összes lehetséges egyszerü változó számára foglalódik hely a memóriában, értéküket a program futásának kezdetén nullára inicializálódik.Fix méretü hely áll rendelkezésre a tömbök, string változók és DATA adatmezők számára. Ez a terület dinamikusan kerül felosztásra.

4. A rendszerrel kapcsolatos tervekről és a felmerült kérdésekről

4.1. A program tárolásának módja alapvetően befolyásolja a memória igényt és a gyorsaságot. Jelenleg a forrásnyelvü programot tároljuk, amelynek helyigénye nem nagy, viszont minden végrehajtás előtt ujra le kell forditani a sorokat, ciklus esetén többször is.

Megoldható lenne, hogy egy munkafile-on tároljuk a szintaktikus analizis által előállitott tömöritett formát, s az igy előállitott kódot hajtassuk végre az executorral. Ez lényegesen növelné a sebességet, különösen sok ciklust tartalmazó program esetén. Ugyanennek a problémának másik megoldása lehetne, hogy csak a közbülső formát tároljuk kiegészitő információkkal együtt, és ebből szerkesztenénk ujra listázáskor a forrásnyelvü szöveget.

4.2. Mivel a memória kezelő program kizárólagosan végzi a tároló beosztást és önálló egységként van megirva, könnyen fel lehet cserélni a meglevő programot egy továbbfejlesztett változattal.
Jelenleg a BASIC programban használható tömbök összméretét korlátozza a részükre fenntartott memória terület nagysága.
A rendszert a jövőben ki akarjuk bővíteni kétszintü tárolóval.

4.3. A BASIC rendszer mostani állapotában két, egymástól jól elválasztható részből áll. Ennek előnye, hogy az egyes elemek és a köztük levő kapcsolatok könnyen áttekinthetők. Valószinünek látszik azonban, hogy bizonyos egységek egybeépitésével helyet és időt is nyerhetnénk.
A lexikális és a szintaktikus box egybeépitése nem jelentene tul sok előnyt, viszont megfontolandó a kérdés a szintaktikus box és az executor esetében.
A meglevő változatban a szintaktikus box előállitja az egész utasitás közbülső formáját és ezt adja át az executornak.

Az executornak végrehajtás előtt ujra fel kell ismernie a kapott adatokat. Megoldás lehet a következő: miután a szintaktikus box kódolt egy végrehajtható egységet, hivja meg az executort és hajtassa is végre a szükséges müveleteket.

5. Befejezés

Az interpretert a GEZA rendszerprogramozói nyelven irtuk és TPA-70-es számitógépen próbáltuk ki, majd a kipróbált változatot vittük át a GD80 GC-re.
Ezt a megoldást az tette lehetővé, hogy a GEZA compiler először egy közbülső kódot generál, amiből mind TPA-70, mind pedig INTEL 8080 kód előállitható.

IRODALOMJEGYZÉK

1. STANDARD ECMA - FOR MINIMAL BASIC
   ECMA ITC21/75

2. GERHARDT,G.: GÉZA LANGUAGE
   A PROGRAMMING LANGUAGE FOR SYSTEM
   DEVELOPMENT
   MTA SZTAKI BUDAPEST-1976

3. A GUIDE TO TIME SHARED BASIC
   SOFTWARE PUBLICATIONS-CUPERTINO,CALIFORNIA-1969

4. SARKADI NAGY I.-SZLANKÓ L.: BASIC 70
   MTA KFKI BUDAPEST-1974

5. MU BASIC IRT-11 - USER'S MANUAL
   DIGITAL EQUIPMENT CORPORATION - MAYNARD, MASS. - 1975

6. LEWIS,P.M.,II. - ROSENKRANTZ,D.J. - STEARNS, R.E.:
   COMPILER DESIGN THEORY
   READING, MASS. ADDISON-WESLEY-1976.

KÖZUTI KÖZLEKEDÉSI HÁLÓZATOK FEJLESZTÉSÉT ÉS TERVEZÉSÉT SZOLGÁLÓ PROGRAMRENDSZER

Dr. Scherr Károly
Közuti Közlekedéstudományi Kutató Intézet

Az életszínvonal és a motorizációs szint emelkedésével, valamint a térbeli munkamegosztás növekedésével párhuzamosan nőnek a közlekedés problémái világszerte. Hazánkban a 70-es évek jelentik azt a döntő időszakot, mikor a közlekedés tervezésében és fejlesztésében minőségi változásnak kell bekövetkeznie ahhoz, hogy a megnövekedett igényeket kritikus feszültségek létrejötte nélkül gazdaságosan és biztonságosan lehessen kielégíteni.

A KPM 5. sz. kutatás-fejlesztési célprogramja keretében a KÖTUKI-ban /Közúti Közlekedési Tudományos Kutató Intézet/ már évek óta folynak kutatások az "analitikus" forgalomelőrebecslési módszer kifejlesztése érdekében. E módszer a forgalomkeltés, forgalomszétosztás és forgalomráterhelés lépésekben határozza meg a távlati közúthálózati terheléseket. 1975-ben a módszer továbbfejlesztése érdekében újabb kutatások indultak meg, melyek egy differenciáltabb eljárás kidolgozását tüzték ki feladatul. A téma aktualitása külés belföldön egyaránt igen nagy. A KGST KÁB és OSzSzD gondozásában két téma is foglalkozik a közúti forgalom előrebecslési módszereivel.

Az elvégzett vizsgálatok eredményeképpen kapott modellösszefüggések realizálására számítógépes programrendszer készült /ld. a folyamatábrát/. A gépi feldolgozás lehetővé teszi nagy adattömegek bevonását, pontos feldolgozását és több tervvariáns előkészítését aránylag rövid idő alatt a szükséges müszaki-gazdasági döntések alátámasztására.

1./ ADATINPUT /INJAV/

A számításokhoz szükséges adatok bevitelét és esetenkénti javítását az INJAV programszegmens végzi. A bevitel történhet kártyáról, szalagról, vagy e kettő kombinációjával. E programszegmens végzi az adatok logikai és korlátellenőrzését is. A hibátlan adatokat tárolja és kilistázza, míg a logikai, korlát, vagy konverziós hibákat tartalmazó rekordokat külön listára írja. A hibás rekordokat javítás után egy javítófutás keretében újra be kell olvastatni. A feldolgozandó adatfajtáknak megfelelően a HÁLIN, STRIN, ÁRMIN alprogramok állnak rendelkezésre, melyek szükségszerinti behívását a vezérkártyákon lévő paraméterek végzik.

1.1. Hálózati adatok /HÁLIN/

A hálózati kapcsolatokat és a szakaszjellemzőket /hossz, szakasztípus, megnevezés, stb./ olvassa be és tárolja. Az érvényes és hibás adatokat külön listázza.

1.2. Struktúrális adatok /STRIN/

A forgalmi körzetek releváns statisztikai jellemzőit /lakószám, foglalkoztatottak száma, járműállomány, stb./ olvassa be és tárolja. E forgalmi körzeteken értelmezzük a következő pont áramlási mátrixát.

1.3. Áramlási mátrix /ÁRMIN/

A forgalmi körzetek között mért áramlások mátrixát olvassa be és tárolja. Csak diagnózis állapotban kerül felhasználásra.

2./ DIAGNÓZIS ÁLLAPOT SZÁMITÁSAI /DIAGN/

Az ú.n. diagnózis állapotban /jelen hálózati és áramlási viszonyok/ végezzük a modellrendszer kalibrálását, a szükséges paraméterek beállítását és a számítási eljárások ellenőrzését.

2.1. Adatbevitel /DIAIN/

A vezérkártyákon megadott forrásokból /szalagok/ disk-re olvassa a releváns adatokat.

2.2. Minimál utak számítása /OPTUT/

Továbbfejlesztett LOUBAL - vektor algoritmussal számítja a hálózati gráf egy-egy kifeszítő fájának megoldásvektorát tizedpercekben.

2.3. Potenciálok számítása /POTENC/

A 2.2. pontban ismertetett OPTUT által kiszámított eljutási idők alapján számítja a forgalmi körzetközéppontok helyzetpotenciálját.

2.4. Forgalom szétosztás /OSZT/

Feladata a gravitációs modell ellenállásfüggvény-típusának és paramétereinek meghatározása. Az opciókártyákon megadott függvénytípusokra és paraméter intervallumokra kiszámítja az ellenállásmátrixot, majd egy RAS feladat megoldásával illeszti ezt a mért áramlási mátrix marginálisaira. Méri az illesztés pontosságát különböző statisztikai jellemzők segítségével. A számítássorozat eredményeképpen meghatározható a legjobb illeszkedést biztosító függvénytípus és paraméterei.

ELEM SEGÉDPROGRAMOK

Ez a programrendszertől különálló segédprogram végzi a korrelációszámítást a körzetek statisztikai jellemzői és potenciáljai, valamint a mért forgalmi értékek között és szolgáltatja a paramétereket a prognózis állapot programláncának KELT eljárásához.

3./ PROGNÓZIS ÁLLAPOT SZÁMITÁSAI /PROGN/

A kiválasztott tervezési időpontra /pl. 1990, 2010 stb./ prognózist szolgáltat a várható áramlási vizsonyokról.

Input-ként a tervezési időpontra vonatkozó struktúrális adatok és hálózati viszonyok szükségesek.

3.1. Adatbevitel /PROIN/

Az opciókártyákon megadott forrásokból beolvassa a prognózis állapot releváns adatait.

3.2. Minimál utak számítása /OPTUT/

Mint 2.2., de a prognózis állapot hálózatán értelmezve.

3.3. Potenciálok számítása /POTENC/

Mint 2.3., de a prognózis állapot struktúráján értelmezve.

3.4. Forgalomkeltés /KELT/

A prognózis állapot struktúrális adatai, a 3.3. pont potenciáljai és az ELEM segédprogram által adott paraméterek alapján számítja az egyes körzetközéppontokban keletkező és az oda irányuló forgalmakat.

3.5. Forgalomszétosztás /OSZT/

A diagnózis állapotban meghatározott és opciókártyán szolgáltatott ellenállásfüggvény-típus és paraméterek felhasználásával a 3.4. alatt kiszámított marginálisokhoz szolgáltatja az áramlási mátrixot.

4./ FORGALOMRÁTERHELÉS /TERH/

A 3.5. alatt számított áramlási mátrixot CAPACITY RESTRAINT típusú modell segítségével az opciókártyákon megadott számú lépésben és százalékos elosztásban terheli rá a hálózatra. Kiszámítja a hálózati elemek terhelését, az azokon kialakuló sebességviszonyokat, továbbá a teljes hálózatra vonatkoztatott összes futáskilométert és futásidőt, valamint ezek hányadosaként az átlagos áramlási sebességet. Ezen utóbbi mérőszámok alkalmasak a hálózati variánsok értékelésére.

5./ MÉRTÉKADÓ FORGALOM MEGHATÁROZÁSA /MÉRTF/

A különböző forgalmi rétegek /személy, teher, hétvégi, nemzetközi/ terheléseinek összevetésével kiválasztja tervezés alapját szolgáló mértékadó szakaszterheléseket.

6./ A PROGRAMRENDSZER ÁLTALÁNOS JELLEMZŐI

Forrásprogram: PL1 OPTIMIZING

Környezet: DOS/VS operációs rendszer
IBM 370/145 hardware

Tekintettel a jelentős méretekre a program "OVERLAY" technikával fut. A DIAGN, PROGN, TERH és MÉRTF képez egy-egy önálló programláncot, ahol az egyes eljárások /pl. DIAIN, OPTUT, POTENC stb./ egymás után egymás helyére kerülnek az operatív tárba.

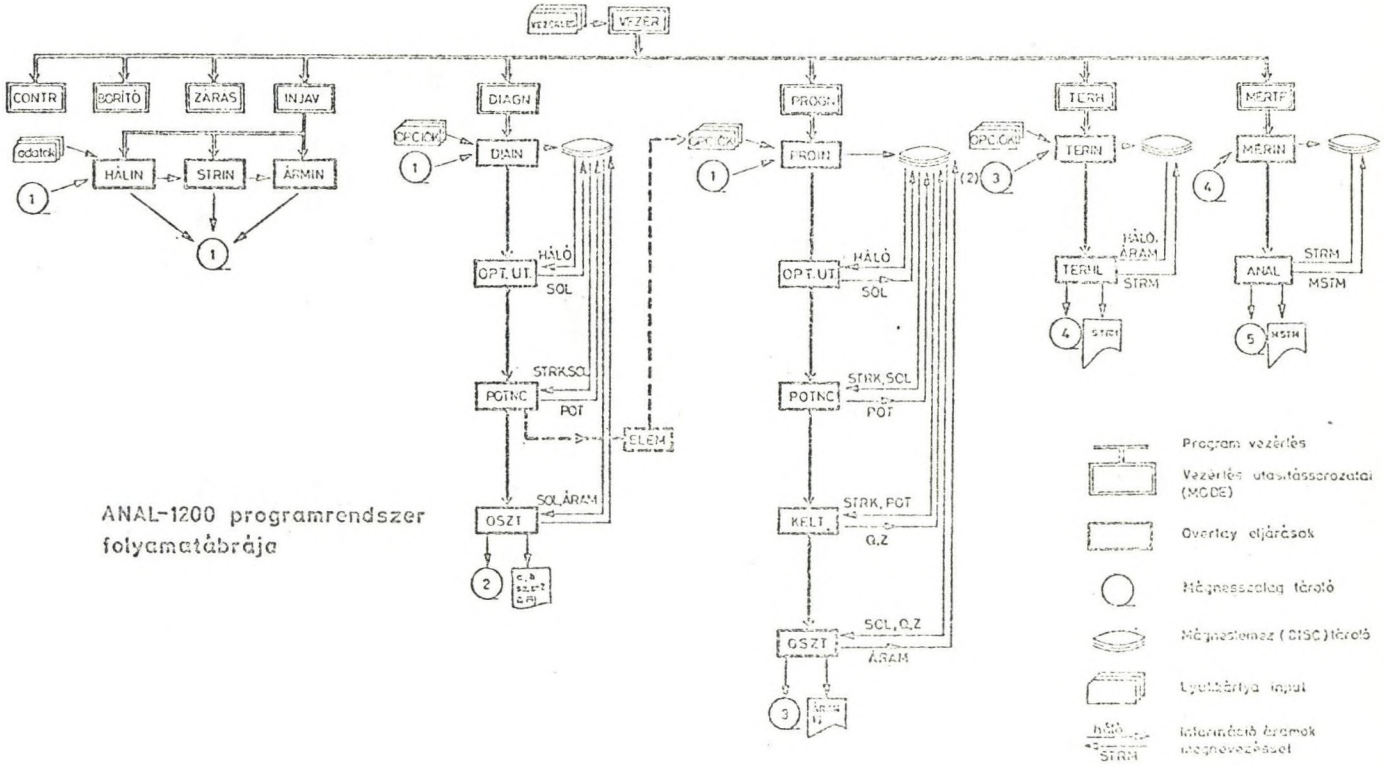
Egy-egy programláncot vezérkártyákkal és opciókártyákkal vezérelhetünk. A vezérkártyákon szerepelnek a fő paraméterek, míg az opciókártyák egy-egy belső eljárás paramétereire vonatkoznak. Az eljárások közötti adatkommunikációt az operatív tár statikus mezői és a külső disk-területek biztosítják.

Az adattömbök és disk-területek kijelölése dinamikusan történik, azaz mindig csak az aktuális modellméretnek megfelelő operatív és háttértárakkal dolgozunk.

A program jelen kialakításában maximálisan 1200/360-as modellméret feldolgozására alkalmas, e méthatár kijelölése azonban gyakorlati okokból történt és a méret növelésének elvi akadálya nincs.

Néhány segédprogram gondoskodik a mellékfunkciók ellátásáról. A CONTR ellenőrzéprogram a vezérlés helyességét ellenőrzi. A BORITÓ készíti el az outprint nyitó- és záróoldalát a legfontosabb programjellemzők nagyméretü kiírásával. A ZÁRÁS program a hátsó borító előtti utolsó oldalra kiírja a feldolgozás szempontjából legfontosabb paraméreket /pl. input és output szalagok száma, modellméret, elvégzett müveletek stb./.

A programlánc kialakítása, belövése és futtatása a SZÁMOK számítóközpontjában történt. Az első futtatásokból nyert tapasztalatok alapján a programrendszert folyamatosan továbbfejlesztjük.


VEZÉRLŐ
VEZÉR
CONTR
BORÍTÓ
ZÁRÁS
INJÁV
adatok
HÁLIN
STRIN
ÁRMIN
DIAGN
OPCIÓK
DIAIN
OPT.UT.
HÁLÓ
SOL
POTNC
STRK,SOL
POT
ELEM
OSZT
SOL,ÁRAM
PROGR
PROIN
KELT.
STRK, POT
G.Z
SCL, G.Z
ÁRAM
TERR
TERIN
TERÍL
HÁLÓ, ÁRAM
STRM
MÉRTF
MÉRIN
ANAL
MSTM
ANAL-1200 programrendszer folyamatábrája
Program vezérlés
Vezérlés utasításssorozatai (MODE)
Overlay eljárások
Mágnesszalag tároló
Mágneslemez (DISC) tároló
Lyukkártya input
Információ áramok megnevezésével

OPERÁCIÓS RENDSZEREK FEJLESZTÉSÉNEK KORSZERŰ MÓDSZEREIRŐL

Simor Gábor
Számitástechnikai Koordinációs Intézet

0. Bevezetés

Az operációs rendszerek és a hardware-firmware eszközök közötti határvonal elmosódása, a különféle speciális processzszorok megjelenése /[1],[2]/ következtében különösen aktuális feladattá válik a konfigurációs követelményeknek eleget tevő operációs rendszerek flexibilis kialakithatósága, másrészt pedig az operációs rendszerek felé közelitő egységes architektura fejlesztési irányok meghatározása.

Az elmult évek kutatási eredményeinek az operációs rendszerek tervezésénél való hasznositási lehetőségeit jól áttekintette az "An Advanced Course on Operating Systems" cimmel megtartott rendezvény /[3]/, amelynek központi gondolata az operációs rendszerek un. "objektum modellek" /[3.1]/ alapján történő létrehozása volt.

1. Az objektum modellek kialakitási elve

Az operációs rendszerek kialakitásához ma már nem elegendőek a több mint egy évtizeddel ezelőtt kialakult olyan absztrakciós fogalmak, mint a file és a processz. Az operációs rendszerek által kezelt objektumok tipusféleségeinek növekedése, a közöttük fennálló kapcsolatok elbonyolódása indokolja az un. tipus modulok kialakitását, amelyek az azonos tipusu objektumok egységes és megbizható kezelését teszik lehetővé azáltal, hogy a bennük leirt objektumhoz való hozzáférés csak az objektumhoz rendelt müveleteken keresztül lehetséges. A tipus modulnak a specifikációs része által meghatározott müveletek hatása határozza meg a környezet számára az objektumot, nem is "látható" az objektum leirásának többi része: a tipus modul reprezentációs része, és implementációs része, amely a specifikált müveletek más /alacsonyabb szintü/ tipus modulokban specifikált müveletekből felépitett algoritmikus megvalósitását tartalmazza. Ilymódon tetszö-

leges számitástechnikai rendszer áttekinthető leirása elkészithető tipus modülok hierarchiájának felépitésével és ez utóbbit nevezik a rendszer objektum modelljének.

Operációs rendszerek objektumaira példák lehetnek: process , message, page, segment, file, file subtree, record, field, message channel /mailbox/, minidisc, virtual machine, devices /tape drive, disc pack, terminal/, device partitions /tape block, terminal CRT window/, stb.

Példák objektumokra müveleteikkel együtt:

Process:	Create	Mailbox:	Create
	Destroy		Destroy
	Fork		Send
	Join		Receive
	Schedule		Conditional Receive
	Unschedule		

Az objektum modell felépitésének stratégiájára különösebb megkötöttségek nincsenek /pl. top-down v. bottom-up/. A tipus modulok szerinti strukturálás és magának a tipus modulnak a belső kialakitása igen kedvező a programhelyesség bizonyithatósága szempontjából is.

2. Az objektum modell kialakitásának főbb aspektusai

Az objektum modell szemléletében kell megoldásokat találni az operációs rendszer lényeges aspektusának problémáira is:

- az objektum elnevezések kezelésére;
- a védelem biztositására;
- a szinkronizálás biztositására.

Az objektumok elnevezéseinek használatánál minimális követelmény, hogy az operációs rendszer irására felhasznált programozási nyelvek forditóprogramjai elvégezzék a megjelölt objektum neve és reprezentációjának neve közötti leképezést. Ez elvégezhető dinamikusan is az operációs rendszer által - ekkor az objektum név a tipus modul specifikációs részében meghatározott valamely müvelet paramétereként kerül átadásra és a leképezést a tipus modul "takart" részei /reprezentációs, implementációs/ végzik - ez azonban hatékonysági problémákat vethet fel. A jelenlegi számitógép architekturák többnyire csak a memória szegmens tipusu objektumok nevének /azaz cimeknek/ a kezelését végzik.

Az objektumokhoz való illetéktelen hozzáférés elleni védelem biztositása megkivánja az objektum modellekben a hatáskörök pontos meghatározását. Egy hatáskör meghatározása egyrészt

azoknak az objektumoknak a felsorolását jelenti, amelyekhez a vonatkozó tevékenység egységnek /process, program, stb./ hozzáférési joga van, másrészt ezen tulmenően azoknak a müveleteknek a felsorolását, amelyeken keresztül van hozzáférési joga az adott tevékenységnek. Valamilyen szinten biztositani kell azt a mechanizmust, amely a tipus modulokban specifikált müveletek hivásait összeveti a kurrens tevékenység hatáskörével. További probléma a hatáskörök dinamikus módosításának igénye.

Az egyes tevékenységek párhuzamos végrehajtásából adódó konfliktusok elkerülését biztositó szinkronizálásnak az objektum modellhez leginkább illeszkedő módja a tipus modulok specifikációs részében meghatározott müveletek egymáshoz való időbeli viszonyára vonatkozó megkötöttségeknek a rögzitése magában a specifikációs részben. Ez a különféle eddig ismert szinkronizálási eszközök közül az un. "path expression"-nek /[3.2.]/ felel meg leginkább.

3. A korszerü módszerekkel szembeni követelmények összefoglalása

a., Jól el kell különiteni szabványos tipus modulokban az operációs rendszernek azokat az objektumait, amelyek azonos - az adott objektumot kezelő - müveletekkel jellemezhetők;

b., Az egyes modulokban jól el kell különiteni az objektum és müveleteik specifikációs részét /amely a modulon kivülről látható/ a megvalósitásnak módját leiró részétől;

c., Jól el kell különiteni a rendszerben minden tevékenységet olyan csoportokba, amelyek hatáskörrel - azaz a tevékenységből jogosan elérhető objektumok és müveleteik felsorolásával - rendelkeznek;

d., Biztositani kell, hogy bármely objektumhoz csak a modulja specifikációs részében feltüntetett müveletein keresztül lehessen hozzáférni és csak akkor, ha hatásköre arra kiterjed;

e., Biztositani kell, hogy az egyes tipus modulokban specifikált müveletek közötti szinkronizálási feltételek egyszerüen meghatározhatók legyenek /és automatikusan érvényesüljenek/;

f., Biztositani kell, hogy a specifikációs részben az egyes müveletekkel szembeni elvárásokat egyszerüen ki lehessen fejezni és teljesülésük a megvalósitási rész leirása alapján ellenőrizhető legyen;

g., Biztositani kell, hogy a kollektiv hozzáférésü objektumokra olyan nevekkel lehessen hivatkozni a különböző - önálló névhasználattal jellemezhető - tevékenységekből, amelyek hatéko-

nyan leképezhetők és hatékonyan használhatók az adott tevékenységen belüli környezetben is;

h., Biztositani kell, hogy könnyen kapcsolhatók legyenek a módszerhez hirarchikusan strukturált grafikus dokumentálási módszerek /"HIPO v. SADT"-szerü: [4.] , [5.]/

i., Biztositani kell a kapcsolatot a firmware-hardware rendszerek kialakitásával is, mind a kialakitás egységes technológiája, mind a módszerrel kialakitott alapsoftware hatékony müködtetése tekintetében.

4. Az objektum modell alapján kialakitott rendszerek számitógép architekturáinak tulajdonságai

4.1. Alapvető tulajdonságok

a., Közvetlen és hatékony hozzáférése van a
- hatáskőri listákhoz /feltételes elérési jogokkal/;
- tipus modul specifikációkhoz;
- objektumok különböző nevei közötti leképezések leirásaihoz;
- elérési jog feltétel leirásokhoz.

b., A tipus modulokban specifikált müveletek hivása teljes egészében architekturálisan hajtódik végre, beleértve
- a paraméterként megadott objektum név leképezését a tényleges reprezentációhoz való hozzáférést biztositó névre;
- az elérés jogosságának ellenőrzését a hatásköri lista és az elérési feltétel leirásokkal való összevetés utján;
- a szinkronizációs feltételek teljesülésének ellenőrzését beleértve a várakozási sorok kezelését is;

c., A hatáskőri lista és az objektumnév leképezési leirás dinamikus kapcsolása tevékenységcsoportok közötti hivások esetén.

4.2. Kiegészitő tulajdonságok

a., A tipusmodulokban specifikált müveletekkel szembeni elvárások teljesülésének dinamikus ellenőrzése;

b., Az egyes tipus modulokban specifikált müveletek használati gyakoriságának, időigényességének mérése, müveletek, ill. teljes tipus modulok HW/FW kiváltásának képessége.

5. Az objektum modell szemléletü technológiák megjelenése

A fentiekben vázolt technológia egyre inkább elérhető közelségbe kerül, ezt jelzik az alábbiak is:

a., Kidolgoztak több, tipus modulokon alapuló kisérleti nyelvet /ALPHARD, CLU, EUCLID/, közülük néhány szinkronizációs eszközöket is tartalmaz /GYPSY, MODULA, Concurrent Pascal/, /[6.]/;

b., Igen kis ráforditásokkal készitettek kisérleti operációs

rendszereket a tipus modulok több lényeges tulajdonságoát tartalmazó nyelveken. A Concurrent Pascal nyelv /ld. még 6./ implementációja Magyarországon is hozzáférhető két igen elterjedt számitógép sorozaton /PDP 11, IBM 37o /DOS/VS//, /[7.]/;
c., Nagyobb cégek is kezdik megvalósitani ezt a technológiát: a Hewlett-Packard Co. "Syspal" rendszere egyesiti az ilyen tipusu nyelvek tulajdonságait és "lehetővé teszi az operációs rendszerek felhasználók általi létrehozását a kivánt rendszer tipus tevékenységek és a konfiguráció definiálása alapján" /[8.]/, /[9.]/.

6. A Concurrent Pascal nyelv és egy maximális elvárásokat kielégitő /több hasonló rendeltetésü nyelv és kapcsolódó módszerek előnyeit egyesitő/ rendszerprogramozási nyelv közötti különbségek.

a., A monitorokban az objektumokhoz csak egymást időben kizáró müveletekkel lehet hozzáférni, igy a modulokra bontás nem a legtermészetesebb módon történik, miszerint az egyes tipus modulokat az azonos müveletekkel hozzáférhető objektumok határozzák meg;
b., A modulokban nem szerepelnek az egyes müveletekkel szembeni elvárások /eltekintve az elérési jogoktól/, amelyet a szekvenciális algoritmus-helyesség vizsgálat feltételez;
c., Egy modul /monitor/ valamely müveletének hivásakor foglaltság esetén a várakozásról magának a hivásnak kell gondoskodnia;

d., Az elérési jogok csak objektumra /modulra/ vonatkozhatnak, nem finomithatók objektumonként még müveletekre is;
e., A rendszer sok tekintetben statikus: processzek nem törölhetők, elérési jogok dinamikusan nem változtathatók , ami sem hatékonysági szempontból, sem az áttekinthetőség szempontjából nem előnyös.

7. Tipus modul példa

Az alábbiakban ismertetett "mailbox"-példa sok tekintetben csak vázlatos illusztrációnak tekinthető, többek között az invariáns, az input-output, a kezdeti feltételek nincsenek kifejtve a /kézi/ verifikálásnoz szükséges teljes részletességgel /amit több vonatkozásban indokol a specifikációs rész "path restrictions" leirásának jóval tömörebb volta is/; az implementációs rész csak részben kidolgozott, feltételezett alacsonyabb szintü tipus modulok használatán alapul; a felirásmód egy ALPHARD-szerü /[6.1]-ben szintén csak illusztrativ célzattal használt/ elképzelt szintaxison alapszik.

```
form mailbox (T:form(←,=), D:form(schedule,unschedule with backtrack),
              n:integer)
  beginform
  specification
    requires n>∅; invariant mailbox=<sender, receiver, queue, wait> where
    isinteger(sender,receiver)∧isboolean(wait)∧isseq(queue); initially
    queue=nullseq∧sender=∅∧receiver=∅∧wait=∅;
    function
    create(m:mailbox,r:integer) post iscurdom(sender)∧receiver=r
    destroy(m:mailbox) pre iscurdom(sender)
    send(m:mailbox,x:T) pre iscurdom(sender) post queue=queue'~x∧if
        length(queue)=1∧wait=1 then wait=∅
    receive(m:mailbox)returns(x:T) pre iscrdom(receiver)∧wait=∅ post if
        length(queue)=∅ then wait=1∧to be repeated on(wait=∅) else
        x=first(queue')∧queue=trailer(queue')
    conditional receive(m:mailbox)returns(x:T) pre iscurdom(receiver)∧wait=∅
        post if length(queue)>∅ then x=first(queue')∧queue=trailer(queue')
    size(m:mailbox) returns(x:integer) pre iscurdom(sender) post x=length(queue')
    path restrictions
    path create;receive,conditional receive;destroy end
    path create;(send - receive,conditional receive)^n;destroy end
    path create;{size};destroy end
  representation
    unique f:fifo(T,n),r:integer,w:boolean; init r,w← ∅; rep (f,r,w) =
    <queue,receiver,wait>; invariant isseq(fifo)∧∅≤length(fifo)<n
  implementation
    body create=generate(i:instance,m:mailbox),amplify(mc:modcap,m:mailbox),
        m.r←r
    body destroy=revoke(mc:modcap), cancel(i:instance)
    body send=enqueue(f:fifo,x:T),size(f:fifo)returns(y:integer)if y=1
        and m.w=1 then schedule(m.r:D),m.w ← ∅
    body receive=size(f:fifo)returns(y:integer)if y=∅ then m.w←1,
        unschedule with backtrack(m.r:D)else dequeue(f:fifo)returns(x:T)
    body conditional receive=size(f:fifo)returns(y:integer)if y=∅ then
        dequeue(f:fifo)returns(x:T)
    body size=size(f:fifo)returns(x:integer)
  endform
```

A mailbox két, tetszőleges* /generálási paraméterként megadható/ tipus modullal /form-mal/definiálható tevékenység egység /D:domain/ közötti egyirányu üzenetforgalom lebonyolitására szolgál. Az üzenetek tipusa szintén egy tipus modul hozzárendelésével* adható meg. A beadott, de még kiolvasatlan üzenetek száma maximált /n/. Egy mailboxon végezhető müveletek segitségével az létrehozható, törölhető, üzeneteket küldhet bele a létrehozó domain, üzenetet olvashat ki a létrehozáskor megadott rendeltetés-domain várakozással /ha a mailbox üres/, vagy anélkül /conditional receive/. A mailboxon végezhető fenti müveletek csak szekvenciálisan hajtódnak végre, ezekkel párhuzamosan bármikor kiadható a "bennlévő" üzenetek száma /"size" müvelet/.

A specifikációs részben meghatározott elvárások leirásánál használt jelölések:

isinteger, isboolean, isseq	- a zárójelben következő objektum tipusára vonatkozó állitás
nullseq	- üres sorozat
length(s)	- az s sorozat hossza
first(s)	- az s sorozat első tagja
trailer(s)	- az s sorozat első tagját lehagyva kapott maradék
s ⌢x	- az s sorozat végéhez utolsó tagként x illesztve
o'	- egy müvelet végrehajtása előtti o objektumra hivatkozás a végrehajtás után
iscurdom(x) /is current domain/	- a végrehajtandó müveletet hivó domain: x
toberepeatedon(p)	- amint a p állitás igazzá válik, a vonatkozó müvelet hivása ismét kezdeményeződik

A "path restrictions" részben alkalmazott jelölések:

path .... end	- szekvenciálisan végrehajtandó müveletek sorrendjére vonatkozó állitás
S1;S2;S3	- kötött sorrendben végrehajtandó müveletek
S1,S2,S3	- tetszőleges sorrendben végrehajtandó müveletek
$(S1\text{-}S2)^n$	- S1-t legalább annyiszor végre kell hajtani, mint S2-t, de végrehajtásuk száma közötti különbség nem lehet több n-nél
{S}	- S müveletet tetszőlegesen sokszor ki lehet adni az előzőleg kiadott S müveletek végrehajtásának bevárása nélkül is, de mindnek be kell fejeződnie mielőtt egy következőre /} utánira/ kerülne a sor.

*az egyetlen megkötés, hogy a tipus modulban specifikált müveletek között szerepeljenek a < > jelek között felsoroltak, mivel azok hivására sor kerül a mailbox tipus modulján belül.

A reprezentációs részben van megmondva többek között, hogy a "seq" absztrakt objektum leképezése egy FIFO-sorra, mint konkrét objektumra történik /a "fifo" form teljes kidolgozása verifikálással bemutatva megtalálható:[6.1]/.

A "fifo"-n kivül az implementációs részben még a következő feltételezett tipus modulok müveletei vannak felhasználva:

- valamilyen domain /tipusa a mailbox modulnak paraméterként átadva:D/:
- instance: egy adott tipus modult paraméterként átadva annak leirása alapján egy külön példányt generál és töröl
- modcap /modul capabilities/: a paraméterként átadott tipus müveleteinek használatához szükséges elérési jogok felirása a vonatkozó domain-ek hatáskőri listájába, ill. azok visszavonása.

8. Következtetések

Jól láthatók az olyan technológia jellemzői, amellyel hatékonyan kialakithatók az uj számitógépek "rendszer magjai" biztositva, hogy

- a mag tartalmazza a rendszer megbizhatósága /szinkronizálási és védelmi konfliktusok elkerülése/ szempontjából összes lényeges funkciót;
- egységes "keretet" adjon a meglévő, különböző rendszerkomponensek konfigurálásához;
- kijelölődjék a számitógép architekturák továbbfejlesztésének egységes iránya;
- a magra épülő uj komponensek hatékony kifejlesztéséhez egyaránt alkalmas korszerü, általános software technológiai rendszerként is használható legyen.

Hivatkozások

[1] J.Bondy: Putting Supervisory Routins into Hardware IFIP 77. p.649-653.

[2] Ch.Lecht: The Wawes of Change. VIII. Computer World 1977.

[3] Lecture Notes in Computer Science Vol. 6o. 1978.

[3.1] A.K.Jones: The Object Model: A Conceptual Tool p.7-16

[3.2] K. Lagally: Synchronization in a Layered System p.252-281

[4] HIPO - A Design Aid and Documentation Technique. Manual. IBM 1974. GC2o-1851-0.

[5.] Ross, D.T.: Structured Analysis: A Language for Communicating Ideas. IEEE Transactions on Software Engineering, 3.1, January 1977, pp. 16-34.

[6.] Bedő Á.-Dávid G.-Dömölki B.: Beszámoló az "IFIP TC-2 Working Conference on Constructing Quality Software /Novoszibirszk, 1977 május/" előadásairól. SzÁMKI, 1977 julius.

[6.1.] Wm.A.Wulf,R.L.London,M.Shaw: Abstraction and Verification in ALPHARD

[7.] Groszmann Gusztáv: Egy párhuzamos programozási rendszer. Programozási Rendszerek" 78.

[8.] HP Creating System Language, Comp. World, March 2o,1978 p.2o

[9.] R.A. Fraley:SYSPAL: a Programming System for Operating System Implementation. Compcon 1978. San Francisco

OPERÁCIÓS RENDSZEREK MÓDOSITÁSAI ELSZÁMOLÁSI ÉS INFORMÁCIÓS RENDSZEREK CÉLJAIRA

Surányi Andor
Számitógépalkalmazási Kutató Intézet

Nagyobb számitóközpontok gazdaságos müködéséhez szükséges, hogy az esetleges kezdeti, felfutási időszak után zárt üzemmódot /u.n. closed shop/ vezessenek be. Ugyancsak szükségessé válik a géptermi események automatikus feljegyzése /pl. melyik job, mikor, mennyi ideig futott, milyen erőforrásokat használt/, továbbá ezen feljegyzések automatikus vagy félautomatikus kiértékelése. /pl. az egyes job-ok futási költségének meghatározása/.
A gyártó cégek által szállitott operációs rendszerek erre a feladatra minden átalakitás nélkül, közvetlenül nem használhatók; ez részben indokolt is, mert az információgyüjtési, kiértékelési szempontok számitóközpontonként változnak. /Más kérdés viszont, hogy egyes operációs rendszerek az elszámolási rendszer kifejlesztéséhez szükséges támogatást nem adják ill. nem következetesen vagy nem elegendő mértékben adják/.

Azok az operációs rendszerek, amelyek számitanak a felhasználó /vevő/ ilyen természetü igényére, módositási szándékára; meghatározott pontokon belépési lehetőséget adnak felhasználói eljárások meghivásával. Ezek az eljárások /IBM OS terminológiával exit rutinok/ az operációs rendszerrel meghatározott interface szerint érintkeznek és annak minden szempont-

ból szerves részét képezik /pl. nem felhasználói módban futnak/.

Az exit rutinok természetesen nemcsak a saját elszámolási rendszer, hanem más helyi jellegű módositások /üzemeltetési szabványok, ütemezési stratégia, mágnesszalagos nyilvántartás, stb./ keresztülviteléhez is alkalmas eszközt jelentenek.

A módositások keresztülvitelére igy három lehetőség van:

= Exit rutin irása a rendszerdokumentációban biztositott interface-el. Ez a legegyszerübb és látszatra teljesen sima és veszélytelen vállalkozás.

= Exit rutin irása a rendszerdokumentációban nem biztositott interface-el. Pl. az interface leirásában nem szereplő, de máshonnan ismert rendeltetésü és tartalmu mező lekérdezésével.

  Az ilyen, a rendszer által használt mező igénybevételének hátránya, hogy az operációs rendszer változtatásánál, későbbi kiadásainál a módositás átvitele az uj rendszerbe már nem automatikus ill. szükség lehet a rutin ujrairására is.

= Az operációs rendszer egy részének megváltoztatásával. Nyilván ez a legnehezebbnek látszó feladat, amely komoly előismereteken kivül a rendszer forrásszintü dokumentációját is feltételezi.

Az elvégzendő módositások közül talán a legfontosabb egy, a job-ok futtatás előtti ellenőrzését elvégző rutin felvétele a rendszerbe. A rutinnak meg kell vizsgálnia a job-ban megadott munkaszám, programozó, szervezeti egység, stb. azonositók közül azokat, amelyeknek helyes kitöltését a felhasználói rendszer igényli és azokat a logikai összefüggéseket, amelyeknek ezek között fent kell állniuk.

A hibás job-okat megfelelő hibajelzéssel törölni kell a rendszerből, mintha JCL hibásak lennének.
A megengedett azonositók és összefüggések természetesen időben változhatnak.

A módositásokkal szemben az alábbi követelmények támaszthatók:

= A müködő teljes rendszerben nem okozhatnak észrevehető hatékonyságcsökkenést.

= Teljesen üzembiztosnak, kipróbáltnak kell lennie. Az esetlegesen bennmaradó hibák ugyanis az egész számitógépes rendszert teszik üzemképtelenné, nemcsak egy speciális alkalmazást, ami egyéb programozási hibánál a helyzet.

= Kifejlesztése ne legyen tulzottan gépidőigényes. A mondottakból következik, hogy a rutint kipróbálatlan állapotban csak "üres", igazi feladatokat nem tartalmazó gépben, vagyis normál üzemeltetési környezeten kivüli állapotban szabad futtatni /Blokkidőben/.

= Végül, de nem utolsósorban, a rutin tevékenysége könnyen változtatható legyen. A müködését meghatározó paramétereket file-ba kell tenni; amit a normál üzemeltetési rendszerben is lefuttatható "batch" programmal is javitani lehet. Ezenfelül ajánlatos kidolgozni egy egyszerü és kevés blokkidőt igénylő algoritmust a rutin lecserélésére.

Az utóbbi években /egyéb munkák mellett/ két ilyen módositást végeztem el:

= Honeywell-Bull 66/60-as gépen GCOS operációs rendszerben: Job érvényességi ellenőrzés rendszerben tárolt információk alapján exit rutinnal. Az ellenőrzés algoritmusához nem állt rendelkezésre elég információ a dokumentációban megadott interface-ben, ezért szükséges volt nem dokumentált

mezők /a System Input program/ bizonyos mezőinek kiolvasása ill. munkaterület céljaira való felhasználása.

Rendszer által készitett automatikus feljegyzés módositása az üzemeltetési lehetőségek bővitésére.
/Az elszámolási igények figyelembevételével/.

= R22-n müködő IBM OS/MVT rendszerben

Job érvényességi ellenőrzés saját file-ban tárolt információk alapján exit rutinnal. Az ellenőrzendő összefüggések viszonylagos bonyolultsága és a gazdaságos tárolási mód meglehetősen nagyméretü és komplex algoritmusu rutint tett szükségessé.

Rendszer által készitett automatikus file kibővitése ujtipusu rekordok irásával. Ez a pótlólagos információ lehetővé tette a JCL hibás job-ok észrevételét és elszámolását.

A rutinok az illető rendszer assembly nyelvében iródtak, a HwB rutinnak tárban bármikor áthelyezhetőnek, az IBM rutinnak pedig reentrant szerkezetünek kellett lennie az operációs rendszer jellege miatt.

A munkák elvégzésénél a fő nehézséget a kellő képzettség hiánya jelentette. Hasonló jellegü munkákat a gyártó cégek kihelyezett szakemberei szokták elvégezni, vagy legalább olyanok, akik a megfelelő és normál felhasználóknak nem hozzáférhető tanfolyamokat elvégezték. Ezért a téma dokumentációja nagyon hiányos és kétértelmüségénél fogva több feltételezésnek is teret enged. További nehézséget jelent, ha a munkákat sürgősen vagy a számitóközpont felfutási ideje után kell elvégezni /Ilyenkor kerül sor az éjszakai gépidőkre/.
A munkák során nyert tapasztalatok azt hiszem, nem lesznek meglepőek senki számára sem, inkább csak annak megerősitésére

állnak itt, hogy ezekre tényleg szükség van és sürgősség esetén való elhanyagolásuk valójában késlelteti az eredményt.

= A munkát a lehetőségek és igények alapos felmérésével kell kezdeni /lehetőleg a dokumentáció tanulmányozásán kivül meg kell kérdezni a témában jártasabb, külföldi, szakembert is/.

= Pontosan meg kell tervezni a végrehajtandó funkciókat és ezeket egyeztetni kell a rendszeren végrehajtandó egyéb /pl. üzemeltetési/ változtatásokkal. Az egyeztetés azért szükséges, mert a módositások fizikai helye azonos lehet. Ennek az együttes tervezettségnek a meglévő módositások továbbfejlesztésénél is fenn kell állnia.

= Pontosan meg kell tervezni a módositásokat bevezető és kipróbáló eljárásokat, job-okat. Ezen belül nagyon lényeges olyan eljárások készitése, amelyek a módositásokat a rendszer ujbóli betöltése nélkül az operátori konzolról indithatóan ki-be kapcsolják /bevezetik ill. érvénytelenitik/ lehetőleg még elrontott rendszermüködés mellett is.

= A teljes módositást /rutint/ kipróbálás céljából két részre kell választani. Az egyik rész tartalmazza a müködő algoritmus javarészét és egy szimulált rendszer környezetben futtatható normál job-ként, normál batch üzemeltetési feltételek mellett. Csak az interface tesztjét, majd a véglegesen összeállitott rendszert /az utóbbit optimális esetben csak egyszer/ kell próbálni "éjjeli" tömbidőben. A kipróbáláshoz saját /az üzemeltetéstől független/ példányra van szükség az operációs rendszerből.

A csak algoritmust tartalmazó batch rész a belövés után is hasznos lehet az üzemszerü használat közben talált észrevételek gyors kivizsgálásához.

= A rendszer üzemszerü bevezetése után az időről-időre szükségessé váló módositásokat /a paraméterfile karbantartását/ ciklikusan és rendszeresen kell végezni egyéb rendszerkarbantartási tevékenységekkel összhangban.

Hasonló jellegü munkák közül jelenleg tervbe van véve a HASP-al müködő MVT rendszer módositása. Itt a job ellenőrzés várhatólag nem okoz különösebb problémát /MVT-hez képest/ viszont más szempontok /HASP szintaxis, elszámolási információk teljessége és HASP nélküli MVT-vel való kompatibilitás elszámolási szempontból stb./ várhatólag szükségessé tesznek forrásprogram szintü javitásokat HASP-ban.

A VT54 ÜGYVITELI KISSZÁMITÓGÉP DISZKES FILE-KEZELŐ RENDSZERE

Szász Eszter
VIDEOTON Fejlesztési Intézet

A VT54 ügyviteli kisszámitógép a VIDEOTON 50 ügyviteli gépcsalád diszkes tagja.

A VT54 8 Kszó memóriával rendelkezik. Ez az alap, amely 12 Kszó-ra bővithető.

Perifériák:

- VT 340 alfanumerikus display
- ügyviteli konzol / numerikus, alfanumerikus és vezérlő-billentyüzettel, kijelző lámpasorral/,
- DZM 180 mozaiknyomtató
- 2 db kazettaegység,
- lyukszalagolvasó /opciomális/
- 1-4 ISOT 137o diszkegység, egységenként egy fix és egy cserélhető lemezzel / kapacitásuk 2,5 - 2,5 Mbyte /.

A diszk fizikai felépitésére itt nem térek ki, hiszen a file-kezelő rendszer által létrehozott struktura szinte független ettől.

A fizikai-logikai kapcsolatot handler biztositja.

A diszkes file-kezelő rendszer a lemezeket , mint egy "nagy memóriát " kezeli - handlerén keresztül a lemezek szavanként hozzáférhetőek.

Logikai egységek, a kötet fogalma

A file-kezelő rendszer a fizikai meghajtóegységek helyett logikai egységeket kezel, a következő módon:

- a fix lemezeket egyetlen kötetként kezeli, közös katalógussal a 0. diszkegység fix lemezén,
- a cserélhető lemezek mindegyike lehet önálló logikai egység, saját katalógussal,

  vagy
- több cserélhető lemez láncolható egyetlen kötetté. Az így értelmezett kötet közös katalógussal rendelkezik a 0. logikai kötetsorszámu lemezen. A folytatáskötetek mindegyike egy-egy kötetcímkét tartalmaz, amelyek alapján a logikai sorrend meghatározható.

A diszk strukturája

KATALÓGUS LEMEZ

BOOT

VOLUME LABEL

LOADER

LABEL1

LABEL2

A FILE-OK

TERÜLETE

FOLYTATÁSKÖTET

VOLUME LABEL

A FILE-OK

TERÜLETE

Ahol:

VOLUME LABEL

A lemez azonositására szolgál

HEADER

A katalógus aktuális állapotát és a diszk foglaltságára jellemző.adatokat tartalmazza.

Itt helyezkedik el a diszktérkép - egy olyan bit táblázat, amelynek minden bit-je a diszk egy-egy területének / granulájának :2048 byte/ felel meg. A bit értéke 1, ha a megfelelő terület foglalt, 0 ha szabad. A file-kezelő rendszer ezen diszktérkép alapján dolgozik, osztja ki a file-ok számára a diszk területetket.

LABEL1

A diszkköteten található file-ok nevének és tulajdonos ·· nevének gyüjteménye.

LABEL2

A file-ok jellemzőit tartalmazó táblázatok /Data SEt LABEL/ összessége.

A katalógus maximum a diszkkötet 25%-át foglalja le.

A file-ok elhelyezése

A file-ok helyének kiutalása dinamikusan történik.

Létrehozandó, vagy bővitendő file esetén az OPEN modul levizsgálja a HEADER-ben található diszk térképet, és ennek alapján kiutalja a file számára a kért nagyságu területet.

A file-ok törlése következtében felszabadult diszk területek is kiosztásra kerülnek.

A helykiutalás egysége a granula.

1 granula a file számára kiutalható minimális diszk terület. Az első változatban 8 szektornak megfelelő 2048 byte.

Ha a felhasználó nem tudja előre megbecsülni a file vagy a bővités előrelátható hosszát -és ezt jelzi az FD /file leiró/ táblában, a rendszer granulánként folyamatosan osztja ki számára a diszkterületeket.

A file-ok a diszken nem feltétlenül összefüggő területen helyezkednek el. Egy - egy összefüggő diszk terület kezdő és végcimét a rendszer a file katalógus elemében /Data Set Label/ tárolja.

A daraboltság maximális mértéke 10.
A file daraboltságát a felhasználó nem érzékeli, a file kezelő rendszer kezeli le minden file tipusnál.
Utility program gondoskodik a bővitések folyamán esetleg tuldaraboltá vált file egybeválogatásáról, ill. ha szükséges egy teljes diszk kötet egybeválogatásáról is /REGEN/.

A file-ok tipusai

Szekvenciális file

A szekvenciális file logikai rekordok sorozata, amelyek hossza fix vagy változo.

Blokkositatlan

A rekordok a file-ban,a diszken , folyamatosan - azaz hézagmentesen- helyezkednek el a számukra kiutalt részterületeken.

A rekordformátum fix vagy változó.

Minden rekordhoz való hozzáférés egy diszkmüveletet igényel.

Az ilyen file tipikusan ügyviteli feldolgozás alkalmával használható. Ebben az esetben a gyakori diszk hozzáférések ideje elhanyagolható az operátor klaviaturán végzett müveletvégzési idejéhez képest.

Blokkositott

A rekordok blokkositása a feldolgozás gyorsitását eredményezi.

Előkészitett adatmennyiségek gyors feldolgozását teszi lehetővé.

Egy blokk a felhasználó által definiált számu rekordot tartalmaz. Egy diszkátvitel egy blokk átvitelét jelenti. Amig a blokk betelik a müveletvégzés a memóriában történik.

Formatum nélküli file

A formatum nélküli file egy logikailag összefüggő terület a diszken. A felhasználó ezen a területen file relativ címekkel dolgozhat.

Ez a file tipus elsősorban fejlesztők számára készült, igy nyujtva nekik egy "kis diszket" , azaz egy olyan területet, amelyen tetszőleges strukturát alakithatnak ki a file-kezelő rendszer felügyelete alatt. Igy nem áll fenn annak a veszélye, hogy a diszk file-kezelő rendszer által kialakitott és felhasznált felépitését akarva -akaratlanul bárki tönkretegye.

Indexelt szekvenciális file

A file-kezelő rendszer a file-okat egységesen kezeli.

Minden file indexelt szekvenciálisnak is tekinthető /kivéve a formátum nélküli file-t/, amennyiben tartozik hozzá egy un. index file.

Az index file egy tetszőleges adatfile-hoz tartozó, rendező programmal létrehozott formatum nélküli file, amely az adat-file rekord-kulcsait tartalmazza a hozzátartozó adat-file relativ címekkel , megfelelő / növekvő vagy csökkenő/ sorrendben; valamint egy összefoglaló táblázatot- amely megkönnyiti a file-kezelő rendszer számára az adott kulcsu rekord kikeresését.

Az index-file egy tulcsorduiasi területtel is rendelkezik. Ezt a területet a file-kezelő rendszer az adatfile bővitésekor tölti ki. Ide kerülnek az adatfile bővitése során a diszkre került adatrekordokból kiválogatott kulcsok, a hozzájuk tartozó file-relativ cimekkel együtt.

Az index-file-t az INDEX utility hozza létre ill. rendezi ujra , ha szükséges.

Az"index" utility elvégzi az adatfile adatállományának a lerendezését is - a felhasználó kérésére-.
Ebben az esetben a file szekvenciális lekérdezésével történhet az adatfile indexsoros feldolgozása.

A file-ok feldolgozása

- File létrehozása
- File lekérdezése
- File bővitése
- File lekérdezése és bővitése
- File lekérdezése, rekordok helyettesitése és a file bővitése

A file rekordjainak helyettesitése egy-egy GET-PUT szekvenciát jelent - tehát a helyettesitendő rekordot előbb be kell olvasni.

A diszkes file-kezelő rendszer munkapufferként a felhasználó FD /file-leiró tábla/ tábláját is felhasználja. /A memória 8 Kszó!/

A diszkes file-kezelő rendszer beilleszkedése a VT54 lapsoftware-be.

A szekvenciális file-ok feldolgozását végző OPEN, GET, PUT és CLOSE modulok monitormodulok. Az OPEN és a CLOSE overlay modul, a GET és a PUT pedig rezidens.

Az indexelt szekvenciális file-ok feldolgozása több lépcsőben történik:

- Az index file létrehozása az INDEX utility programmal,
- IS:OPN, IS:GET, IS:PUT, IS:CLO file-kezelő modulok működtetése.

Az "indexelt szekvenciális" OPEN, GET, PUT és CLOSE egy - egy forrásnyelvi szubrutin, amelyek a felhasználói programhoz fordithatóak, és jelentősen épitenek a monitorba épitett rokon modulokra.

Ez a file-kezelő rendszer CDC gyártmányu diszkeken is alkalmazható / amelyek felépitése hasonló az ISOT 1370-hez, kapacitása 5-5Mbyte/.

Utility programok

KATINI

A lemezek inicializálására szolgál

LISTD

A kötetcimke, a katalógus ill. a file-ok listázását végzi.

RENAME

Fileok nevének megváltoztatását teszi lehetővé

DELETE

A file-ok törlését, és egyidejüleg a katalógus tömörítését végző program

REGEN

Diszkes file-ok másolását , a file-ok daraboltságának megszüntetését ill. diszk kötetek rendezését teszi lehetővé.

COPY

File-ok másolására szolgáló program: diszkes file másolása kazettára, és forditva.

INDEX

R endező program.

Adatfile-okhoz "index file"-okat készit.

A fent vázlatosan ismertetett diszkes file-kezelő rendszer az ügyviteli adatfeldolgozás követelményeit kielégiti , sőt lehetőséget teremt arra is, hogy RPG II. fordító program müködjön a VT 54-en.

Igy a VT 54 ügyviteli kisszámitógép egy kimondottan adatfeldolgozásra készül magasszintü nyelven is programozható.

AZ R-10 ÉS CII-10010 /VT-1010/B/ SZÁMITÓGÉPEK ÖSSZEKAPCSOLÁSA

Szeľényi László
Szegedi Orvostudományi Egyetem
Számitástechnikai Központ

Dr.Sára Attila
József Attila Tudományegyetem

1./ Előzmények:

A Szegedi Orvostudományi Egyetem Számitástechnikai Központja jelenleg két számitógépet üzemeltet, mégpedig az 1973. januárjában beállitott CII-10010 /VT-1010/B/ és az 1976. szeptemberében átadott R-10 kis-számitógépeket.

A két kis-számitógép on-line összekapcsolásának gondolata 1977. elején merült fel a következők miatt:

- A CII-10010 számitógéppel 1973. óta on-line mérésadatgyüjtés folyik /EEG, EKG/ a hozzáillesztett analóg-digitál konverter segitségével. A gép kicsiny belső memória /16 Kbyte/ és háttértároló kapacitása /800 Kbyte/ valamint szükös utasitásrendszere miatt a nagytömegü mérési eredmény tartós tárolása és gyors feldolgozása /Fourier-transzformált/ szinte megoldhatatlan feladatot jelentett. Az R-10-es és a CII-10010 on-line gépkapcsolat megvalósitása esetén a feladat megosztható a két gép között:
  A közvetlenül az adatgyüjtéssel kapcsolatos szervezési és ideiglenesen szükséges kistömegü adattárolási feladatokat továbbra is a CII gép végzi.

Az adatoknak a két gép kapcsolatát biztositó on-line vonalon történő R-10-be küldése után az adatok gyors feldolgozását és nagytömegü tárolását az R-10 számitógép végezheti, mely utasitás-készletét és tárolókapacitását /5 Mbyte-IZOT diszk, mágnesszalag-egységek/, valamint software-kiépitettségét tekintve lényegesen alkalmasabb e feladat ellátására, mint a CII számitógép.

- Az összekapcsolás másik - nem kevésbé fontos - célja az, hogy a CII számitógép hardware erőforrásait /lyukszalagállomás, sornyomtató, minidiszk, teletype, analóg-digitál konverter/ software eszközök közbeiktatásval az R-10 számitógép saját erőforrásaiként használhassa. Ezáltal az R-10 számitógép periféria-hibák miatti állásideje csökkenthető, másrészt az egyes perifériák megduplázódását követően /lyukszalagállomás, sornyomtató/ nagyobb input-output teljesitmény-igényü feladatok egymással multiprogramozhatók.

A megvalósitandó on-line kapcsolattal szemben a következő hardware-software alapkövetelményeket állitottuk:

- Memóriából memóriába történő átvitelt biztositson megadott byte-szám szerint, kódfüggetlenül.
- Mindegyik gép a másikat programozott perifériaként kezelje hasonlóan a többi programozott perifériájához, real-time rendszerben.

Ezek figyelembevételével készült el a két gép közötti on-line vonal hardware 1977. végére.

2./ A megvalósitott hardware

A számitógépek összekapcsolásának alapvető célja a hardware és software erőforrások megosztott használata.

Egymáshoz közel telepitett gépek esetén nincs szükség postai kommunikációs vonalak igénybevételére, amelyek jelentősen korlátoznák a gépek közötti adatátviteli sebességet, és nincs szükség a bit-soros átviteli rendszerre sem. Ezért egyedi, nagy sebességü kábel segitségével 16 bit szélességü átviteli utat létesitettünk a két gép csatornái között. A csatornákra kapcsolódó saját fejlesztésü egység az un. csatorna-csatorna adapter egy adat-byte és mellette egy állapot-byte párhuzamos átvitelét teszi lehetővé a két gép között egy-egy müködési ciklusban. A CCA az R-10 oldalról egy EP-15-2 tipusú real-time periféria-csatolón keresztül programozott I/O perifériaként kezelhető; a CII oldalon a CCA szintén a programozott csatornára kapcsolódik, mint egy hex. 42 cimü input és egy hex. 43 cimü output periféria.

Bármelyik gép küldhet ki a CCA-ra I/O parancsokat és egyúttal olvashat be, illetve vihet ki adatokat, amelyek a másik gépnél megszakitást váltanak ki.

A megszakitás kezelése során a másik gép a megfelelő I/O utasitásokkal tölti vagy üriti a CCA adat-pufferét, amely tevékenység viszont az első gépnél vált ki megszakitást. Ezt a "hand-shaking" folyamatot nevezzük a CCA egy mükö-dési ciklusának.

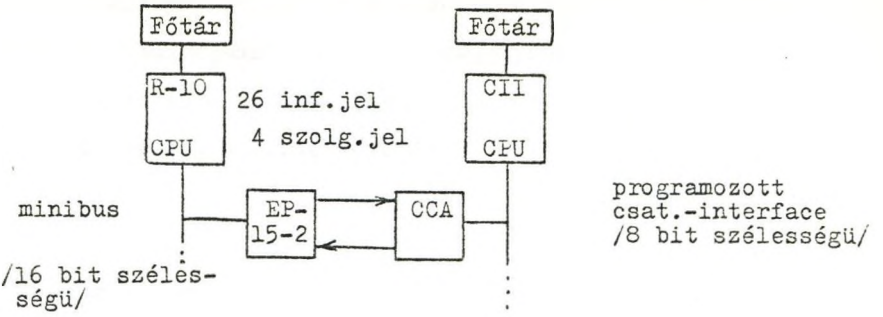
Az adatátvitel sebességét alapvetően befolyásolja a meg-szakitó rutinok megirási módja. Ezt figyelembe véve, a tesztekkel mérhető maximális adatátviteli sebesség a két gép operativ memóriája között 5 Kbyte/sec.

Érdekességképpen bemutatjuk a gép-gép kapcsolat rendszer-technikai blokkvázlatát /1.ábra/, valamint a CCA állapot- és adatregisztere bitjeinek jelentését, amely jól érzékel-teti a CCA fő üzemállapotait /2.ábra/.

3./ A csatorna-csatorna adapter software

A bevezetésben emlitett - a két gép összekapcsolásának gondolatát, megvalósitását kiváltó - kettős célnak meg-felelően a CCA software is kettős tagozódást mutat, me-lyek hierarchikusan egymásra épülnek. A két gép hard-ware-software szervezettségét és lehetőségeit tekintve mind a két-szintü software tervezésénél a következőket tekintettük alapelvnek:

- - A gépkapcsolat vezérgépe mindig az R-10 gép, azaz az átvitelek logikai kezdeményezője kizárólag az R-10 a szokásos CSV M:IO SV-hivással.



1. ábra: Az R-10 CII-10010 összekapcsolás blokkvázlata

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.
OC	IC	EC	SC	OB	IB	DIT	CIT	D0	D1	D2	D3	D4	D5	D6	D7

OC=1, ha az R-10 által kezdeményezett output parancs van érvényben

IC=1, ha az R-10 által kezdeményezett input parancs van érvényben

EC=1, ha a CII által kezdeményezett input parancs van érvényben

SC=1, ha a CII által kezdeményezett output parancs van érvényben

OB=1, ha az R-10 oldali output-puffer feltöltött állapotban van, /R-10 töltötte fel, és CII nem üritette/

IB=1, ha az R-10 oldali input-puffer feltöltött állapotban volt, /CII töltöttel fel, és R-10 éppen üritette/

DIT=1, ha a CII oldalon a CCA-nak várakozó megszakitás kérése van,

CIT=1, ha a CII oldalon éppen a CCA megszakitás-kezelése folyik,

D0-D7 a beolvasott adat-byte bitjei,

2. ábra: A CCA-állapot- és adatregiszter bitjeinek jelentése

/Bár a megvalósitott hardware a két gépet fizikai kezdeményezés szempontjából egyenrangúnak tekinti./

- Minden átvitel a két gép között párbeszédes formában zajlik le.

- A tényleges adatátvitel fizikai kezdeményezője, időzitője, végrehajtója a CII-gép.

- Az átvitel végének ellenőrzője az R-10-gép, mely az átvitel ellenőrzésének eredményéről tájékoztatja a CII-gépet.

3.1. CCA-SOFTWARE-1 rendszer /alacsonyabb szintü megvalósitott rendszer/ a mérés-adatgyüjtéssel kapcsolatos probléma gyors megoldására készült. Két fő részből, egy R-10-es CCA handlerből, és egy CII-10010-ben müködő IT-2-es megszakitásfelismerő és CCA-kezelő programból áll.

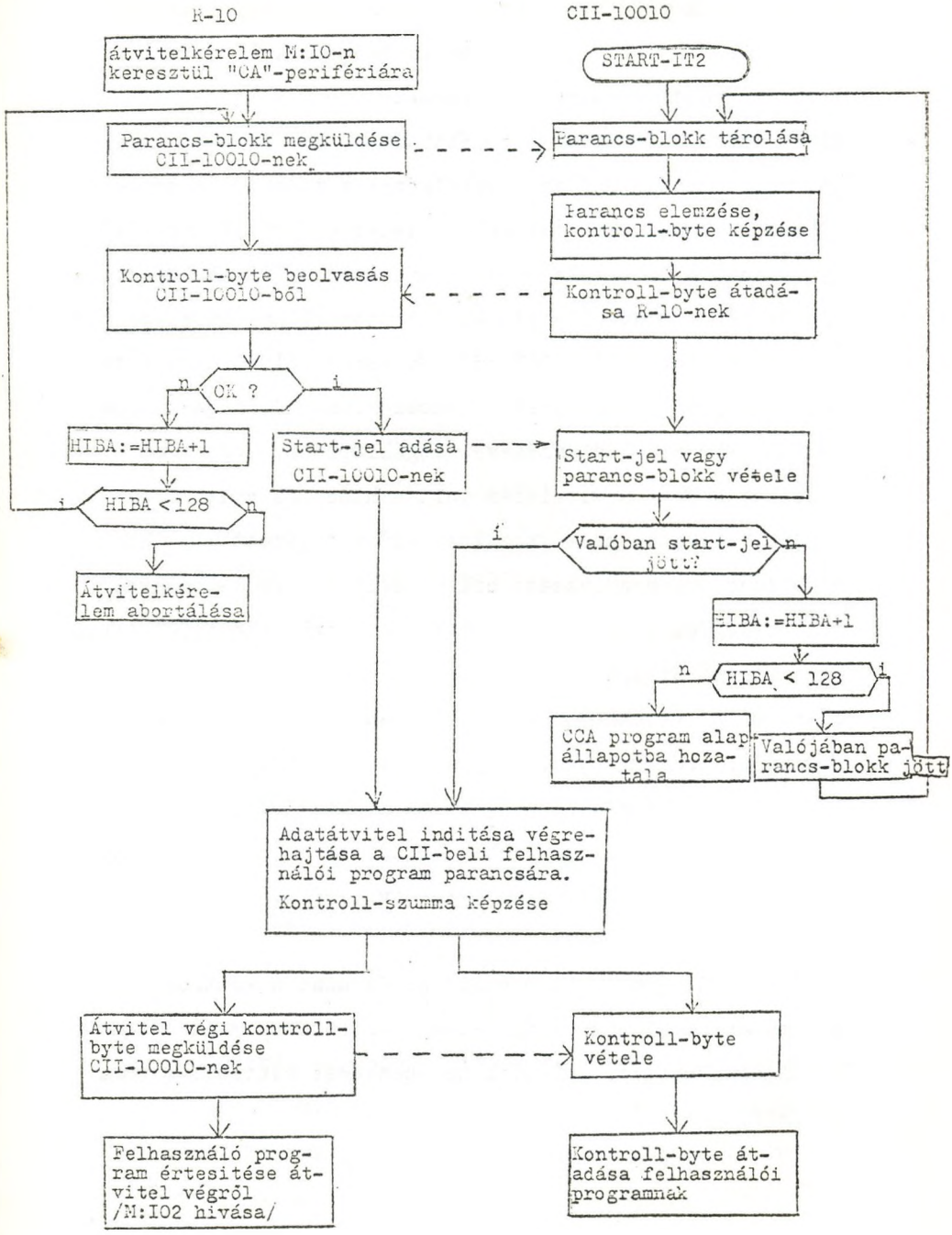
Ez a két program biztositja, hogy két együttmüködő felhasználói program, melyek egyike az R-10-ben, másik pedig a CII-ben müködik, adatátvitelt hajtson végre a két gép között tetszőleges irányban. A CII oldali részrendszer ellátja még az IT-2-es megszakitási szintre kapcsolt perifériák /teletype, lyukszalagállomás, analóg-digitál konverter, minidiszk/ megszakitásainak felismerését-kezelését. Mind a két gép esetében a CCA-periféria real-time rendszerben müködik a többi perifériával.

Az R-10 oldali CCA-handler nem része a monitornak, mivel állandó jelenlétére nincs szükség /jól elhatárolható időpontok között folyik mérésadatgyüjtés/. A monitorba a generálás során csak egy nagyon rövid /10 utasitásból álló/ úgynevezett pre-handler került beszerkesztésre, abból a célból, hogy a rendszertáblázatokban helyet foglaljon a valódi CCA-handler rendszerelemeinek. Amig a pre-handler van rákapcsolva a monitorra, minden CCA-ra kiadott I/O parancs "logikai hibával" visszautasitásra kerül. A valódi CCA-handler az EP-könyvtárból egyszerü operátori parancssorozattal tölthető a memóriába és kapcsolható a monitorra a pre-handler helyére.

A CII oldali CCA-handler és az azt használó felhasználói program láncolt betöltéssel tölthető-inditható a CII-diszkes rendszer bináris könyvtárából.

3.2. Az átvitel logikai felépitése

A 3. számu ábrán követhető az adatátvitel párbeszédes formában szervezett folyamata:


R-10
CII-10010
átvitelkérelem M:IO-n keresztül "CA"-perifériára
START-IT2
Parancs-blokk megküldése CII-10010-nek
Parancs-blokk tárolása
Parancs elemzése, kontroll-byte képzése
Kontroll-byte beolvasás CII-10010-ből
Kontroll-byte átadása R-10-nek
n
OK ?
i
HIBA:=HIBA+1
Start-jel adása CII-10010-nek
Start-jel vagy parancs-blokk vétele
i
HIBA <128
n
Átvitelkérelem abortálása
i
Valóban start-jel jött?
n
HIBA:=HIBA+1
n
HIBA < 128
i
CCA program alapállapotba hozatala
Valójában parancs-blokk jött
Adatátvitel inditása végrehajtása a CII-beli felhasználói program parancsára.
Kontroll-szumma képzése
Átvitel végi kontroll-byte megküldése CII-10010-nek
Kontroll-byte vétele
Felhasználó program értesitése átvitel végről /M:I02 hivása/
Kontroll-byte átadása felhasználói programnak

3. ábra

Az átvitelek mind két irányu logikai kezdeményezése az R-10 M:IO SV-szekcióján keresztül történik. Miután az R-10 CCA-handler elemezte a parancsot egy 6 byte-ból álló "parancs-blokk"-ot /továbbiakban: PB/ ir át a CII-be, mely tartalmazza az átvitelkérelem irányát, a byte-számot, továbbá a PB helyes átvitelét ellenőrző kontroll-byte-okat. A PB vétele és elemzése után a CII kialakit egy "parancs kontroll-byte"-ot /továbbiakban: PKB/, melyet az R-10 visszaolvas a CII-ből. Ezután a CII várakozik az R-10-től jövő "start-jel" /továbbiakban: SJ/ megérkezésére. Az R-10 a PKB elemzésével megállapitja, hogy a PB /esetleg a PKB/ átvitelében volt-e hiba. Ha volt, legfeljebb 128-szor próbálkozik az előbb emlitett PB, PKB /megküldés, visszaolvasás/ ciklus helyes végrehajtásával, majd sikertelen esetben abortálja az átvitelkérelmet vonalhiba miatt. Sikeres esetben az R-10 megküldi CII-nek a SJ-t. Az SJ vétele után a CII-CCA -handler jelzi a felhasználói programjának, hogy a csatorna átvitelre kész. A CII-beli felhasználó program az átvitel CII-beli memóriacimének megadásával egyidőben elinditja az átvitelt, melyet fenntart a szükséges számu byte átviteléig.

Az átvitel végén az R-10 ellenőrzi az adatok kontroll-szummáját, majd egy átvitel-végi kontroll-byte-ot küld a CII-nek, mellyel értesiti az esetleges kontroll-szumma hibáról.

lgy az esetleges adatátviteli hibáról mindkét gép felhasználói programja értesül.

3.3. CCA-SOFTWARE-2 rendszer /a közlemény megirása idején tervezés alatt/-a bevezetőben emlitett második problémát hivatott megoldani, nevezetesen azt, hogy az R-10 gép felhasználói programjai:a CII-perifériákat, mint R-10 perifériákat használhassák. A két gép közötti átvitel elve megegyezik a 3.2. pontban elmondottakkal.

A CII-perifériák R-10 perifériaként való használatát a következő elven kivánjuk megoldani:

- Az R-10 oldalon definiált "CA" perifériát multiperiférias egységnek tekintjük, és az ASSGN operátori parancsok esetén megengedjük, és a R-10-CCA-handlerben elemezzük az $\left\{\begin{matrix}M\\U\end{matrix}\right\}$:XX,T:CA,D:Y, $\left\{\begin{matrix}AN\\BN\end{matrix}\right\}$ tipusú hozzárendeléseket.

  Jelentse Y = 0, 1, 2, ..., 7 rendre a CII egyes perifériáit valamilyen sorrendben.

- Az R-10 által CII-nek küldött PB-t kiegészitjük a fenti Y értékkel, továbbá az átvitelkérelem $\left\{\begin{matrix}AN\\BN\end{matrix}\right\}$ jellemzőjével /a PB továbbra is 6 byte, mivel a CCA-SOFT-1-ben nincs kihasználva mindegyik byte/.

- A CII számitógépben egy speciális felhasználói program gondoskodik az R-10-től a CCA-n keresztül kapott parancs végrehajtásáról, a szükséges periféria inditásáról, müködtetéséről.

Végezetül megemlitjük, bár az átviteli rendszer tartalmaz hibaelemzéseket az átviteli hibák viszonylag pontos feldieritésére /különösen a PB-PKB-SJ ciklusban/, ezekre szándékosan nem tértünk ki, mivel nem a nagytávolságu soros adatátvitelekre jellemző véletlenszerü egyedi hibák fellépése a várható, /kis távolság, párhuzamos átvitel/ hanem standard vonalhibák jelentkezése, melyek könnyebben felfedhetők /pl: fix "bit vesztés" vagy "bit nyerés" valamely adatvonal/ak/on/.

AZ ANSWER DOKUMENTÁCIÓS RENDSZERE

Szőke Péter
Számitógépalkalmazási Kutató Intézet

Az előadásban ismertetésre kerülő rendszer a SzÁMKI-ban fejlesztés alatt álló ANSWER rendszer része. Az ANSWER célja az, hogy hatékony segitséget nyújtson programok /programtermékek/ készitéséhez.

Felfogásunk szerint

programtermék = programok + dokumentáció,

ezért az ANSWER a konkrét programok fejlesztésén túl segitséget nyújt a dokumentáció elkészitéséhez is.

A programtermék részét képező dokumentáció nem az egyetlen természetes nyelvü információ, mely a programtermék kifejlesztése során létrejön. Ilyenek a különféle szintü tervek, beszámolók, szerződések, előadás-vázlatok és szövegek, stb. A jelenlegi pr. fejlesztési módszer szerint ezen információknak van egy olyan része, mely csak szóban él /pl. egyes programozási konvenciók, munkamegosztás, a vezető ismerete a munka pillanatnyi állásáról, stb./. Amig az ilyen információ nincs lerögzitve irásos, és bármely kompetens személy számára szabadon hozzáférhető formában, mindig fennáll annak a veszélye, hogy az információ egy része feledésbe merül, illetve az a személy, akinek egy bizonyos döntést meg kell hoznia, nem tudja áttekinteni az összes aspektusokat.

A dokumentum-kezelő rendszer /ADR/ az ilyen információk kezelését megkönnyitő információs rendszer egy részét képezi:

ANSWER = Nyelvi rendszer + Információs rendszer

A rendszer elsősorban az alábbi szolgáltatásokat nyújtja:

- Alapismeret jellegü, tématól független információk gyors elérése: ha egy programot fejlesztő személynek bármilyen problé-

mája merül fel, utánanézhet a megoldásnak úgy, ahogy azt egy könyvtárban, illetve levéltárban is megtehetné /pl. programozási nyelv és operációs rendszer tulajdonságai, szabványok,stb./

- Programfejlesztés során felmerülő döntések, javaslatok, utasitások, ütemezések, ötletek, elhalasztott, de később megfontolandó döntések stb. tárolása; a "szemét" kiszürése, szelektiv információ-szolgáltatás a szakmai és adminisztrativ vezetés, tervezés, megvalósitás, tesztelés, karbantartás számára.

- Előzetes dokumentumok /rendszertervek, vázlatok, stb./, végleges dokumentumok /fejlesztői és felhasználói kézikönyvek/, és egyéb természetes nyelvü irásos anyagok elkészitésének és módositásának támogatása, tárolása, javitása.

Az ADR tehát tulajdonképpen két fő részből áll; ezek egyrészt a tárolt információk, másrészt az információk használatát biztositó kezelőrendszer.

ADR = adatbázis + kezelőrendszer.

Az ADR-rel szemben támasztott alapvető követelmények:

- interaktiv használat;
- természetes nyelvü szövegek kezelését "gépesiti" /nem automatizálja!!!/;
- autodidaktiv;
- a papirmunkát helyettesiti,
- könnyü kezelhetőség;
- nem teljes rendszernek is müködnie kell /az ADR fejlesztése lépcsőzetes/;
- szimultán, sokoldalú, szelektiv, gyors hozzáférést biztosit a tárolt információkhoz;
- biztonságos /a benne tárolt információ szándékos vagy véletlen elrontásának az esélye nagyon kicsi legyen/.

Az adatbázis

Alapstruktúra

Az adatbázis olvasótermi és alkotóházi /working/ könyvtárakból áll. Ezeket egészitik ki az esetleg meglévő archiv könyvtárak. Az egyes könyvtárak könyvekből, a könyvek kötetekből állnak. Minden kötetnek van egy "irott" és egy "tördelési" strukturája.

A tördelési struktúra szerint a kötet oldalakra, azok sorokra, a sorok pedig karakterekre oszlanak.

Irott strukturája szerint egy kötet fejezetekre, alfejezetekre, paragrafusokra, pontokra oszlik /ezeket mind a továbbiakban egy szóval pontoknak fogjuk nevezni/. Az olyan pontok, melyek nem oszlanak további pontokra, bekezdésekből, azok pedig mondatokból állnak. A mondat irott részei a szavak, melyeket karakterek alkotnak.

Az irott és a tördelési struktúra az alábbi szinteken érintkezik:
- mindkét struktúra legkisebb, bonthatatlan eleme a karakter;
- a bekezdések új sorban kezdődnek;
- minden kötet új oldalon kezdődik.

A fenti struktúrában bevezetett fogalmakat közösen szövegegységeknek nevezzük. Általában két szövegegységet egy jól meghatározott terminátor választ el egymástól.

Speciális szövegegységek

Az előző pontban definiált szövegegységek képezik az adatbázis központi részét, amelyeket többféle speciális szövegegység egészit ki:
- Könyvtárak, könyvek, kötetek és további pontokra bomló pontok katalógussal, ill. tartalomjegyzékkel rendelkeznek. /A "katalógus" és "tartalomjegyzék" fogalmak szinonimák./;
- a könyveknek és köteteknek cimlapjuk van;
- minden könyvhöz
  rövidítésszótár, szövegmakró szótár, parancsmakró-szótár tartozhat;
- minden kötethez tárgymutató tartozik.

További struktúra

Az adatbázis egyes részeihez bizonyos tulajdonságok rendelhetők hosszabb-rövidebb időre, vagy akár állandóra is. Ezek egyrészt a
- minősitések, melyek lehetővé teszik a szelektiv olvasást.
  A minősitéseknek több osztálya van; minden osztály egy, a rendszerre egységesen definiált értékhalmazzal rendelkezik /különböző osztályok értékhalmazai diszjunktak/;

- a "cimkék", melyekkel szövegrészek jelölhetők meg egy megadott időtartamra; a megjelölt szövegrészekre a legegyszerübben a cimkék megadásával hivatkozhatunk;
- markerek. Ezek szabványos nevü, ideiglenes cimkék.

Az ADR használata

Az ADR olvasótermi könyvtára tartalmaz előre megirt kézikönyveket. Ezekben olyan információ található, melyre az ANSWER rendszer felhasználójának szüksége lehet /alapismeretek, konvenciók, szabványok, definiciók stb./. Ezenkivül itt tárolódnak a gazdasági vezetés számára szükséges információk, és kisebb körök számára hasznos ismereteket tartalmazó témakönyvek. Ezek a felsőbbszintü irányitás, a szakmai irányitás és a kivitelezés információit, továbbá a témával kapcsolatban kiadásra szánt dokumentumokat tartalmazzák.

Az alkotóházi könyvtár tartalmazza az olvasótermi könyvek piszkozatait és az olyan irott anyagokat, melyek elkészültük után a rendszerből kikerülnek.

A két könyvtár együtt használható: az olvasótermi könyvtár könyveit csak olvasni lehet, ill. a könyveket egészben vagy azok részeit ki lehet másolni vagy ki lehet listáztatni. Bármilyen módositás csak az alkotóházi könyvtárban végezhető el. Az olvasótermi könyvtár köteteit csak a könyvtáros törölheti vagy cserélheti újabb verzióra.

Egy irásmü rendszerbe történő bevitele vagy kiadásra történő előkészitése az alábbi módon történik:
Ha a szerző egy egészen új anyagot készit, akkor generál egy üres könyvet vagy kötetet. A generálás során a rendszer először kitölteti a cimlap bejegyzéseit, majd a tartalomjegyzéket, végül elkezdődhet a szöveg irasa. A tartalomjegyzek kitöltése egyenlőre le is maradhat. Ha csak egyszerü, rövid irást akar valaki készitени, akkor a generálást úgy is végezheti, hogy egy szabványos munka-kötet jön létre. Ilyenkor a felhasználónak a lehető legkevesebb előkészitő müveletet kell végeznie.

Ha egy már meglévő kész irásmüvet akar valaki átdolgozni, akkor /generálás helyett/ először is készittet a rendszerrel egy nem kész-nek nyilvánitott másolatot. Az ilyen változathoz egyedül az a felhasználó tud hozzáférni, aki létrehozatta.

Arra is van lehetőség, hogy egy már meglévő könyv tartalomjegyzékét vagy egy, a szabványok között tárolt tartalomjegyzéket másoljunk le új,"nem kész" könyvbe. /Pl. abban az esetben, ha szabvány irja le, mit kell tartalmaznia a dokumentumnak./

1. Könyvek, kötetek létrehozására, meglévő könyvek adatainak változtatására és könyvtárak közti mozgatására szolgálnak a GET, PUT, CREATE, ERASE, ARCHIVE, REARCHIVE és RENAME parancsok.

   A PUT parancs szolgál könyveknek az olvasótermi könyvtárba történő bevitelére. Mivel az alkotóházi könyvekkel szemben támasztott követelmények nem olyan szigorúak, mint az olvasóterem előirásai, ezért az átvitel során ellenőrzések és bizonyos módositások is megtörténnek. Problémás esetekben a rendszer párbeszédet kezd a parancs kiadójával a módositással kapcsolatban.

2. Egy kötet irása-olvasása közben lehetséges az irásmüben oda-vissza mozogni, azaz bármikor megjelenithető képernyőn a szöveg tetszőleges része. Arra is lehetőség van, hogy eközben más kötetekbe is betekinthessünk, ill. "könyvjelzőt" tehessünk a kötet több részére is.

   A távolabbi szövegrészek közti kényelmes közlekedést szolgálja a képernyőn látható "pozició-lista". Itt több olyan szövegrész koordinátái találhatók, amelyeket időnként meg akarunk tekinteni. A szövegrész megtekintéséhez elegendő a pozició-lista megfelelő elemére állitani a kuzort és kiadni a READ parancsot. A látott szövegrész környezetében ROLL és PAGE parancsokkal mozoghatunk /utóbbi kiadásakor pl. a következő egy képernyőnyi szövegrész jelenik meg a display-en/. NEXT utasitással az irott és a tördelési struktúra egy adott szintjén szövegegységenként ugrálhatunk a szövegben.

   Ha a READ utasitást úgy adjuk ki, hogy a kurzor egy tartalomjegyzéki bejegyzésre mutat, vagy pedig a tárgymutató valamely szavának egy adott előfordulásának koordinátájára /mely szin-

tén tárolódik a tárgymutatóban/, akkor a könyv /kötet/ a megjelölt helyen "kinyilik", tehát a jelzett fejezet, vagy a kulcsszó adott környezete a képernyőn olvashatóvá válik.
Ezek a hozzáférési módok PRINT utasitással is használhatók; ilyenkor az operandussal kijelölt szövegrész /a kurzor által mutatott pozició is lehet operandus/ kilistázódik.
Ha valaki a könyvet /kötetet/ módositani is akarja, akkor előre "levédheti" az OPEN utasitással. Ennek hatására a rendszer tudomásul veszi, hogy az adott könyv /kötet/ módositás alatt áll, és azt más számára hozzáférhetetlenné teszi. Ennek forditottja a CLOSE.

Az irott struktúra szintjei között UP és DOWN utasitásokkal lehetséges a mozgás.
A parancsmező és a szövegmező közötti kurzor-ugratásra szolgálnak a TEXT és a COMMAND utasitások.

A fentieket összefoglalva, az olvasás-nyomtatás utasitásai a következők: OPEN, CLOSE, READ, PRINT, ROLL, PAGE, UP, NEXT, COMMAND, TEXT.

Az ADR használata közben a display-nek csak egy részét foglalja el a szöveg /a nagyobbik részét/; néhány sor az utasitásoknak, ill. a pozicósornak és a rendszer-azonositónak, dátumnak, stb. van fenntartva. Az egy billentyü lenyomásával nem realizálható utasitásokat a parancsmezőnek nevezett részbe gépelhetjük be.

3. A módositás parancsai: INSERT, DELETE, EXCHANGE, COPY, TRANSFER, REPLACE, NEW VOL, AMALGAMATE.

Az INSERT segitségével az operandusával adott egy vagy több szövegrész mögé, illetve elé lehet beszúrni, akár képernyőn begépelt, akár már a könyvtárban tárolt szövegrészt.
A DELETE parancs az operandusában adott szövegrészt törli.
Az EXCHANGE segitségével két szövegrész felcserélhető.
A COPY egy szövegrészt lemásol.
A TRANSFER a szövegrészt átviszi, azaz az az eredeti területéről törlődik.

A REPLACE utasitás ekvivalens egy DELETE-INSERT párral: az első operandusban adott szövegrészt a második szövegrésszel helyettesiti. A NEW VOL utasitás segitségével egy kijelölt szövegrész bemásolódik úgy, hogy belőle rögtön új kötet generálódik az alkotóházi könyvtárba.
Az AMALGAMATE utasitás segitségével két szomszédos, azonos szintü szövegegység eggyé olvasztható össze.

4. Ha tehát egy, az alkotóházban található kész kötetet olvasunk, és eszünkbe jut azt módositani, akkor először egy "nem kész" változatot kell készitenünk. Ez elvégezhető előre az OPEN utasitással, de az első módosító-utasitás enélkül is automatikusan végrehajt egy /implicit/ OPEN-t. A módositó- és olvasó utasitások segitségével a szöveg tetszőlegesen változtatható.
A CLOSE utasitás a könyvet "kész" állapotra hozza: ellenőrzi, hogy javitás során nem rontottuk-e el annak strukturáját; ha igen, akkor figyelmeztet, ill. párbeszédet kezdeményez, melynek során parancsokat és adatokat adhatunk a struktúra kijavitásához. A "kész" könyv már más által is olvasható.

5. Az, hogy egy könyvet "kész"-nek nyilvánitottunk, még nem jelenti azt, hogy egyben nyomtatásra alkalmas, szép formában is van. Ehhez előzőleg a formázó meghivásával a könyvet "tördelni", "szerkeszteni" kell. A formázó meghivása az EDIT paranccsal történik. Az utasitás kiadásakor a formázó program párbeszédben lekérdezi a formázás általános paramétereit: sorhossz, sorszám a lapon, fejléc, sortávolság, bekezdéskép, stb. majd nekilát az operandusban megadott könyv v. kötet megszerkesztéséhez. Ennek során

   1/ a szövegben található rövidités eket és szövegmakró-hivásokat kifejti;
   2/ a szövegbe ágyazott módositó utasitásokat interpretálja, ezáltal a szöveget a kivánt külső alakra hozza. Ennek során egy "nem kész" új szöveg készül, mely egy bizonyos "belső ábrázolásban" tartalmazza az eredeti módositó utasitásokat is. Ez a belső ábrázolás olyan, hogy nyomtatás, ill. képernyőre vetités során az igy ábrázolt utasitások elnyelődnek. Ez teszi lehetővé azt, hogy egyszer már megformázott szöveget újabb formázó utasitások beszúrása nélkül újra formázhassunk. /Viszont a "belső ábrázolásu" módositó-utasitások már nem módosithatóak/;

3/ a szöveg "strukturális mutatóit" /tartalomjegyzéki bejegyzéseket, tárgymutató-bejegyzéseket, minősitéseket/ megfelelően módositja, és kiegészíti azokat a szerkesztett tördelésének megfelelő koordinátákkal /oldal- és sorszám/.

A formátumvezérlő utasitások felsorolását lásd a Dll kötet 113-114 oldalain.

6. A szövegmakrók és röviditések kifejtése formázástól függetlenül is elvégezhető. Erre szolgál a SUBSTITUTE parancs, melynek paraméterrel adhatók meg a kifejtendő makrók és röviditések.

7. Formázáshoz és makrókifejtéshez alapértelmezések vannak; ennek megfelelően, ha az összes szövegmakrót, ill. röviditést ki akarjuk fejteni, ill. szabványos alakra kivánjuk hozni a szöveget, az EDIT, ill. a SUBSTITUTE nem kéri a paramétereket.

8. Struktúra elrontása, megvédése

8.1 Felmerülhet, mi történik akkor, ha javító utasitás segitségével "elrontunk" v. törlünk egy cimet, tárgymutató-beli bejegyzést, vagy pl. beszúrunk egy minősitett szövegrészbe. Több megoldás lehetséges:

a/ Az első módositástól kezdve megszünik annak a lehetősége, hogy tartalomjegyzék, tárgymutató, ill. minősités segitségével kereshessünk ki szövegrészeket. A továbbiakban hozzáférésre csak a tördelési- ill. képernyőstruktúra, továbbá a mintaillesztés használható, mig végül is a szöveget késszé nyilvánitjuk; ekkor a struktúra helyreáll az új helyzetnek megfelelően.

b/ Minden javitást azonnal követ a struktúra megfelelő változása.

c/ A javitások nem hajtódnak azonnal végre, hanem "feljegyződnek". A struktúra módositása a felgyüjtött javító utasitások tényleges végrehajtása után azonnal végrehajtódik.

8.2 Az olvasóterem előirásai sokkal szigorúbbak, mint az alkotóházéi. Igy pl. a tartalomjegyzék számozásának megszakitás nélkülinek kell lennie. A PUT utasitás az átvitellel egyidejüleg ellenőrzi az olvasótermi követelmények fennállását.

Irodalomjegyzék:

1 Aszalós-Kakas-Szőke, ANSWER információs rendszerének dokumentáció-kezelő programcsomagja /Nagyvonalú rendszerterv/, SOFTTECH sorozat, D11.
SzÁMKI, 1977. december

2. Aszalós János: ANSWER információs rendszerének részletes rendszerterve, SOFTTECH sorozat, D18, SzÁMKI, 1978.április.

A HARDWARE ÉS SOFTWARE KAPCSOLATA A GÉPHIBA KEZELÉS TERÜLETÉN

Tarján Mihály
Számitástechnikai Koordinációs Intézet

Bevezetés

Minden nem triviális számítógép-alkalmazás alapfeltétele, hogy hamis eredmények nem keletkezzenek észrevétlenül. Ez a követelmény független a megbízhatóság követelményétől. A hibák abszolút detektálhatósága a számítógép HW géphiba jelentési eszközein és az operációs rendszer géphiba kezelési eszközein múlik. A mai miniszámítógépek között sok olyan van, amelyik nem rendelkezik megfelelő géphiba jelentési eszközökkel. Az ú.n. mainframe számítógépek csoportjában már a harmadik generációra jellemző, hogy ez a követelmény elsődleges célja volt a kifejlesztésnek /1/.

Ezek a gépek olyan géphiba jelentési rendszert használnak, amely a gép HW egységei meghibásodását állapítják meg /2/.

Az erre épített operációs rendszerek képesek voltak a hiba detektálást megoldani, azonban csak "worst case" módszerrel voltak képesek az abszolút detektálást elérni /3/. Ez azt vonja maga után, hogy a futó programok abortálódnak olyan esetekben is, amikor csak gyanú és nem bizonyosság áll fenn a sérülésről.

Az előadás olyan architektúra bővítési módszert ismertet, amely lehetővé teszi a jelenleginél lényegesen jobb géphiba kezelési módszer alkalmazását HW és SW szinten egyaránt.

A teljes alkalmazási rendszer védelme

Egy teljes alkalmazási rendszer konstrukciójának ismeretében meg lehet határozni a működőképesség szempontjából kritikus elemeket. Ezek adatmezők, vagy kódszegmensek. Az a kérdés, hogyan kell ezeket az elemeket meghatározni, külön probléma; amely túlmegy az előadás keretein. Feltételezzük, hogy ismertek.

A géphiba jelentés rendszerének kiterjesztése

Az alkalmazási rendszer kritikus elemei fizikailag mint a főtár tartalmának egyes része azonosíthatók. Ezeket kell megfelelő figyelésben részesíteni a gép HW egységei mellett. Az adatmező sérülhet, míg a kódszegmensek végrehajtásának folyamata sérülhet meg. Az első feladat definiálni egy eszközt ezek azonosítására. Erre szolgál a leíró mező.

Egy leíró mező egy védelemben részesítendő elemet ír le. A mező fix hosszúságú és almezőkre oszlik. Az almezők leírják:

a védett elem kezdőcímét
az elem hosszát
az elem típusát

Az előadásban tárgyaljuk a javasolt architektúra kiterjesztés megvalósíthatóságát: a tárhozzáférés elveit
a tárcímzés
a mezők formai ellenőrzése
szempontjából.

A leíró mezők a tárban egymás után folytonosan helyezkednek el, végüket egy speciális típusjelölést tartalmazó mező jelzi.

A leíró mezők aktivitása az IPL fogalmának kiterjesztésével történik. A géphiba jelentési mezők egy pointermezővel bővülnek, amely egy védett elemet leíró mezőre mutat abban az esetben, ha az megsérült és ennek következtében géphiba megszakítás történt.

Az előadásban elemezzük a javasolt bővíté kapcsolatát az architektúra többi elemével. Elemezzük a bővítés hatását a bonyolultság növekedése, valamint a gép utasítás végrehajtási sebessége szempontjából. Elemezzük, milyen lehetőséget nyújt a kiterjesztés a meglévő architektúra géphiba kezelésének kompatibilis kiterjesztésére.

Az architektúra bővítés általános problémája

A kommersz számítógépek fejlesztésében követelmény az evolúciós fejlesztés a meglévő alkalmazások támogatása érdeké-

ben, ugyanakkor mindíg van indok, ami az architektúra kiterjesztése mellett szól. A problémakör exakt megfogalmazása jelenleg ismeretlen. A géphiba kezelés javasolt kiterjesztése tekinthető kísérletnek az architektúra kiterjesztés általános területén is.

Irodalom

1. R.P. Case, A. Padegis: Architecture of the IBM System /370
Communications of the ACM, Vol 21 No 1 p 73-96

2. IBM System/370 Principles of Operation

3. J.P. Birch: Architecture and Design of DOS/VS
IBM Syst. J. 1973 No 4. p. 401-411

INTERAKTIV KISSZÁMITÓGÉPES ELEKTRONIKAI TERVEZŐRENDSZER

Dr. Tarnay Kálmán - Baji Pál - dr. Gartner Péter - Kerecsenné Rencz Márta - Masszi Ferenc - dr. Nagy András - dr. Székely Vladimir - dr. Zólomy Imre
BME Elektronikus Eszközök Tanszéke

Bernus Péter
MTA Számitástechnikai és Automatizálási Kutató Intézet

1. BEVEZETÉS

Napjainkban egyre fontosabb szerep jut az összetettebb elektronikus alkatrészek alkalmazásának, számos területen elterjedten alkalmaznak kisebb sorozatu gyártás céljára hibrid (vékony- és vastagréteg) integrált áramköröket.
Fontos követelmény ezen áramkörök gyors tervezési és realizálási lehetőségeinek biztositása. A tervezés gyors elvégzését számitógépes tervezőrendszer segitségével lehet biztositani. A számitógépes tervezőrendszer a tervezés elvi fázisában és a gyártáshoz szükséges maszkok előállitásában egyaránt számottevő előnyökkel rendelkezik a hagyományos módszerekkel szemben. Miután közvetlenül biztositható a megtervezett maszkok előállitására szolgáló rajzgépvezérlő adatok gépi adathordozón való előállitása, kiküszöbölhetők ezek megrajzolásánál bekövetkező hibák.
A Budapesti Müszaki Egyetem Elektronikus Eszközök Tanszéke az MTA Központi Fizikai Kutató Intézetével és a REMIX Rádiótechnikai Vállalattal együttmüködve egy interaktiv tervezőrendszert dolgozott ki, mely hibrid integrált áramkörök kisszámitógépes tervezésére alkalmazható.

A programrendszer hardware szükséglete:

TPA-i kisszámitógép 16 kiloszó memóriával
256 kszavas minidiszk
Raszterdisplay
Mágnesszalag egység (2 db)
Gyorsolvasó és lyukasztó
Sornyomtató
Konzolirógép.

Az interaktiv tervezőrendszer ez idő szerint
3 általános célu programot (adatbáziskezelő, FOCAL interpreter és mérésadatok feldolgozására szolgáló programcsomag)
3 áramkörtervező programot (aktiv szürők szintézise, transzverzális szürők szintézise és TRANZ-TRAN 3 nemlineáris áramköranalizisprogram), valamint
2 kiviteli tervező programot (vékonyréteg és vastagréteg hibrid áramkörök kiviteli tervezése)
tartalmaz.

2. Az ISYS FELÜGYELŐPROGRAM

Az interaktiv tervezőrendszer céljára a tervezés sajátosságainak megfelelő szolgáltatásokat nyujtó felügyelőprogram, az ISYS (Interactiv System) került kidolgozásra.
A felügyelő program müködése:
A program inditásakor a display-en kiiródik a rendszerprogramok menüje. A tervező kiválasztja a futtatni kivánt programot. Ezután a felügyelő program kérdést intéz a felhasználóhoz, hogy a kiválasztott programot elejéről kell inditani (programkezdés) vagy pedig egy korábban már megszakitott futtatást kell a megszakitási ponttól folytatni (ujrainditás). Ez utóbbi esetben az ISYS közli a felhasználóval, hogy hány megszakitott futtatást tárol a mágnesszalag, majd a tervező válaszát várja, hogy ezek közül melyik variánst töltse be. A mágnesszalagról diszkre ill. memóriába való töltés után a display képernyőjén

az "első menü" jelenik meg, mely a következő lehetőségeket biztositja a felhasználónak:

1. Továbbinditás: a korábban megszakitott programot a megszakitás helyén továbbinditja.
2. Korábbi kép megjelenités: az ISYS megjeleniti a képmenüt, melyből a tervező kiválaszthatja a megjelenitendő képet.
3. Ujratervezés: az ISYS megjeleniti az ujratervezési menüt, a tervező ebből kiválaszthatja azt, hogy mely ponttól kivánja megismételni a tervezést.
4. Megszakitás: az ISYS megjeleniti a "második menüt", amely az alábbi lehetőségeket tartalmazza:
   a/ Továbbinditás: az ISYS megjeleniti az első menüt.
   b/ Programtárolás: az ISYS a programot megszakitott állapotban mágnesszalagra tárolja.
   c/ Adatbank tárolás: az ISYS a program által használt adatbank-file-t a felhasználó által adott névvel mágnesszalagra irja
   d/ Adatbank beolvasás: az ISYS a felhasználó által megadott adatbank-file-t beolvassa mágnesszalagról.
   e/ FOCAL: az ISYS meginditja a 4k FOCAL futását (a rendszer egy olyan FOCAL variánst tartalmaz, melyből vissza lehet térni az ISYS-be).
   f/ Futás befejezése: az ISYS alaphelyzetbe tér vissza és ismét a rendszerprogramok listáját tartalmazó menüt jeleniti meg.

Az ISYS tervezőrendszer valamennyi fentiekben felsorolt funkciója kiterjedt ellenőrzést végez és a nem megfelelő felhasználói utasitások esetén hibaüzenetekkel tájékoztatja a felhasználótaz elkövetett hibákról.

3. TERVEZŐPROGRAMOK

A bevezetésben felsoroltuk a rendszerhez tartozó programokat, a következőkben részleteznénk az egyes programok müködését és a felhasználás szempontjából fontosabb jellemzőit.

3.1 Adatbáziskezelő

A tervezőrendszer valamennyi programja egy közös adatbázist alkalmaz, a kezelőprogram elsősorban az adatbázis inicializálásával és karbantartásával kapcsolatos funkciók ellátására szolgál.

3.2 FOCAL interpreter

A tervező részére nyujt segitséget közbenső részletszámitások elvégzésének megkönnyitésével.

3.3 Mérésadat feldolgozó és kiértékelő programcsomag:

A programcsomag a gyártásközi és végellenőrzési mérések statisztikai jellegü kiértékelésére szolgál. A felhasználó által bevitt vagy az adatbázisban tárolt adatcsoportokon végez különféle müveleteket. A fontosabb szolgáltatások:

a./ Diszkrét halmazokkal kapcsolatos müveletek (átlagérték és szórás-számitás, sulyozott átlag- és szórásszámitás, hisztogram készités).

b./ Eloszlás függvények számitása (normál eloszlás, KHI-négyzet eloszlás, F-eloszlás)

c./ Regresszós eljárások (sik illesztése, másodfoku parabola és hatványfüggvények, exponenciális függvény, Weibull-függvény illesztése, univerzális regresszió).

d./ Szórás analizis (egyszeres, kétszeres osztályozással, valamint kölcsönhatások figyelembe vételével)

e./ KHI-négyzet próba (egyenletes és nem egyenletes várható eloszlás esetére).

A program a fenti számitásokat interaktiv módon végzi, és a programcsomag szubrutinjaira támaszkodva megbizhatósági adatok elemzésére is alkalmazható.

3.4 Transzverzális szürő szintézis program

A transzverzális szürő tervező program a sokmegcsapolásu késleltető rendszerből és összegezőből álló transzverzális szürő összegezőjének megtervezésére szolgál. A transzverzális szürő ill. korrektor tervezésének kiinduló adata lehet

a./ frekvenciatartománybeli specifikáció,

b./ időtartománybeli specifikáció.

A program alkalmas adott sulyfüggfényhez zajszempontból és zavaró jelkomponensek szempontjából optimális korrekciót jelentő reciprok sulyfüggvény meghatározására és T-szürővel való realizálásának tervezésére, valamint olyan transzverzális korrektor tervezésére, amelyet mintavételező detektor követ.
Az alkalmazott fontosabb matematikai módszerek: gyors Fourier transzformáció és dekonvolúció.

3.5 Aktiv szürő szintézis

A program az alul- és felülátereszto, valamint sávátereszto szürők karakterisztikáinak approximációját és különféle alaptagokkal való realizálásának tervezését teszi lehetővé. A polus-zérus elrendezés meghatározására az alábbi approximációs lehetőségeket nyujtja:
- maximális laposságu approximáció,
- Csebisev approximáció,
- inverz Csebisev approximáció,
- elliptikus (Cauer) approximáció.

Ezután lehetőség van az átviteli jellemzők táblázatos vagy grafikus dokumentálására:
- amplitudo karakterisztika,
- fázis (futási idő) karakterisztika,
- tranziens átvitel (egységugrás vagy Dirac delta gerjesztés).

A program a pólus-zérus elrendezés alapján az alábbi alaptagokkal való realizálást teszi lehetővé:
- Sallen-Kee alaptag
- végtelen erősitésü müveleti erősitős alaptag
- kettős visszacsatolásu alaptag
- integrátoros alaptag
- elliptikus alaptag

3.6 TRANZ-TRAN 3 nemlineáris áramköranalizis program

Funkcióit, szimulációs algoritmusait, modelljeit tekintve ez a program nem uj. Tanszékünk körülbelül egy évtizede foglalkozik univerzális áramköranalizis programok kidolgozásával. E munka eredményeként jött létre a TRANZ-TRAN áramköranalizis program több változata [1], [2]. Különlegessége viszont a TRANZ-TRAN

3/D programnak az eddigi változatokkal és a szokásos áramkörszimulációs programokkal szemben, hogy grafikus display-t alkalmazó, interaktiv müködésü program. A tervezőnek csak fel kell rajzolnia a képernyőre az analizálandó áramkört; a gép elvégzi a kért analizist, s eredményeit grafikusan, rajzok formájában közli a display képernyőn. Az ember-gép kapcsolatnak ez a - nálunk ujszerü - szervezése igen nagy jelentőségü a program kényelmes használata, eredményeinek könnyü értékelhetősége szempontjából. "Csak le kell ülni a képernyő elé" - a számitástechnikai elő-ismeretek teljes szükségtelensége és a vizuális ember-gép kapcsolat attraktiv volta nagyon megkönnyiti a tervezők számára a program használatba vételét.

A TRANZ-TRAN 3/D program teljesitőképesség szempontjából(kérhető analizis fajták, beépitett modellek, maximális hálózat méret) teljesen megegyezik a [3] alatt már ismertetett nem-interaktiv programváltozattal, ezért csak a program ujszerü vonatkozásait (grafikus bevitel és dokumentáció) részletezzük.

A tervező felrajzolja a képernyőre a vizsgálandó áramkört: az ernyőn az alkatrész helyét jelöli ki a CURSOR-pont segitségével. Ezután leüt egy billentyüt a klaviaturán (ellenállás esetén R betüt, kapacitásnál C-t, stb). Ennek hatására az ernyő kijelölt pontján megjelenik az illető alkatrész szokásos jelképi jele.

A program természetesen nemcsak a képernyőn nyugtázza egy-egy alkatrész bevitelét. Belső táblázatokat, listákat vezet a képernyőn épülő áramkörről. E listák szolgálnak a további feldolgozás alapjául. Minden müvelet egyidejüleg zajlik le a képernyőn és a fenti listákon. Ha pl. törlünk egy alkatrészt, eltünik az ernyőről a képe, és vele együtt mindazon listaelemek a táblázatokból, melyek hozzá tartoznak.

Az eredményközlés, ahol csak lehet, függvények formájában, grafikus módon történik.

Az egyenáramu analizis végén megjelenik a vizsgált áramkör képe, a mellette az ernyő jobb alsó sarkában egy kis keretezett mező. Ez utóbbi ugy funkcionál most, mint egy digitális voltmérő kijelzője. A CURSOR-ral, mint a voltmérő vezetékével "rálépve" a hálózat bármely

pontjára, a keretezett mezőben megjelenik az illető pont feszültsége.
A tranziens analizis során az eredényközlés olyan, mintha oszcilloszkóp ernyőjén jelennének meg a tranziens hullámformák.
A hálózat bármelyik pontjának feszültség-idő függvényét megnézhetjük és több csomópont hullámformája is egymásra rajzolható.
A display kis felbontóképessége miatt a tranziens időfüggvények finomabb részletei elveszhetnek. Ezen segit a dokumentálás nagyitás-funkciója: a függvények kijelölt kis részletei kinagyithatók a teljes képernyő méretére.

3.7 Vastagréteg hibrid integrált áramkör tervező program

A program vastagréteg hibrid integrált áramkörök kiviteli tervezésére szolgál. Kiindulási adat a TRANZ-TRAN 3 program által felhasznált áramkörleirás. Ennek adatait a program diszken keresztül veszi át a TRANZ-TRAN 3-tól. Az áramköröknek egy vagy két lapkás kiviteli tervét képes elkésziteni, kétlapkás kivitel esetén az alkatrészeket két csoportba kell osztani és a kiviteli tervek egymással párhuzamosan készithetők. A kétlapkás kivitelnél a lapkákat hátoldalukon ragasztják össze, ezért lehetőségünk van a lapkák tükrözött megjelenitésére is. A lapkák, paszták beültetett aktiv és passziv alkatrészek elektromos és geometriai adatait a rendszer adatbázisából veszi. Az alkatrészek elhelyezését és összekötését a tervező végzi és a program ellenőrzi azt, hogy valamennyi elkészült-e. A program lehetővé teszi több réteg egymás fölötti tervezését és a háttérben lévő rétegek szaggatott vonallal való megjelenitését. A program végeredménye a tervezett áramkör sornyomtatón kiadott dokumentációja, valamint rajzgép vezérlő lyukszalag.

3.8 Vékonyréteg hibrid integrált áramkör tervező program

A vékonyréteg hibrid integrált áramkör tervező program számos vonatkozásban hasonló az előzőkben ismertetett vastagréteg hibrid integrált áramkör tervező programhoz. Eltérés azonban, hogy itt csak egyféle négyzetes ellen-

állásu anyag használható, ezért bonyolultabb ellenállás alakzatok tervezése szükséges. A program lényeges részét képezi egy olyan meander tervező algoritmus is, mely tetszőleges ellenállás értékre és disszipációra meghatározza a lehetséges meander alakzatokat és a tervező által kiválasztott meandert értékbeállitó söntökkel is ellátja. A program végeredménye az aramkör dokumentációja valamint a rajzgép vezérlő lyukszalag a maszkok elkészitésére.

A program grafikus interaktiv lehetőségei tulajdonképpen csak az "élő" bemutatás során domborodnak ki igazán; irott szövegben kevés érzékeltethető belőlük. Előadásunkban viszont - vetités segitségével - megkisérlünk jobban érzékelhető képet adni e lehetőségekről.

Irodalom

1 dr.Tarnay Kálmán - dr. Székely Vladimir: TRANZ-TRAN 2 nemlineáris áramköranalizis rendszer, a Programozási Rendszerek '72 találkozó kiadványa, 229-303. old. Szeged, 1972.

2 dr.Tarnay Kálmán - dr. Székely Vladimir: A TRANZ-TRAN nemlineáris áramköranalizis program, Hiradástechnika, V.24.No.9. pp.257-264.(1973).

3 dr.Székely Vladimir - dr. Tarnay Kálmán - Rencz M. - Baji Pál: TRANZ-TRAN 3 - uj áramköranalizis programrendszer a TPA-i számitógépre, Programozási rendszerek '75 konferencia előadásai, NJSzT kiadvány, pp. 497-509, (1975. Szeged).

V. Székely - K. Tarnay: TRANZ-TRAN 3/A - a New Circuit Analysis Program for Small Computers, Proceedings of the Third International Symposium on Network Theory, pp.351-358. Sept. 1975, Split, Yugoslavia.

MULTIFUNKCIÓS OPERÁCIÓS RENDSZER AZ R11 TIPUSU MINISZÁMITÓGÉPEN

Trencséni István
VIDEOTON Fejlesztési Intézet

Bevezetés

Mi tette szükségessé az R10/R12 továbbfejlesztését?

Az R10/R12 real time alkalmazásai során kiderült, hogy komoly felhasználás esetén /pl. INTERFLUG/ a CPU kihasználtsága mindössze 30-40 %-os. Ez azt jelenti, hogy az idő mintegy 60 %-ban a CPU üres utasitást hajt végre. A CPU jobb kihasználása érdekében célszerünek látszik, hogy a gépet ne egycélúan /egy adott funkcióra/ használjuk. Ha azonban több funkciót akarunk látszólagos egyidejüséggel végrehajtani két problémával találjuk szembe magunkat:

. korlátozott memória méret,
. taskok és funkciók védelme.

- Memória méret
  Mint ismeretes az R10/R12 maximális mérete 32Kszó. Ez a tárméret elegendő egy-egy bonyolult real time alkalmazás implementálására. Azonban, ha nem egy célú az alkalmazás /több funkció/ szükséges a központi memória méretének növelése. Az R11 esetén a maximális kapacitás 512Kszó. Azonban, meg kell jegyezni, hogy egy-egy task mérete továbbra sem haladhatja meg a 16 bites szó által adott korlátot /32Kszó/. A reentrant program definiálásának lehetősége valamint a megosztott programszegmens bevezetése mellett ez a korlát valójában nem okozhat problémát.

- Taskok és funkciók védelme

Az Rlo/Rl2 hardware által biztositott védelmi lehetőség a memóriavédelmi kulcs /szavanként egy bit/.

Ez a védelem elegendő egy monofunkciós felhasználás esetén. Hiszen egyes taskok rendelkezhetnek nyitó kulccsal mig más taskok számára ez szükségtelen. S mivel monofunkciós a gép a rendszer belövése után biztonságosan üzemelhet. Meg kell jegyezni, hogy az Rlo/Rl2 real time felhasználása mellett a backgroundban program preperálás nem folyhat még akkor sem ha elegendő szabad memóriaterület állna rendelkezésre, hiszen egy belövetlen task megzavarhatja a real time funkciót. A fenti probléma kiküszöbölésére az Rll hardware architektúrájában a bázisregiszterek mellett be kellett vezetni a limitregisztereket is. Igy a bázis és limitregiszter segitségével a taskok szeparálhatókká váltak s nincs lehetőségük, hogy elérjenek /pl. hibás utasitás révén/ olyan memóriaterületet, mely ezen intervallumon kivülre esik /1. ábra/.

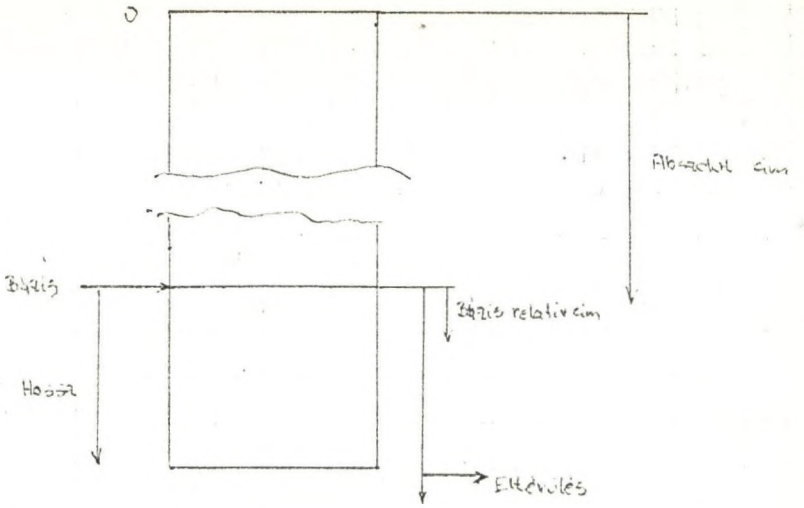
Igy már lehetőség nyilik arra, hogy egymás mellett belőtt és belövetlen taskok fussanak. A 2. ábrán egy példa látható a multifunkciós elv bemutatására.

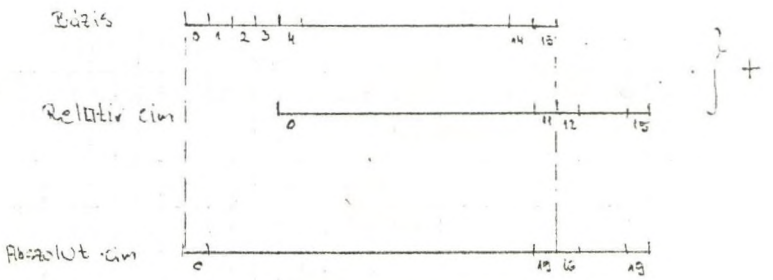
2. Általános tudnivalók

Az Rll-es software-jét oly módon definiálták, hogy megfeleljen a felhasználói problémák megoldásának. Ezek a problémák lehetnek akár real time jellegüek /folyamatirányitás, tranzakció kezelés.../ akár számitóközpont jellegüek.

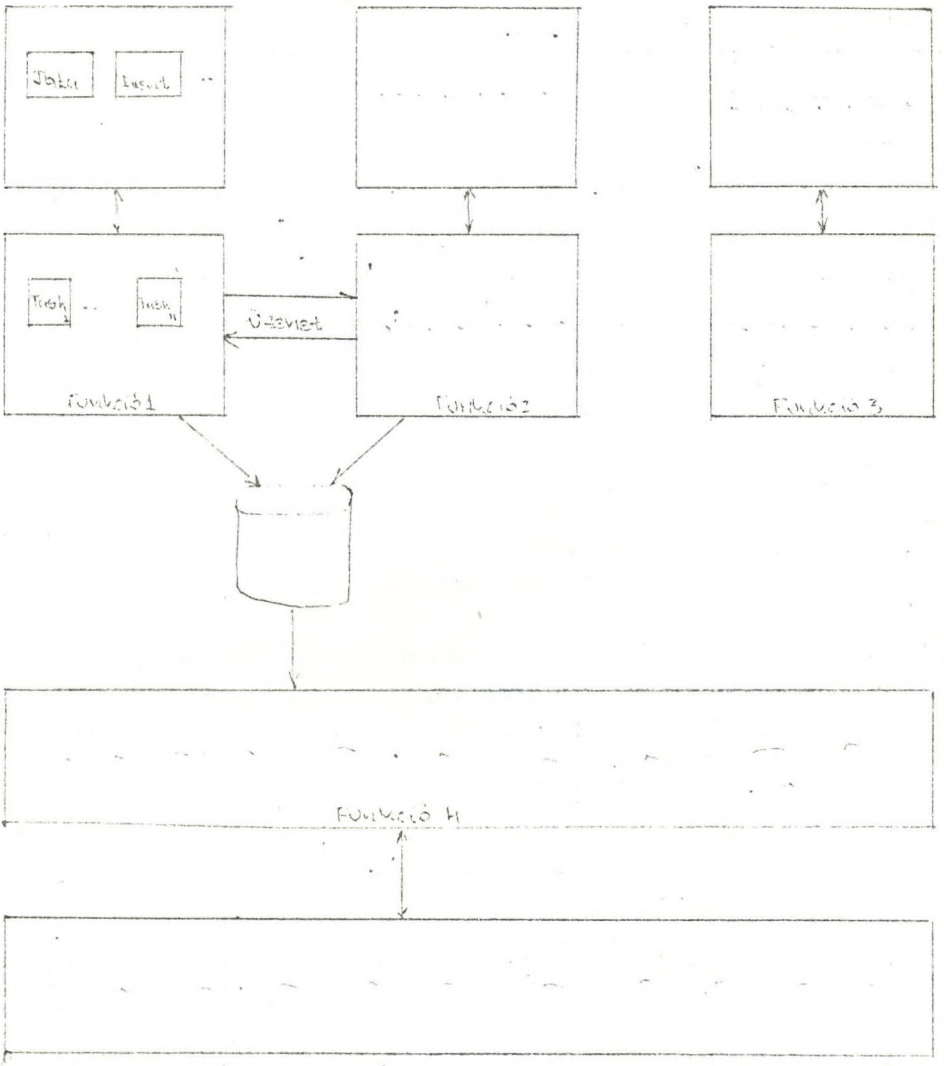
A gép hardware felépitése és az erre épülő software megadja annak lehetőségét, hogy ugyanazon a gépen különböző alkalmazások fussanak egymás közötti abszolút védelemmel.

Minden egyes alkalmazást bizonyos számú funkcióra oszthatunk /multifunkció /, egy-egy funkciót további feladatokra bonthatunk szét /multitask/. A hardware és software védelem lehetővé teszi a funkciók teljes függetlenségét. A szükséges kommunikáció az adott funkciók között a monitor segitségével üzenetek révén történik. Egy másik lehetőség a kommu-


0
Abszolút cím
Bázis
Bázis relatív cím
Hossz
Eltérülés


Bázis
0 1 2 3 4
14 15
Relatív cím
0
11 12
15
Abszolút cím
0
19 16
19
+

1. ábra



Funkció$_1$ = adat gyűjtés
Funkció$_2$ = szabályozás
Funkció$_3$ = program preparálás
Funkció$_4$ = statisztika

2. ábra

nikációra közös file-okhoz való hozzáféréssel történik. Ez utóbbi esetben a védelem a file-kezelő szintjén bevezetett elszámolási számokkal /account number/ és a hozzáférési jogokkal /access right/ történik.

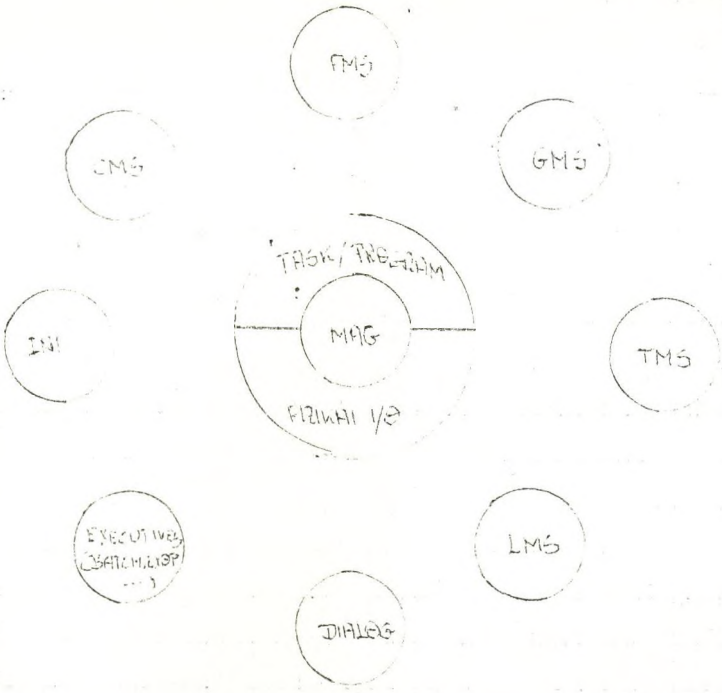
3. A monitor alapelve

A multifunkciós /multitask/ monitor a felhasználónak olyan központi szolgáltatásokat biztosit, amelyeket valamennyi task rendelkezésére lehet bocsájtani. Ezek a szolgáltatások segitik a felhasználót problémáinak megoldásában.
A monitor által nyújtott főbb szolgáltatások a következők /3. ábra/.

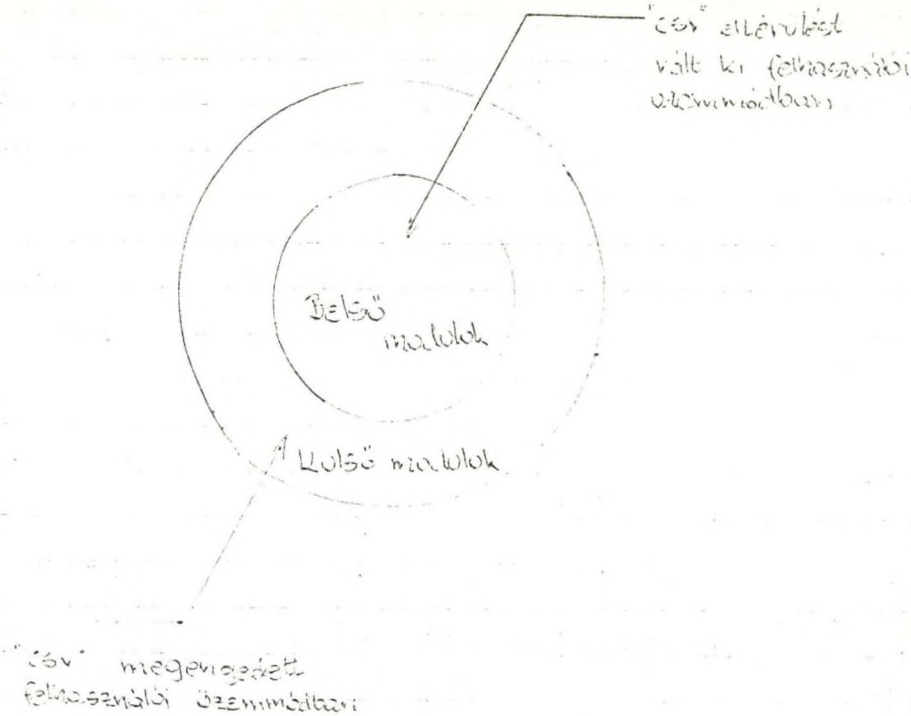
- programok kezelése,
- könyvtárak kezelése,
- fizikai input-output müveletek kezelése,
- task kezelés,
- taskok csoportjainak /funkcióknak/ a kezelése,
- dinamikus memória kezelés,
- események kezelése,
- üzenetek kezelése,
- szemaforok kezelése,
- idő és időzitések kezelése,
- eltérülések kezelése,
- megszakitások kezelése,
- operátori párbeszéd vezérlése,
- file katalógus kezelés.

Minden szolgáltatás supervisor primitivek halmazából áll. Ezen primitivekhez makrók útján férhetünk hozzá /makró-assembler/. A supervisor modulok CSV utasitással történő közvetlen hivása továbbra is lehetséges, de nem ajánlatos mivel a makrók használata egyrészt sokkal flexibilisebb másrészt a különböző monitorok közötti kompatibilitást egyszerübb biztositani ezen a szinten. Bizonyos supervisor primitivek csak privilegizálási üzemmódból érhetők el /4. ábra/.

A következő néhány fogalom bevezetése és definiálása segitséget nyújt a monitor alapelvének jobb megértéséhez.



3. slika


"cső" ellenőrzést
vált ki felhasználói
üzemmódban
Belső
modulok
Külső modulok
"cső" megengedett
felhasználói üzemmódban

4. ábra

Logikai gép /process/

Az R11 feldolgozó egysége olyan fizikai gép, amely egyszerre csak egyetlen utasitást hajt végre. A regiszterek elmentési és helyreállitási mechanizmusa azonban lehetővé teszi hogy ezt olyan n darab logikai gépből álló halmaznak tekintsük, amelyek közül egy adott pillanatban csak egy aktiv.

Taskok és programok

- Egy program egy utasitás sorozatból áll.
- Egy task egy végrehajtható programnak egy végrehajtási prioritási szintnek és ennek végrehajtásával megbizott logikai gépnek az összerendezéséből áll.

Két programtipust különböztetünk meg:

- nem reentrant program; ez esetben a memóriába a programnak annyi képét kell betölteni, mint ahány task hivja.
- reentrant program; ez esetben csak egyetlen egyszer kell betölteni a program képét a memóriába függetlenül a hivó taskok számától.

Egy task kontextje

Minden task állapota egy kontext-tel jellemezhető, melyet a központi memóriába helyezünk el és ahová a task cseréjekor elmentjük a task újrainditásához szükséges információkat.

Kontext csere

A kontext csere olyan müvelet, amely egyrészt a futó task újrainditásához szükséges információkat elmenti a futó task kontextjébe, másrészt újrainicializáljuk a gyorsregisztereket a megszakitó task kontextjének tartalmával.

Ezta müveletet automatikusan hajtja végre egy mikroprogramozott ütemező azonnali taskok esetén, mig elhalasztott taskok esetén a monitor programozott ütemezője hajtja végre.

Azonnali task

Azok a taskok amelyek az IT szinthez rendeltek.

Elhalasztott taskok

Azok a taskok, amelyek nincsenek IT szinthez rendelve.

Szegmentálás

A látszólag szimultán feldolgozott taskok szeparálásának szükségessége a szegmentáláson alapuló védelemhez vezetett. A programokat és/vagy adatokat változó hosszúságú /max. 32Kszó/ és áthelyezhető memória particiók tartalmazzák. Ezeket szegmenseknek nevezzük.

Szegmens

Egy szegmens egy bázissal és hosszal definiált folyamatos memóriaterület.

Szegmens leiró

Minden egyes szegmenst két szó ir le:

. bázis cim,
. hossz.

A hardware kezeli a bázisokat és hosszokat s igy lehetővé válik a szegmensek áthelyezhetősége.

Szegmens cimzés

Egy szegmens nem érhető el a felhasználói üzemmódban, csak ha leiróját előzőleg privilégizált módban betöltöttek a megfelelő regiszterekbe.

Funkció /task csoport/

Egy funkció olyan taskok együtteséből áll, melyek bizonyos számú közös szegmensen osztoznak, amelyek a csoport környe-

zetét képezik /virtuális gép/. Minden csoporthoz egy szegmens leiró tábla van hozzárendelve, amelyhez a felhasználó nem férhet hozzá. Ezen szegmens leiró táblára mutató pointer minden task kontextjében megtalálható.

- Szegmens tipusok
  - program szegmens,
  - stack szegmens,
  - megosztott szegmens,
  - adat szegmens.

4. A taskok közötti szinkronizálás és kommunikáció

Az előző fejezet bemutatta ugyanazon csoport taskjai közötti és különböző csoportok közötti védelem szükségességét. Azonban a taskoknak egymás között kommunikálni kell valamint egymást szinkronizálniuk kell. Tehát definiálni kell olyan ellenőrzött eszközöket, melyek biztositják a kommunikációt és szinkronizálást.

Kommunikáció ugyanazon csoport taskjai között

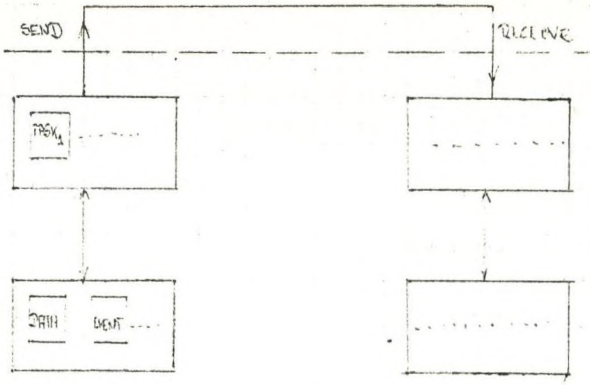
Ugyanazon csoporton belül a kommunikáció történhet a megosztott adatszegmens segitségével vagy a monitor segitségével küldött üzenetekkel.

Kommunikáció csoportok között

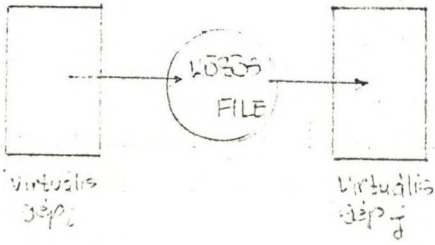
Két lehetőség adott:
- kommunikáció közös file-okon keresztül,
- kommunikáció üzenetekkel /monitor szolgáltatás/ /5. ábra/.

Szinkronizáció ugyanazon csoport taskjai között

A taskok szinkronizálása a csoporton belül definiált eseményekkel és szemaforokkal történhet.



Kommunikáció üzenettel



Kommunikáció lógós file-okkal

5. ábra

5. Kompatibilitás

Az R11 monitora kompatibilis az R1o/R12 multitask monitorával. A software lehetőséget nyújt R1o/R12 funkció definiálására. Ez azt jelenti, hogy az R1o/R12 multitask monitorának standard funkció szimuláltak az R11-en.

A P147PANG PRÓBAANYAG-GENERÁLÓ PROGRAM

Zágon Csaba
KSH Számitástechnikai Igazgatóság

A P147PANG program a direkt adatgenerátorok közé tartozik, azaz használatakor nem generálási feltételeket, szabályokat /rekordon belüli és rekordok közötti összefüggéseket/ kell megadni, hanem magát a generálási algoritmust kell paraméterezni, azaz azt, hogy az egyes mezők által felveendő értékek milyen törvényszerüségek szerint váltakozzanak. Ilyen tipusu adatgenerátorokból több is használatos Magyarországon /pl. az IEBDG[1], a DFANOUT[2], a TDG[3], stb./, azonban a P147PANG program szabadabban paraméterezhető és többféle müvelet elvégzésére képes, mint a többi.

Mielőtt rátérnék a P147PANG próbaanyag-generáló program ismertetésére, a próbaanyagok főbb jellemzőiről szólnék néhány szót.

A próbaanyagok előállitási módja lehet többféle /kézi, automatikus, vegyes/, de előállitásuk célja megegyezik - ellenőrzési célokra szolgálnak. Ebből következik, hogy a próbaanyagoknak teljeseknek kell lenniük. Azt, hogy milyen szempontból kell teljeseknek lenniük, a kivánt ellenőrzés fajtája szabja meg. Közös a próbaanyagokban az is, hogy időben általában gyorsan van rájuk szükség. Következésképpen - ha van választási lehetőségünk - előállitásukra a minél inkább automatizált mód az előnyösebb.
Az ellenőrzés tipusai a következők:
- kontrollista[1]/programok ellenőrzése,
- táblázó " " ,
- egyéb " " .

---

[1]/A "kontrollista" a statisztikai gyakorlatban használt fogalom, megfelelője a vállalati gyakorlatban a hibalista.

Ha kontrollista programokat kivánunk ellenőrizni, a próbaanyagnak nagyrészt hibás rekordokból kell felépülnie, annyi és olyan hibás rekordból, hogy azok összességükben tartalmazzák a lehetséges hibák jelentős részét. Fontos szempont az is, hogy a próbaanyag elkészitője ismerje ugy a próbaanyagot, hogy tudja azt, hogy mely hibáknak kell vagy kellene a kontrollistán kijönniük. Ennek akkor van szerepe, ha a próbaanyag előállitása automatikus, hiszen ekkor a próbaanyag-generáló programnak - a várakozással ellentétben - olyannak kell lennie, hogy ne véletlenszerüen hibás rekordokat szolgáltasson, hanem a program paraméterezéséből könnyen követhető hibákat.

A táblázó programok ellenőrzésekor a próbaanyagtól azt várjuk el, hogy jó rekordokból épüljön fel és tartalmazzon minden táblához rekordot. Itt is fontos szempont az, hogy a próbaanyag felépitése, tartalma ismert legyen, hiszen csak ekkor ellenőrizhető könnyen a táblázó program jósága.

Egyedül az egyéb programok ellenőrzésekor juthatnak jelentősebb szerephez az olyan próbaanyagok, amelyek valamely adott eloszlásokat követő adatokat tartalmaznak. Az ilyen jellegü alkalmazásra azonban az igények erősen korlátozottak, ezért itt az automatizálás kisebb jelentőségü.

A P147PANG próbanyag-generáló programmal elsősorban az olyan próbaanyagok előállitását igyekeztem megkönnyiteni, amelyek a két fontosabb ellenőrzési tipus által támasztott követelményeknek tesznek eleget - azaz a kontrollista programok és táblázó programok ellenőrzésére alkalmas próbaanyagok paramétermegadással történő előállitása volt az elsődleges célom, hiszen az igények tulnyomó többsége ilyen próbaanyagok iránt jelentkezik.

A P147PANG program fő jellemzői

A P147PANG program próbaanyag-generáló program. A program paraméterezése kulcsszavas paraméterek megadásával történik. Összesen 19 kulcsszó adható meg.

A program FORTRAN nyelven iródott, moduláris szerkezetü. A felhasználók csak a program load-moduljához férhetnek hozzá, a megfelelő eljárás meghivásával.

A program egyszeri hivásra 1-5 próbaanyagot /állományt/ állit elő. Ezek az állományok általában változó hosszuságu rekordokból épülnek fel, de megadhatók fix rekordhosszuságu állományok is. Az állományok külső jellemzőit - "kereteit" - az eljárás paramétereiben, tényleges képüket, tartalmukat pedig a program paraméterezésével határozzuk meg.

Pontos paraméterezési leirás[1] helyett az alábbiakban inkább a végrehajtható programfunkciókat szemléltetem.

A program működésének az alapja a rekord-kép.[2] A rekordképet elsősorban - de nem kizárólag - a rekordot felépitő mezők jellemzői határozzák meg. Csak a mezőjellemzők és egy-két rekord-szintü paraméter meghatározásával már lehetséges a program egyszerü használata.

A mezőjellemzők a következők:
név - a mező azonositója,
hossz - a mező hossza byte-ban,
kezdőpozició - a mező kezdőpoziciója a rekordban,
tipus - a mező tipusa /a két főtipus a kód és a számérték, de lehet a mező alfanumerikus és egyéb is/,
választási mód - a mezőhöz tartozó lehetséges értékek közül való választás módja /tetszőleges, minden, véletlenszerü/,

---

1/ A P147PANG program részletes ismertetése - igy a kulcsszavas paraméterek tételes felsorolása és pontos megadási szabályai is - [4]-ben található.

2/ A rekord-kép az egy adott tipusu rekordra vonatkozó rekordjellemzők összessége. A rekord-képet nemcsak az adott rekordszerkezete /a rekordot felépitő mezők jellemzőinek az összessége/ jelenti, hanem az olyan egyéb rekordjellemzők is, mint például a generálni kivánt rekordok kért és maximális száma, stb. A rekord-kép az a teljes információ, amit a P147PANG program egy adott állomány előállitásakor az állományt felépitő rekordokról tárol. Futás közben a rekord-kép természetesen változik.

érték - a mező lehetséges értékei,
ciklikussági paraméter - a mezőértékek képzési algoritmusa.[1/]

A mezőjellemzők és a kért rekordok száma rekord-szintü paraméter segitségével már készithető adott képü próbaanyag. A ténylegesen elkészülő adott képü rekordok számát az egyes mezőkhöz rendelt választási mód is befolyásolja, hiszen azoknál a mezőknél, amelyeknél minden lehetséges értéket kiválasztunk, a lehetséges értékek száma meghaladhatja a kért rekordok számát, ezért aztán a kért számunál ténylegesen több rekord is készülhet.

Alapvető a programban a kódok és a számértékek kezelése. Kód esetén a ciklikussági paraméter indexet határoz meg, azt, hogy a lehetséges kódok közül hányadik legyen a mező aktuális értéke, mig számérték esetén magát az értéket. Ha mezők egy adott rekordon belül egymással összefüggnek - főként, ha számértékek közötti számszerü összefüggésről van szó -, ez az összefüggés a ciklikussági paraméterek megfelelő megadásával megvalósitható.

1/

A ciklikussági paraméter egyes alparaméterei:

n_1 - a kezdőérték /alapértelmezés 1/

n_2 - a növekmény /alapértelmezés 1/

n_3 - a ciklus /alapértelmezés 1/
/minden n_3-dik rekordban n_2-vel megnöveljük az addigi értéket/

n_4 - a maximális érték /alapértelmezése kódnál a lehetséges kódértékeknek a száma, számértéknél viszont 10^K-1, ahol K- az adott mező hossza/.

A számitási képlet: $I = n_1 + \left\{ \left(\frac{J-1}{n_3} \cdot n_2 \right) \pmod{(n_4 - n_1 + 1)} \right\}$

ahol $J=1,2,\ldots,N$ /N az output rekordok száma/

$\div$ — az egészre osztás jele

$(\mod(n_4-n_1+1))$ — az adott szám (n_4-n_1+1) -gyel való osztásának a maradéka.

Paraméterezés, a paraméterezés megkönnyitése

A mező-szintü paraméterek rekord-szinten is megadhatók /kivéve természetesen a nevet és a kezdőpoziciót/. Ez a paraméterezés megkönnyitését jelenti /irásbeli könnyebbség, jobb áttekinthetőség/, hiszen a valamilyen szempontból azonos tulajdonságok egyszerre meghatározhatók.

Hasonló célt szolgálnak, az alapértelmezések is. Alapértelmezésről két szinten beszélhetünk. Az alapszintet a program alapértelmezései jelentik, a felső szintet pedig a parakártyán bevitt alapértelmezések /pl. az, hogy valamely mezőjellemző egy adott értékét a rekord-kép minden mezőjéhez hozzárendeljük - a "#" jelet használjuk a mezőnevek tételes felsorolása helyett/.

A paraméterezésről általában elmondható az, hogy a programban mindig a később megadott paraméter érvényesül. Ez alól az egyedüli kivételt az az eset jelenti, amikor valamely mező valamely jellemzőjét már konkrétan /mezőnév használatával/ megadtuk, és azután visszük be az adott mezőjellemző alapértelmezését. Ekkor a konkrétan megadott tulajdonságok érvényben maradnak, az alapértelmezés csak az addig konkrétan meg nem adott mezőkben érvényesül /a program alapértelmezése mindig csak utolsó sorban kerül behelyettesitésre/. Ez igy logikus is, hiszen megadhatjuk először az eltérő mezőjellemzőket, az összes többi mezőt pedig utólag ruházzuk fel az alapértelmezés szerinti tulajdonsággal.

A program bonyolultabb használata

Az összes programlehetőség kihasználását a speciális és szokványos rekord-szintü paraméterek használata biztositja. Ezekkel különböző müveletek végezhetők el.

Az elvégezhető müveletek a következőképpen csoportositha-tók:

- állománykezelés,
- rekord-kép változtatás,
- listázásvezérlés,
- feltételhasználat.

Az állománykezelés egy speciális rekord-szintü paraméter segitségével történik. Ezzel a paraméterrel adható meg az, hogy a paracsomagban a soron következő parakártyák melyik állományra vonatkozzanak az öt lehetséges állomány közül, valamint az is, hogy a próbaanyag előállitása az adott állományban az állomány elejétől vagy pedig folytatólagosan - az adott állományba a futás során eddig már beirt rekordok után - történjen-e.

A rekord-kép változtatás lehetősége lényegében a program paraméterezésének a megkönnyitését szolgálja, hiszen bármilyen rekord-kép változtatás elérhető a program egyszerü használata esetén is, csak akkor mindig ujra és ujra meg kell határozni a teljes rekord-képet, azaz a különböző rekord-képek közötti közös rész paraméterezését - ami esetenként számos parakártyát jelenthet - ismételten el kell végezni.

A rekord-kép változtatására a következő kulcsszavas paraméterek szolgálnak:
- az RV speciális rekord-szintü paraméter,
- az RM és RB rekord-szintü paraméterek.

Az RV kulcsszavas paraméterrel az aktuális /élő, utoljára meghatározott/ rekord-képet változtathatjuk meg. A változtatás lehet:
- uj mező/k/ beszurása,
- régi mező/k/ törlése,
- régi mező/k/ felülirása tetszőleges számu uj mezővel,
- régi mező/k/ kiválasztása a mezők nevének vagy sorszámának a megadásával.

Egy adott rekord-kép változtatás végrehajtása után természetesen már az uj rekord-kép lesz az aktuális. A beszurás, törlés és felülirás, valamint a kiválasztás egymást helyettesitheti. Az, hogy mikor melyiket célszerü használni, a konkrét helyzettől függ.

Az RM kulcsszavas paraméter segitségével az aktuális rekord-kép tárolása /a futás időtartamára szóló megőrzése/ történik meg. A tárolt rekord-képet egy számjegy azonositja. /Ennek a számnak semmi köze az állomány logikai egységszámához!/ Az RB kulcsszavas paraméter segitségével bármely - már eltárolt - rekord-kép betöltése történik meg. Az ilymódon visszahozott rekord-kép akár közvetlenül, akár közvetve - valamilyen rekord-kép változtatás segitségével - felhasználható output rekordok irására.

A fenti három kulcsszavas és a speciális állománykezelő paraméter segitségével a rekord-kép szabadon változtatható helyben /állományok között/ és időben /állományon belül/ - vagy akár helyben is és időben egyszerre.

A listázásvezérlés az IR rekord-szintü paraméter segitségével történik. Azt vezérli, hogy milyen mélységig jelenjenek meg üzenetek a sornyomtatón a paracsomag értelmezéséről, a P147PANG program müködéséről. Az IR paraméter a paracsomagban rekord-szinten bárhol megadható, mindig az utolsó megadása az érvényes /a rekord-képbe például ennek a paraméternek a tartalma is beletartozik/.

A listázásnak három - egyre bővülő - szintje van. A 0-szint, amikor csak a hibaüzenetek jelennek meg, nem tiltható le /a hibaüzenetek mindenképpen megjelennek/. Ez egyben az IR paraméter alapértelmezése is. Részletesebb információ kérése akkor célszerü, ha vagy maga a próbaanyag, vagy a paracsomagban meghatározott müveletek bonyolultabbak, hiszen igy könnyebben követhető a program müködése, az előállitott próbaanyag jobban áttekinthető, a hibakeresés egyszerübb.

A feltételhasználat speciális rekord-szintü paraméter segitségével történik. Ezen paraméterben összefüggések adhatók meg az adott rekord-képet kialakitó egyes kódszerü mezőkre. A feltétel hatására olyan output készül, amely tartalmazza a feltételben szereplő mezőkben a mezőkre itt megadott kódértékek - vagy minden lehetséges kódérték - összes változatát. A feltétel szerint előállitott próbaanyag azokban a mezőkben, amelyek az adott feltételben nem szerepelnek, olyan értékeket vesz fel, amelyek a rekord-képben megadott - az adott mezőre vonatkozó - általános értékképzési szabályoknak felelnek meg.

A feltételek használata főleg a táblázó programok ellenőrzésére szolgáló próbaanyagok esetén célszerü, hiszen igy könnyen előállitható olyan próbaanyag, amely minden lehetséges táblához tartalmaz rekordot. Azonban éppen azért, mert feltétellel igen könnyen - igen kevés irással - igen sok rekord előállitható, a feltételeket körültekintően kell felirni, nehogy véletlenül tul nagy legyen a próbaanyag.

Néhány szóban szeretném még vázolni a P147PANG program fejlesztési lehetőségeit. Ilyen - a program szemléletét meg nem változtató - fejlesztési lehetőségek a következők:

- adott feltételnek eleget tevő rekordok kihagyása a generált állományból;
- a generált állományról készitett lista átalakitása és megadott kulcsok szerinti esetleges rendezése;
- karakterestől eltérő számábrázolási formák biztositása a generált outputon;
- aritmetikai müveletek kulcsszavas megadása egyes mezők értékeinek más mezők értékeiből és megadott konstansokból való kiszámitására.

Irodalom

[1] IBM Systems: OS/VS Utilities. IBM dokumentáció, GC35-0005-4/1975.

[2] Jóba Csaba: A DFANOUT tesztadat-generáló program. Információ Elektronika 1975/3.

[3] ESZ 1012: TDG Tesztadat generáló felhasználói kézikönyv. - VT 202.051. 20.01-SW/1976.

[4] Felhasználói leirás a "P147PANG" próbaanyag-generáló programról - KSH SZIG belső kiadvány, SKSZ/31. 1977.

A BATCH ÜZEMMÓD SZIMULÁCIÓJA A SZÁMITÓGÉPKIHASZNÁLÁS VIZSGÁLATÁRA

Dr. Zárda Sarolta - Nagy Anna
Államigazgatási Számitógépes Szolgálat

Bevezetés

A számitógépes berendezések értékének növekedésével egyre jelentősebbé válnak a számitástechnika gazdaságosságával, a számitógépes rendszerek hatékonyságával kapcsolatos problémák. A fejlett számitástechnikai kultu-rával rendelkező országokban a számitástechnikai tevékenységek mérésével, a mérési adatok feldolgozásával foglalkozó új szakterület jött létre, a compumetrics. A compumetrics a számitógépes rendszerek méréseken alapuló kiválasztásával, a működő rendszerek hatékonyságának mérésével, javitásával és a jövőbeni rendszerek fejlesztésének előkészitésével foglalkozik. A vizsgálati körébe tartozó problémákat a következőképpen csoportosithatjuk:

- a gép és konfiguráció kiválasztása;
- új rendszerelemek,,ill. rendszerek fejlesztése;
- működő rendszerek hatékonyságának javitása;
- erőforrások korrekt elszámolása.

Ezen feladatok megoldására az alábbi eszközeink, módszereink vannak:

- gépi teljesitményidők mérése, összhasonlitása;
- mixek;
- magprogramok;
- analitikus modellek

- benchmark programok;
- szintetikus programok;
- monitorok
  - -- hardware monitor;
  - -- software monitor;
- szimuláció
  - -- szimulációs programcsomagok;
  - -- sorbanállási modellek alkalmazása.

Az utóbbi időben a szimulációt egyre szélesebb körben alkalmazzák a compumetrics-ben. Ennek oka a multiprogramozásu, time-sharing, multiprocesszoros rendszerek elterjedése. Ezen rendszerek kifinomult megszakitási technikájának dinamizmusát, bonyolult müködését nem képesek tükrözni az analitikus modellek és módszerek.

A számitógép üzemeltetés gazdaságossági problémáival hosszabb ideje foglalkozunk a KSH Államigazgatási Számitógépes Szolgálat keretén belül. Előadásunk célja bemutatni egy számitógépkihasználást vizsgáló szimulációs modell felállitását és SIMSCRIPT nyelven való megfogalmazását.

A feladat pontos definiálása: egy olyan szimulációs modell készitése, amely egy multiprogramozott időosztásos elv alapján müködő, kötegelt /batch/ számitógépes feldolgozást szimulál teljes leterhelést feltételezve. A vizsgálati cél az operációs rendszer paraméterezhető részének beállitása ugy, hogy adott feladat-struktura mellett a számitógép kihasználás a lehető legjobb legyen. Ezt hivjuk megfelelő üzemeltetési stratégia kidolgozásának.

Előadásunk vázlata:

1. A modellezés folyamatának sajátosságai, szintjének megválasztása.
2. A rendszer ismertetése és szimulációs modellje
   2.1. A modell adatbázisa, beáramlási folyamat
   2.2. A rendszer erőforrásai, kapacitásai
   2.3. A SIMSCRIPT modell működése
3. A rendszer értékelési szempontjai, statisztikai outputok

1. A modellezés folyamatának sajátosságai, szintjének megválasztása

Bonyolult rendszerek esetében /pl. számitógép működése/ különösen fontos a modell szintjének helyes megválasztása. Ez egyik kritériuma a modellvizsgálat hatékonyságának és gazdaságosságának. Ennek alapján a HwB 66/20-as számitógép modellezésekor funkcionális modellezést végeztünk. A funkcionális modell elhanyagolja a konkrét berendezések mikroszintü jellemzését /pl. 1 utasitás végrehajtásának idejét, módját/, azonban egy program, ill. egy jobóstep feldolgozási jellemzőit hüen tükrözi. Az operációs rendszer müködésének, feladatainak , software-hardware kölcsönhatásoknak, az információ áramlásának vizsgálatával a számitógép működése jól jellemezhető.

A számitógép működése elsősorban tömegkiszolgálási rendszerként modellezhető. Az igények a jobok, ill. job-stepek. Ezek " állnak sorban" és várnak a kiszolgálásra. A kiszolgáló egységek az operációs rendszer elemei és a hardware egységek, amelyek foglalkoznak az igényekkel. Például tárigényük kielégitése, perifériák hozzárendelése, tényleges beolvasás. Az operációs rendszer elemei egy-egy hardware erőforrással gaz-

dálkodnak. A gazdálkodás időbeli dinamizmusát megszabja az elosztandó hardware erőforrások kapacitása, a kiszolgálási, ill. megmunkálási ideje, és az erőforrást igénylő jobok jellemzői. A modell igy figyelemmel kiséri a hardware egységek kihasználását, az előttük várakozó sorokat, választ ad az ütemezési politikák, valamint a feladatstrukturából adódó fontos mutatókra, pl. a multiprogramozási faktorra. Ezek alapján egy több kiszolgáló berendezéssel ellátott, nyilt tömegkiszolgálási rendszerről beszélhetünk.

2. A rendszer ismertetése és szimulációs modellje

A rendszer összetevői:

- a végrehajtásra váró jobok, ill. jobstepek,
- a jobok végrehajtását elősegitő erőforrások.

2.1. A modell adatbázisa, beáramlási folyamat

A vizsgált rendszer célja a jobok, ill. a job-stepek feldolgozása. Ahhoz, hogy modellezhető legyen egy igény végigfutása a rendszeren, ismerni kell a kiszolgálási sorrendet, valamint az egyes kiszolgálási időket. A jobok, ill. a job-stepek ezen értékeinek meghatározásakor két információ-forrásra támaszkodhatunk:

- vezérkártyák információi,
- az operációs rendszer által elszámolási célra gyüjtött részletes rendszerstatisztika.

A szimulációs modell input adataiként egy meghatározott nap 196 jobstepjéből álló 81 jobjának adatai álltak rendelkezésünkre. Ebből a halmazból választottuk ki a rétegezett mintavétel szabályai szerint azt az 57 jobstepből álló 20 jobot tartalmazó job folyamot, amelyet szimulációnk során végig futtatunk a rendszeren. A minta tükrözi az átlagos terhelést.

Az igények rendszerbeérkezését beáramlási folyamatnak definiáljuk. Dinamikus a beáramlási folyamat, ha /0,t/ időintervallumban folyamatosan, valamely F_t eloszlásfüggvénnyel jellemezhető módon érkeznek be. Az igények beérkezési folyamatán a jobok beolvasását értjük. Igy modellünk beáramlási folyamata dinamikus. A beáramlás másik sajátossága a jobok egymás közötti beérkezési sorrendje, ez határozza meg ugyanis egy adott időpillanatban a job-mixet, a rendszer munkaösszetételét. Egy adott időpillanatban a job-mixet a ciklikus érkezések és az uj érkezések igényei szabják meg. A ciklikus beérkezéseket a rendszer algoritmusai alapján maga kezeli, az uj beérkezések az alapsokaságban elfoglalt időbeli sorrendjük szerint érkeznek a rendszerbe, biztositva a véletlenszerüséget.

2.2. A rendszer erőforrásai, kapacitásai

Az erőforrásokat két nagy csoportba sorolhatjuk:

- hardware erőforrások
- software erőforrások.

A hardware erőforrásokhoz tartozik a központi egység, az operativ tár, a háttér tárak, az input/output egységek, a csatornák. A software erőforrás az operációs rendszert és a különböző rendszerfileokat foglalja magába.

Hardware erőforrások

A szimulált konfiguráció a következő:

- egy processzoros alapgép /proc.+ op. tár/
- perifériák: lyukkártyaolvasó /CRU 1050/
  sornyomtató /PRU 1200/
  2 db mágneslemezegység /MSU 400/

4 db mágnesszalagegység /MTU 400/

A hardware egységeket kapacitásukkal és teljesitményadataikkal jellemezzük.

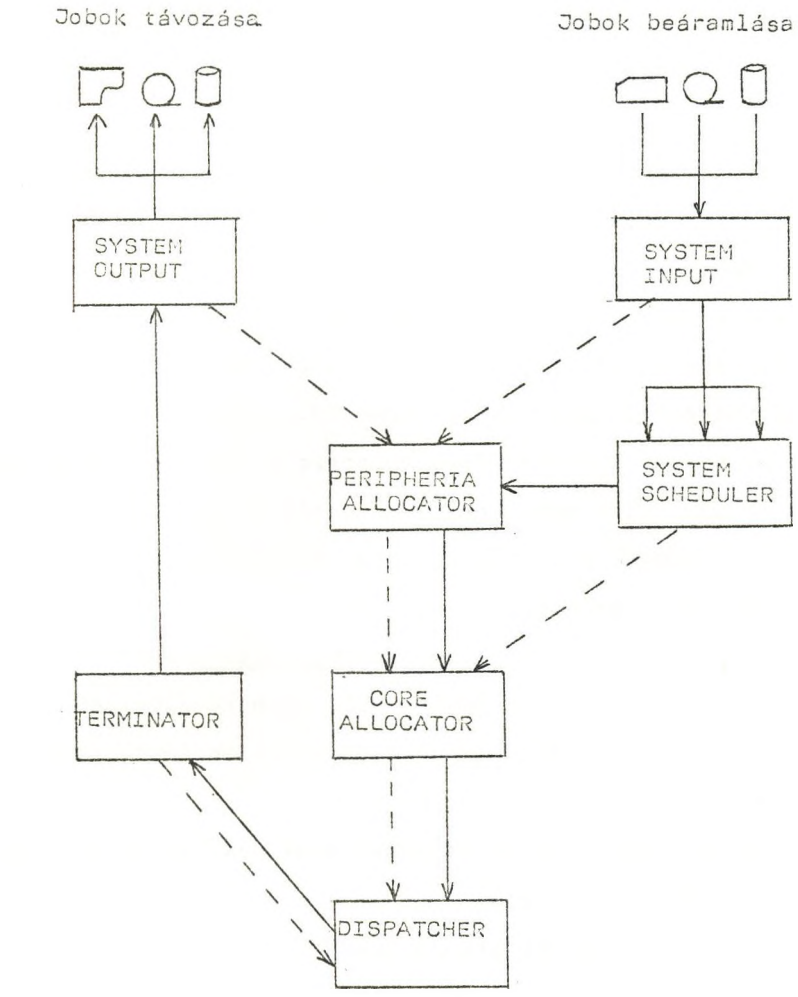
A software erőforrások az operációs rendszer keretében működő programok. Az operációs rendszer vezérlőprogramjai képezik modellünk gerincét. A vezérlőprogram elemei és feladatai:
SYSTEM INPUT gondoskodik egy jobéstream rendszerbeérkezéséről, működése során létrejövő program. A szükséges tárkapacitás, igényelt berendezések, output követelmények, stb. alapján erőforrásigényeiknek megfelelően sorokba rendezve adja tovább a jobokat a következő rendszerelemnek. Nem rezidens.-
SYSTEM SCHEDULER feladata a már háttér tárakon várakozó jobok megfelelő összeválogatása, ütemezése és a jobok job-stepekre való felbontása. Nem rezidens.
PERIPHERIA ALLOCATOR funkciója a job-stepek periféria igényének megállapitása, szabad perifériák hozzárendelése, ill. lefoglalása. Nem rezidens.
CORE ALLOCATOR gondoskodik a memória kezeléséről és kiosztásáról a job-stepek között. Rezidens elem.
TERMINATOR funkciója a feldolgozott job-stepek által felszabadult erőforrások "elengedése", visszaadása a "készletraktárnak", erőforráshasználat elszámolása. Rezidens elem.
DISPATCHER feladata a processzor kiosztása a job-stepek között, a központi egység időosztásos algoritmusának, megszakitási rendszerének kezelése. Rezidens elem.
SYSTEM OUTPUT gondoskodik a jobokhoz tartozó job-step outputjának "összeszedéséről" és az output müvelet végrehajtásáról. Nem rezidens.

2.3. A SIMSCRIPT modell müködése

A rendszer gépi modelljét SIMSCRIPT nyelven fogalmaztuk meg. A SIMSCRIPT diszkrét, eseményorientált szimulációs nyelv. A rendszer összetevőinek ábrázolása modellünkben a következő: A jobok, ill. a job-stepek a modell ideiglenes, a hardware erőforrások pedig állangó objektumai. Ezeket attributumaikkal jellemezzük, pl. a job-step egyik attributuma memóriaigénye. A software erőforrások müködési algoritmusát endogén esemény-rutin keretében irjuk le. Az algoritmus tükrözi az operációs rendszerelemek müködését. Azok a rutinok, amelyek legalább egy másik operációs rendszer-rutin kiszolgálását igénylik bizonyos szempontból hasonlitanak egy felhasználói programhoz. A modell tükrözi ezt a sajátosságot. A felhasználói programoktól való megkülönböztetésük magas prioritásuk, programszámuk, és attributumainak értékadásával történik. Mindegyik operációs rendszerelem előtt sor áll, amelyben az adott op. rendszerelem funkcióját igénylő programok állnak. Ezek a sorok /halmazok/ többnyire prioritásos szervezésüek. A modell megpróbálja tükrözni a rendszer bonyolult megszakitási mechanizmusát, igy a megszakitást kiváltó eseményeknek megfelelően eseményrutin kezeli ezt a problémát.

Minden program tényleges végrehajtása csak akkor kezdődik, ha processzort kapott. /A DISPATCHER neki osztotta ki./ Ha operációs rendszerelem kapta meg a CPU kvantumot a "futás" ugy megy végbe, hogy végrehajtja az op. rendszerelem saját feladatát az előtte várakozó igényen.
A szimuláció a müszak elején indul, egy szimulációs időegység 10^0 msec.

A SIMSCRIPT modellben a jobok áramlását, valamint az operációs rendszerelemek funkcionális kapcsolatát az alábbi ábra szemlélteti:



3. A rendszer értékelési szempontjai, statisztikai outputok

A rendszert különböző célfüggvények alapján értékelhetjük:

A felhasználó igénye:

- a job átl. átfutási idejének minimalizálása /válaszidő/
- a job átl. késési idejének minimalizálása.

Az üzemeltető igénye:

- számitógép egységek átl. kihasználási %-ának maximalizálása
- számitógép egységek állásidejének minimalizálása
- átbocsátó képesség növelése.

Felállitható olyan cél is, amely egy konkrét számitóközpont üzemeltetési sajátosságait veszi figyelembe. Pl. a memória szükkeresztmetszetet képez a HwB 66/20-on. A multiprogramozási faktor az ütemezési algoritmustól /üzemszervezés + op. rendszer/ és az ütemezendő jobok erőforrás igényeitől függ. Igy a megfelelő rendszer müködés kritériumának az átl. multiprogramozási faktor maximális értékét kell tekinteni.

Modellünkben a rendszer értékelési szempontjaként válasszuk: az adott idősávban mozgó - a felhasználó számára elfogadható válaszidő mellett - a számitógépegységek kihasználási százalékának maximalizálását.

Ezt számolhatjuk modellünk outputjának statisztikai mutatóiból:

- job átlagos átfutási ideje
- sorstatisztika minden periféria tipusra voatkozóan
- központi memóriához kapcsolódó mutatók
- processzor kihasználtságát jellemző mutatók.

Futtatásaink számszerü eredményeiből még nem tudtunk tapasztalatot leszürni, a modell fejlesztés alatt van, további kisérleti környezetre és futtatásokra van szükségünk.

Felelős kiadó: Károly Antalné
Felelős szerkesztők: Dávid Gábor, Havass Miklós
Engedélyszám: 49586
Megjelent 45,3 /A/5/ iv terjedelemben
/I. kötet 22,4 /A/5/ iv, II. kötet 22,9 /A/5/ iv/
400 + köteles példányban

SZÁMKI Sokszorosító Üzem, Budapest
Felelős vezető: Büky Józsefné
SZÁMKI 780796