NJSZT
MUSZAKI ÉS TERMESZETTUDOMANYI EGYESULETEK SZOVETSÉGE
NEUMANN JÁNOS SZÁMÍTÓGÉPTUDOMÁNYI TÁRSASÁG
BUDAPEST VI., ANKER KOZ 1 • • LEVÉLCÍM: 1368 BUDAPEST PF 240 • • TELEX 22-5369 • • TELEFON 229-870

PROGRAMOZÁSI RENDSZEREK '78
Konferencia

I. 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

I. kötet

SZEGED
1978. november 8-20.

Szerkesztette:

DÁVID GÁBOR

HAVASS MIKLÓS

ELŐSZÓ

Kötetünk az 1978 novemberében Szegeden megrendezett PROGRAMOZÁSI RENDSZEREK'78 konferenciára beküldött előadások szövegét tartalmazza. A konferenciát a MTESZ Neumann János Számitógéptudományi Társaság és az MTA Matematikai és Fizikai Osztálya harmadizben hirdette meg.

A háromévenként sorra kerülő konferencia célja a software rendszerek, ezenbelül elsősorban a hazai kisszámitógépek, valamint az ESZR berendezések programrendszerei terén az előző évek során kutatásban, fejlesztésben, felhasználásban elért hazai eredmények bemutatása, megvitatása, összehasonlitása.

A konferencia előadásai, amelyek jól reprezentálják elért eredményeinket, alkalmasak az önvizsgálatra. Mennyit dolgoztunk? Vannak-e időtálló eredményeink? Észrevettük-e a valóban ébredő igényeket?

A földmüves dolga nemcsak vetés és aratás. Munkája nagy része: okos ritkitás, védő gyomlálás.

A mi szárba szökkenő vetésünk dus vegetáció képét mutatja - a vetés megtörtént! Azonban mintha kevés lenne a termést hozó erős egyed, a hasznos hozam.

Sok például az alapfokon végzett párhuzamos fejlesztés, az ismételten megtett zsákutcák száma nagy. Nem önmagában a párhuzamosság ténye az elgondolkodtató, hanem az, hogy ezek a fejlesztések nem támaszkodnak egymás

eredményeire, nem törekszenek a próbálkozások értékelése utján magasabb rendü szintézisre. Talán ennek eredményeként kevés az olyan igényes eredmény, amely tulnőtt a kisérleti munka keretein, amely a számitástechnika vérkeringésébe került. De kevés az igazán mély elméleti eredmény is. Hiányzanak egy-két sokat igérő uj irány szárnypróbálgatásai, igy például a mikroprocesszorok, az osztott adatbázisok.

Mik az okok? Talán nem látjuk jól a célt? Nincsenek jól definiálva a követelmények? Nincs erőnk végigvinni a gondolatokat?

Használjuk fel konferenciánk lehetőségeit arra, hogy kilépve a napi teendők közel tekintő gondjaiból, szemünket a közösen megteendő utra vetve, diszkutáljuk jövőnket. Hiszem, hogy konferenciánk hozzájárul ahhoz, hogy Karinthy kiképző tisztjével elmondhassuk: bár kezünk még reszket, de szemünk már tisztábban látja a célt.

Havass Miklós
a programozási rendszerek
(software) szakosztály
elnöke

TARTALOM

I. KÖTET

MŰKÖDÉSELEMZŐ PROGRAMRENDSZER

dr. Adámy László - Kovács János
Számítógépalkalmazási Kutató Intézet

Az Országos Számítástechnika-alkalmazási Iroda és a Számítógépalkalmazási Kutató Intézet között fennálló szerződés alapján a SZÁMKI vállalta középkategóriájú R-gépek általános rendszerelemező és hatékonyságvizsgáló, valamint elszámolási programrendszerének kidolgozását. E munka során közép-kategóriájú R-gépeknek tekintjük mindazokat az /R-10-esnél nagyobb/ ESZR--alapú rendszereket, amelyek adatfeldolgozó, alap-irányítási, számítástechnikai és információs feladatokat látnak el úgy, hogy feladataik túlnyomó többségét jól körülhatárolható munka-egységekben /JOB-okban/ oldják meg. Fő üzemmód a "kötegelt" /batch./ feldolgozás.

A programrendszer célja: mindazoknak az igényeknek a kielégítése, amelyek egy, a munkáját meghatározott szolgáltatások formájában végző számítástechnikai szervezetben /SZGSZ-ben, Számítógépes Szolgálat/ a rendszerelemző, hatékonyságvizsgáló és elszámolási rendszerrel kapcsolatosan felmerülnek.

A megvalósítás módja: a különböző felhasználói feladatok végrehajtásához az alap-tevékenységet biztosító számítógépes programokon kívül különböző alrendszereket hozunk létre, amelyek összessége az SZGSZ működéselemző rendszere.

Ezek az alrendszerek a következők:

ellenőrzési alrendszer

- on-line ellenőrzés végrehajtása OS és DOS/POWER operációs rendszerben,
- on-line adatgyüjtés OS, DOS és DOS/POWER operációs rendszerben,
- off-line adatgyüjtés és feldolgozás az automatikusan nem rögzített eseményekről /kártyainput/,
- egyes kisegítő adatállományok direkt, ill. indexszekvenciális szervezésüek, de a törzs-adatállomány jelentős része szekvenciális módon kerül feldolgozásra,
- jelentéseinket egy általános jelentés-generáló programmal szolgáltatjuk,
- dinamikus erőforrás-felhasználás orientált elszámolást biztosítunk.

A rendszer első fázisának üzeme a SzÁMKI SZGSZ-ben 1978. január 1-én kísérleti jelleggel megkezdődött.
A fejlesztés második szakaszában főbb feladatink az alábbiak:

- off-line adatgyüjtés csoportos adatrögzítőn,
- on-line ellenőrzés OS-HASP-ban,
- a rendszer kiterjesztése TAF környezetre,
- ESZR gépeken általánosan használható adatbázis-kezelő rendszer alkalmazása.

Tapasztalatok

Mivel a SzÁMKI-ban DOS+POWER, OS/MVT+HASP környezetben a rendszer 1978. január 1-től kísérleti jelleggel üzemel, sok tapasztalat, észrevétel összegyült a müködéselemző rendszerrel, illetve az őt körülvevő környezettel /Hw, Sw, üzemeltetés, felhasználók/ kapcsolatban.
Az előadásban ezzel kapcsolatban a következő területekkel foglalkozunk:

- megbízhatóság /Hw, Sw/

. adatgyüjtő alrendszer
. kiértékelő /elszámolási/ alrendszer
. adatbank generáló és karbantartó alrendszer és
. visszajelentő, lekérdező alrendszer.

A müködéselemező rendszer kidolgozása

Tervezési szempontok:
- A rendszer kidolgozásánál fő szempont volt az általános /DOS, OS, off-line adatgyüjtés/ használhatóság kialakítása, ami lehetővé teszi a rendszer használatát
  - vegyes operációs rendszer környezetben /DOS, OS/
  - egy meghatározott operációs rendszer /DOS, ill. OS/ környezetben.
- A rendszer az adatgyüjtésből származó adatokat /rekordokat/ egységes szerkezetü belső rekordokká alakítja át, ahol formailag már nincs különbség a különböző helyekről származó rekordok között.
- A rendszer felépítése olyan, hogy a rendszer egyszerüen bővíthető, új elemek, funkciók viszonylag egyszerüen beépíthetők. Továbbfejlesztési szempontból a rendszer nyitott.

A rendszer programjainak tervezésénél alapvető szempont volt az egyszerüség, áttekinthetőség, a könnyen karbantarthatóság, a DOS, ill. OS környezetben egyaránt való használhatóság.

A kidolgozás szakaszai

Az első fázisban az alapvetően batch környezetre választjuk meg a rendszert.
E realizálás jellemzői:
- multiprogramozhatóság /DOS, OS/
- hatékonyságnövelő eszközök
- felhasználói környezet
- üzemeltetés technológia

FELADATMEGOSZTÁS ESZR GÉP ÉS FRONT-END PROCESSZOR KÖZÖTT A CEDRUS TERMINÁLHÁLÓZATBAN

Arató András - Sarkadi-Nagy István - Telbisz Ferenc
MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A KFKI-ban 1977-ben installálásra került egy R40-es számitógép. A gép átadása után rövid időn belül felmerült az igény, hogy a felhasználóknak interaktiv szolgáltatásokat kell nyujtani, elsősorban a jobok előkészitéséhez, és a háttértárak kezeléséhez, tekintettel arra, hogy az ilyen célokra használható segédprogramok kezelése meglehetősen kényelmetlen, és az erőforrások kihasználása sem gazdaságos ebben a környezetben.

2. A feladat és az architektura elemzése

Az első sorozatu ESZR gépek architekturájának alapos megvizsgálása után arra a megállapitásra jutottunk, hogy az erőforrások gazdaságos felhasználását csak alapvetően "batch" jellegü üzemmód mellett lehet elérni. Ezt az alábbi meggondolásokkal lehet alátámasztani:

- A memória maximális bővithetőségi határa, legalább is a sorozat nagyobb sebességü gépeinél, viszonylag alacsony a processzor sebességéhez képest. Igy mivel virtuális memória kezelés nincsen, a rendelkezésre álló memória még munltiprogramozás esetén sem teszi lehetővé a processzor kapacitásának a kihasználását. Ehhez hozzájárul a perifériák /különösen a mágneslemezek/ lassu volta is. Egy időosztásos rendszer /TSO/ használata ezt az ellentmondást csak még jobban kiélezné.

- A gép megszakitási rendszere olyan, hogy nem alkalmas nagyon nagyszámu programmegszakitás kiszolgálására: a korszerü kisgépeknél meglévő, a hardware által támogatott nagyszámu és különböző prioritásu szintü interrupt belépési ponttal ellentétben itt lényegében csak egy belépési pont áll rendelkezésre a külső programmegszakitások számára.

- A gép operációs rendszere /OS/ is alapvetően kötegelt fel-

gozásra készült, amit többek között a task-váltás hosszadalmassága /300-500 utasitás/, a preallokálási stratégia stb is mutat.

Ezzel szemben viszont a nagygép háttértára igen nagy mértékben bővithető /400-500 Mbyte sem irreális/, ami nagy on-line adatbázis kiépitését teszi lehetővé.

Mindezek alapján olyan megoldást választottunk, ahol a több felhasználó egyidejü, interaktiv kiszolgálását front-end processzorral valósitjuk meg, és a nagygépben csak azokat a funkciókat tartjuk meg, amik szorosan kapcsolódnak a mágneselmezes háttértárak kezeléséhez /v.ö. [1]/. Az ilymódon kialakitott rendszernek az alábbi szolgáltatásai vannak:

- interaktiv szövegszerkesztés
- job-ok leadása a kötegelt feldolgozás számára, illetve az eredmények lekérdezése
- általános file-kezelés.

A két előbbi szolgáltatást interaktiv módon használt display terminálokról lehet igénybe venni, mig a harmadik a rendszerbe kapcsolt kisgépek /intelligens terminálok/ programjainak számára teszi lehetővé a hozzáférést a nagygép mágneslemezes háttértáraihoz [2].

3. Feladatmegosztás a rendszer komponensei között

A szövegszerkesztést végző rendszerkomponenseknél az volt a célunk, hogy minden funkciót abban a gépben valósitsunk meg, amelyikben az a leggazdaságosabban elvégezhető.

Igy a szerkesztő programnak a nagygépben futó része csak a file kezelést és a rekord szinten történő módositásokat végzi el. A front-end processzor által sorbaállitott utasitásokat sorosan dolgozza fel, egyidejüleg mindig csak egy felhasználóval foglalkozva. Ez csökkenti a központi egység terhelését, a memóriaigényt, és jól illik a nagygépes feldolgozás kötegelt jellegéhez.

A front-end processzor végzi párhuzamos feldolgozással a felhasználói parancsok dekódolását, a rekordon belüli javitásokat /felhasználva a képernyő lehetőségeit/, a karakter sorozatok szerinti keresést, illetve azok cseréjét.

A job kezelés lényegében a nagygép fukciója maradt. Igy a jobok bevitelére, illetve az eredmények kiadására továbbra is az operációs rendszer "reader/writer" programjait haszmáljuk. A front-end processzor feladata, hogy a szövegszerkesztő rendszer, illetve a távoli batch terminálok felé a kommunikációt elvégezze, és igy a rendszer be/kiviteli programok ezeket a job-okat is teljesen ugy tekinthetik, mintha helyi perifériális berendezéseken történne meg a job-ok bevitele, illetve az eredmények kinyomtatása. Erre a kérdésre a 4. fejezetben még visszatérünk.

Hasonlóképpen a job-ok, illetve a nagygép állapotának a lekérdezése is megtörténhet bármelyik terminálról, a lekérdezést a front-end processzor egy, az operációs rendszer által összetett konzolként kezelt kártyaolvasót/sornyomtatót megvalósitó cimpáron végzi el. A front-end processzor itt egyrészt szintén a kommunikáció terhétől szabaditja meg a rendszert, másrészt a teljes lekérdezhetőség biztositása mellett megakadályozza a rendszer állapotának meg nem engedhető módositását a terminálokról.

A file kezelő rendszer feladata az, hogy a távoli kisgépek felhasználói számára hozzáférést biztositson a nagygép mágneslemezes háttértáraihoz és sornyomtatóihoz. Itt a nagygépben futó program végzi el a file kezelést a kisgépekben futó programok számára. A kisgépek ezeket "helyi file"-oknak látják. Ez esetben is a front-end processzor feladata, hogy a kommunikációt az adatátviteli protokolok segitségével lebonyolitva, a nagygépet tehermentesitse.

4. A nagygép és a front-end processzor szinkronizálása az I/O rendszer segitségével

Mivel az elvégzendő feladatok a nagy ESZR gép és a front-end processzor között meg vannak osztva, felmerül a különböző gépeken egymástól függetlenül futó programok, illetve operációs rendszerek összehangolása az input/output tevékenységen keresztül. Ennek a problémának a megoldásához nagy segitséget jelentettek annak a hardware illesztőbe-

rendezésnek a lehetőségei, mely kapcsolatot teremt a TPA busza és az ESZR csatorna között. Ennek a berendezésnek a segitségével 128 cim emulálható az ESZR gép felé különböző parancskészletekkel.

A szinkronizációhoz két tipusu berendezést emuláltunk 16 fizikai cimen: 8 kártyaolvasót és 8 sornyomtatót. Közülük egy részt az ESZR/OS rendszerprogramok kezelnek, mint szabványos géptermi perifériákat, egy másik részét fizikai szinten felhasználói programok, mint blokk strukturált perifériákat, vagy mint szabványos perifériákat logikai szinten programozva. A job-ok átadását a kötegelt feldolgozás számára például ugy valósitottuk meg, hogy az ESZR gépen futó editor program a leadandó feladat JOB vezérlő kártyáit egy blokk-strukturált perifériás cimen "leküldi" a front-end processzornak, az pedig egy szabványos géptermi kártyaolvasót emulálva, az OS READER-nek adogatja a kártyaképes rekordokat. Amennyiben távoli géptől érkezik a feladat, a megfelelő taszk közvetlenül a kártyaolvasót emuláló cimre továbbitja a rekordokat. OS WRITER esetében az irány forditott. Az emulált sornyomtató cimen érkező sorokat a fizikai cimeket emuláló LINK TASK vagy visszaküldi az ESZR gépben futó EDITOR programnak, vagy egyenesen a távoli kisgépnek küldi el a kommunikációs vonalakat kezelő TRANSPORT MANAGER TASK-on keresztül.

Az ilyen feladatmegközelitésnek két előnye van. Egyrészről nincs szükség a nagy terjedelmü kommunikációs elérési módszerekre az ESZR gépben, másrészről az operációs rendszer módositására sincs szükség, mivel a szabványos géptermi perifériák módositás nélkül beilleszthetők a rendszerbe.

A kártyaolvasó és sornyomtató páros emulálásával még egy igen fontos gyakorlati probléma oldható meg. Az OS rendszerbe mint összetett konzol generálhatók be. Igy a TPA-hoz kapcsolt egyik display berendezéssel és egy mátrix nyomtatóval kiváltható az ESZR gép megbizhatatlan, lassu mechanikus irógépe.

Az input/output szinkronizáció kérdése a front-end processzorban jelentkezik, mivel az ESZR gép nem "veszi észre", hogy nem valódi perifériái vannak. A szinkronizációt a bufferek biztositására lehet

visszavezetni. Ha kellő időpontban megfelelő helyen van elegendő szabad buffer, akkor az input/output tevékenység engedélyezhető, a szinkron a két operációs rendszer között biztositva van. Hogyan valósitható ez meg ESZR csatorna esetében "passziv" kártyaolvasó és sornyomtató segitségével? Az ESZR csatornák alapértelmezés szerint "aktivak", ez adódik a centralizált gép filozófiájából. A perifériák nem adhatják meg azt, hogy milyen utasitást kivánnak végrehajtani és mikor. Egyetlen lehetőség adódik a szinkronizációra: a státusz válasz. Három alapvető státusz választ alkalmaz a fizikai cimeket emuláló LINK TASK. Ha olyan cimet szólit meg az ESZR csatorna, melyhez a többi taszk nem biztositott buffert, akkor a végső statusz válaszban a LINK TASK a UNIT CHECK /UC/ bitet egybe állitja, és az OS supervisor SENSE utasitására beavatkozás szükséges SENSE bitet /INT REQ/ küld be, ezenkivül jelzi az érdekelt taszknak, hogy az ESZR gép a cimen olvasni vagy irni akart. Ha a LINK TASK megkapja a buffert, akkor aszinkron statuszként DEVICE END /DE/-et küld a csatornának, amire az OS supervisor elinditja ujra a korábban visszautasitott input/output tevékenységet. Az irás vagy olvasás végén a LINK TASK csak CHANNEL END /CE/ statuszt küld vissza mindaddig "visszatartva" a DE bitet, amig ujabb buffert nem kap a cimhez. Igy biztositható az emulált cim folyamatos müködése.

Ennek a szinkronizációs algoritmusnak a használatával elérhető, hogy az emulált fizikai perifériacimekhez dinamikusan rendelhető buffer, csak ha arra szükség van, ha kellő időben nincs megfelelő buffer, akkor az input/output tevékenység felfüggesztődik, ha megérkezik a buffer, akkor a tevékenység operátori beavatkozás nélkül ujraindul.

5. Következtetések.

A KFKI-ban elkészült CEDRUS hálózatban a front-end processzor alkalmazásával sikerült interaktiv szövegszerkesztési és job kezelési lehetőséget biztositani az ESZR nagygép felhasználói számára. Ugyancsak

a front-end processzor tette lehetővé azt, hogy a nagygépben nem kellett alkalmaznunk a memóriaigényes és nehézkes távadatfeldolgozási hozzáférési módot. Bár a rendszer lehetővé tenné azt is, hogy egyes felhasználói programokkal is lehessen a terminálokról interaktiv kapcsolatot létesiteni, a 2. fejezetben mondottak miatt ennek szolgáltatásként történő megvalósitását nem tartjuk időszerünek.

6. Irodalom

[1] R.D.Russel - P.Sparman - M.Krieger: ORION - The OMEGA remote interactive on-line system.
Proc.International Computing Symposium, 1973. p. 143 /North Holland, 1974/

[2] A.Arató-I.Sarkadi-Nagy-F.Telbisz: A local network for the support of software development.
Proc.of COMNET '77. Budapest 1977. Vol. I. p. 227.

AZ ANSWER INFORMÁCIÓS RENDSZERÉNEK /IR/ AXIOMATIKUS LEIRÁSA

Aszalós János
Számitógépalkalmazási Kutató Intézet

A probléma

A tágan értelmezett IR-ben /lásd SOFTTECH D6./ a mai software gyártás legnehezebb problémái együttesen léptek fel. Létre kell hoznunk egy sokféle tipusú információt tartalmazó adatbázist; biztositanunk kell nagy adatmennyiségek mozgatását és szimultán, többszintü, gyors elérését egy kérdés-válasz rendszer keretén belül; meg kell oldanunk a párhuzamos processzek deadlock-mentes müködését, az adatbázis módositásának és hozzáférésének szinkronizációját. A rendszer interaktiv jellegéből adódóan a legmesszebbmenőkig "humanizálni" kivánjuk a rendszer kezelését, az input megszervezését, és számolunk a rendszer és az adatbázis hordózhatóságára előirt követelményekkel. A szövegkezelés általánossága többszintü nyelvi rétegek tervezését igényli, beleértve a makrótechnikát és távlatilag egy élő nyelvi réteget is. Elképzelhető, hogy a jövőben a rendszert egy számitógép-hálózaton kell müködésbe hoznunk, tehát az ezzel kapcsolatos problémákat is figyelembe kell vennünk.

Mindezek a tényezők együttesen egy olyan programtervezési módszer kifejlesztését követelték meg, mely képes a fenti tényezők együttes figyelembevételére. Hangsúlyozzuk,

hogy "módszeren" egy software-eszközbázist és annak használatára vonatkozó eljárásmódot /és gondolkodási módot/ értünk, tehát nem csupán fegyelmi intézkedések betartását.

Az előadásban először egy mesterséges példa kapcsán bemutatjuk a módszer magját képező alapgondolatokat /1. fejezet/. Ezek súlyával Dijkstra, Hoare, Jones és mások munkáira támaszkodnak /1: az irodalomjegyzéket/. A módszerünk célja, hogy az ott kifejtett elveket a gyakorlati megvalósitás, az egyszerü kezelhetőség irányába továbbfejlesszük. A 2. fejezetben bemutatjuk a módszer eszközbázisát képező programcsomagot. "Éles" példaként az IR első fázisában kivitelezett dokumentációs rendszer axiomatikus programtervét külön kiadásban tettük hozzáférhetővé.

1. Az eljárásmód

Az eljárásmódunkat erős egyszerüsitéssel mutatjuk be.

A világ, vagy annak bármilyen kis része, a külső szemlélő számára meghatározott "dolgok" /="objektumok"/, "viszonylatok" és "mozgássémák" /=algoritmusok/ együttesének tekinthető. A csecsemő, a gyerek, a felnőtt ugyanazt a világot látja, de mégis mást-mást fog fel belőle, más-más szinten érti meg. A programozási feladatot is egy olyan zárt világnak foghatjuk fel, melyen a világot több szinten kell leirni, ill. megérteni.

Minden szinten lerögzitjük az észlelhető objektumok tipusait, az egyes tipusokba tartozó változókat vagy elemeket, majd megadjuk a rájuk vonatkozó, vagy kapcsolataikat kifejező lehetséges állitásokat. Végül axiómák formájában rögzitjük az egyes állitások közötti kapcsolatokat.

Az egyes szintek közti kapcsolatokat az ún. absztrakciós leképezések teremtik meg. Ezek mutatják meg, hogy egy meghatározott szinten megfogalmazott objektumok, állitások, ill. axiomák milyen más objektumok, állitások, axiomák együttesévé bomlanak az alacsonyabb szinten, ill. milyen objektumokba, állitásokba, axiomákba sürüsödnek a magasabb szinten.
A rendszer az elsőrendü predikátumkalkulus nyelvén "beszél". Ezenkivül néhány egyszerü fogalmat használ, pl. "lista", "lista i. eleme", "≡" /azonosság/, "=" /egyenlőség/, stb.

A mondottakat egy egyszerü feladat első két szintjén mutatjuk be.

Legyen egy törzsadat file-unk /RÉGI/, melyet egy hasonló szerkezetü /MOD/ file-lal hozunk naprakész állapotba /UJ/. Mindegyik file rekordok listája. A rekordok kulcsszót és értéket tartalmaznak, és mindegyik file-on belül a kulcsszó szerint vannak rendezve. Egy file-on belül egy adott kulcsszó csak egy rekordban szerepelhet.

A többi részlet a példa szempontjából érdektelen. A céljainknak megfelelően egyszerüsitett jelölésmód értelmezése nem kiván magyarázatot. /A feldolgozó programcsomag által elfogadott jelölésrendszer bővebb, és ettől bizonyos mértékig eltér./

1. szint

objektumok: tipus: file:x,y,z,RÉGI,UJ,MOD
proc: ERROR
állitások: üres /x/
nemüres /x/
rendezett /x/
összefésült /x/

axiomák:

	axioma	magyarázat
1.	file /x/ $\Longrightarrow$ üres /x/ $\oplus$ nemüres /x/	A file "üres" vagy "nemüres". A "$\oplus$" jel a kizárólagos 'vagy' jele
2.	üres /x/ $\Longrightarrow$ reńdezett /x/	Az üres file-t eleve rendezettnek tekintjük.
3.	$\neg$ rendezett /RÉGI/ V $\neg$ rendezett /MOD/ V $\neg$ nemüres /MOD/ $\Longrightarrow$ ERROR	Megadjuk az elvégzendő szintaktikus vizsgálatokat. Megköveteljük a file-ok rendezettségét.
4.	üres /RÉGI/ & nemüres /MOD/ & rendezett /MOD/ $\Longrightarrow$ UJ=MOD	Ha a RÉGI üres-pl. az első felvitelkor - akkor az UJ értékben azonos lesz a MOD-dal.
5.	rendezett /RÉGI/ & rendezett /MOD/ & nemüres /RÉGI/ & nemüres /MOD/ $\Longrightarrow$ összefésült /UJ/ & rendezett /UJ/	Megadjuk a program feladatát a szinten bevezetett fogalmak segitségével
6.	összefésült /x/ $\Longrightarrow$ nemüres/x/	

A primitiv szemlélő az 1-es szinten csak három, file-tipusú objektumot észlel. Az egyes objektumokról négyféle állitást tud kimondani. Az axiomákban előirja, hogy az egyes állitások milyen kapcsolatban állnak egymással.
/Vegyük észre, hogy az axiomák egy része input /3/, ill. input-output /4. és 5./ specifikációnak felel meg. Ezen a szinten azonban a szinten tartozó algoritmust, azaz "mozgásformát" sem adtuk meg, s ezért az input-output specifikációkat az axiomák közt soroltuk fel. Más esetben, amikor a szinten algoritmus-leirás is szerepel, ezt éppen ezen specifikációk segitségével definiáljuk./

2. szint

objektumok: tipus: rekord: a,b
természetes szám: m,n
lista: w
elem: EOF
állitások: eleme /a,w/
előbb /a,b/
megfelel /a,b/
length /w,n/

Axiómák:

	axiómák	magyarázat
1.	rekord /EOF/	A file-vég is rekord.
2.	eleme /v,w/ & lista /w/ $\Longrightarrow$ rekord /v/	Minden lista rekordokból áll.
3.	length /w,m/ & m $\geq$ 2 & eleme /a,w/ & eleme /b,w/ $\Longrightarrow$ előbb /a,b/ $\oplus$ előbb /b,a/ $\oplus$ a $\equiv$ b	Trichotómia elve, mely a file-ok teljes rendezettségét lehetővé teszi.
4.	lista /w/ & length /w,m/ & m $\geq$ 2 & a $\equiv$ elem /i,w/ & b $\equiv$ elem /j,w/ & i $<$ j $\Rightarrow$ előbb /a,b/	A file-ok rendezettségét mondja ki.
5.	lista /w/ $\Longrightarrow$ eleme /EOF,w/	A file-vég minden listában szerepel.
6.	eleme /a,w/ & eleme /b,w/ & $\neg$ (a $\equiv$ b) & a $\equiv$ EOF $\Longrightarrow$ $\neg$ (b $\equiv$ EOF)	A file-vég minden listában legfeljebb egyszer szerepel.
7.	eleme /a,w/ & $\neg$ (a $\equiv$ EOF) előbb /a,EOF/	A file-vég minden lista utolsó eleme.
8.	eleme /a,RÉGI/ V eleme/a,MOD/ $\Longrightarrow \exists$ b (eleme /b,UJ/ & megfelel /a,b/)	A RÉGI és MOD file-ok minden rekordjához van az UJ-nak megfelelő eleme.

absztrakció

1. szint		2. szint
file /x/	$\Longleftrightarrow$	lista /x/
üres /x/	$\Longleftrightarrow$	length /x,1/
nemüres /x/	$\Longleftrightarrow$	length /x,m/ & m$\geq$2
rendezett /x/	$\Longleftarrow$	3., és 4. axióma
összefésült /x/	$\Longrightarrow$	8. axióma

Vegyük észre, hogy a megfigyelő mindkét szinten csak a világ objektumait és relációit figyelte meg; a mozgásokat nem. Az axiomák alapján azonban könnyü generálni a szintnek megfelelő részletességü "mozgássémát", azaz primitiv algoritmust. /Más természetü feladatoknál a mozgássémák megfogalmazása válhat elsődlegessé. Ilyen megfogalmazási módot az általános szövegkezeléssel kapcsolatban [SAM-IV -3] -ban találhatunk; az ott ismertetett eljárás azonban már egyfajta cselekvési logika ismeretét tételezi fel./

Mi határozza meg, hogy a szemlélő egy adott szinten milyen objektumokat vesz észre? A szinten felsorolt állitások generálása önkényes, csak a tapasztalaton alapul vagy létezik valamilyen kényszeritő erejü, tudományosan is megalapozott eljárásmód ezzel kapcsolatban? Az objektumok kapcsolatait leiró axiomák a valóságos feladat leglényegesebb tulajdonságait fedik-e? Mi biztositja azt, hogy egyet sem "felejtettünk ki" a felsorolandók közül?

A fenti kérdésekre vonatkozóan jelenleg nem rendelkezünk olyan alapokkal, mely az elméleti szigorúságot a gyakorlat tapasztalatával egyesitené. A software-komponensek egyik feladata az, hogy az "advocatus diaboli" szerepét játszva a felhasználót következetességre és alapos meggondolásra kényszeritse.

2. A programcsomag feladatának leirása

A módszert támogató, interaktivan müködő programcsomag feladata a következő:

1. Asszisztál az egyes szintek kialakitásánál. Az asszisztencia a szöveg szintaktikus elemzésében, hibajelzések kiadásában, a szöveg belső formátumú tárolásában, és a módositást megkönnyitő eljárások biztositásában áll;
2. Segitséget ad az állitások, az axiomák és az absztrakciós összefüggések kifejlesztéséhez azzal, hogy "rákérdez" az egyes objektumokra, az adott szinten kialakitott állitások közti összefüggésekre, az aktuális és az előző szint állitásainak kölcsönös kapcsolatára, s a válaszokat értelmezi, tárolja /lásd az alábbi dialógus-mintát/.

3. Egyes szinteken belül ellenőrzi az axiómák ellentmondásmentességét.
4. Ellenőrzi az egyes szintek közti összefüggések teljességét, azaz megvizsgálja, hogy az adott szinten megjelölt objektumok, állítások egyértelműen bomlanak-e fel az alacsonyabb szintü objektumokra, ill. állításokra.

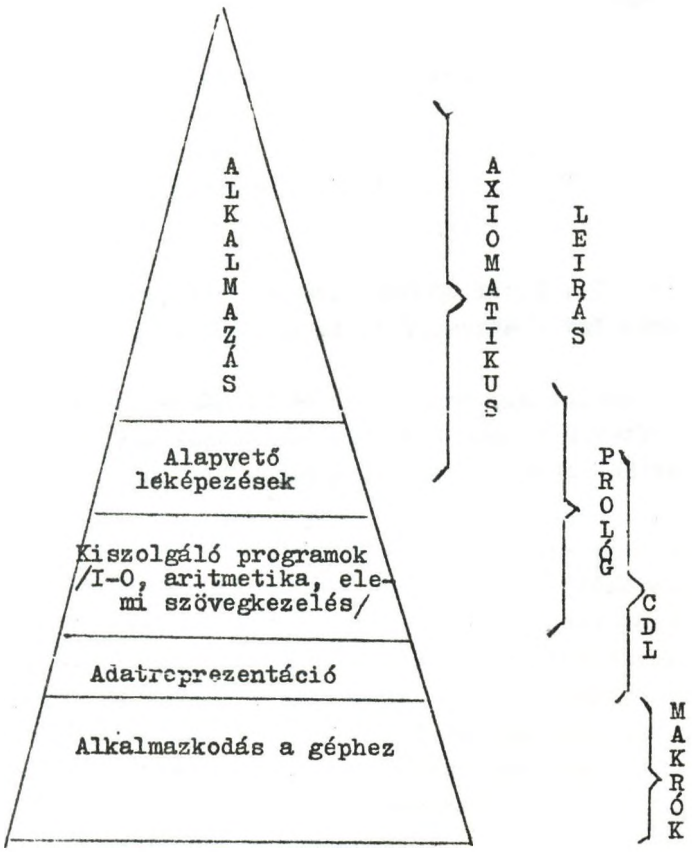
Bemutatjuk az első szint kialakitásánál lefolytatott dialógus röviditett és az olvashatóság kedvéért "magyarositott" formáját. /G: gép, F: felhasználó/

G: Ird le a specifikációt, jelöld meg és minősitsd a kulcsszavakat!
F: [: begépeli a feladat fent megadott specifikációját, és meghatározott módon megjelöli és minősiti a kulcsszavakat; pl. file, RÉGI, rekord, stb. A minősités azt jelzi, hogy a kulcsszó "objektum", "viszony", vagy "mozgásséma"-e. A rendszer a későbbiekben ellenőrzi, hogy ezeket a kulcsszavakat, ill. a kapcsolatukat az egyes szintek kialakitásánál definiáltuk-e:]
G: 1. szint következik.
Milyen objektumokat ismersz?
F: File, proc.
G: Sorold fel elemeket tipus szerint!
F: File: x,y,z,RÉGI,MOD,UJ; proc: ERROR.
G: Milyen állitás igaz a file-ra?
F: Üres, nemüres, rendezett, összefésült.
G: Mi a kapcsolat az "üres" és "nemüres" között?
F: axioma: file /x/ $\Longrightarrow$ üres /x/ $\oplus$ nemüres /x/
stb.

Befejezés, korlátok

A bemutatott módszernek nyilvánvalóan korlátai vannak. Egy bizonyos határon túl a formulák bonyolultakká válnak; az algoritmus kifejlesztése az axiomák alapján nem mindig kézenfekvő. Ezért a módszer alkalmazásának igazi helye az alapdokumentumként kezelendő specifikációk kialakitásában és a programtervezés első fázisában van.

Az alábbi, piramis-szerü sematikus ábrán bemutatjuk a program tervezésének és kivitelezésének logikai szintjeit, s megjelöljük az ismertetett módszer és a többi, szerepet játszó nyelv alkalmazási területeit.



A rendszer továbbfejlesztéseként a rendszert egy primitiv következtetőrendszerrel /kalkulussal/ kivánjuk ellátni; fel akarjuk készíteni induktiv állítások generálására, ill. dinamikus ellenőrzésre egy tesztelő rendszer keretén belül.

Irodalomjegyzék

Dijkstra, E.W.: Notes on Structured Programming. Academic Press, 1972. pp. 1-82.

Hoare, C.A.R.: The Axiomatic Basis of Computer Programming. Comm. ACM, Vol. 12, pp. 576-583, 1969.

Jones, C.B.: Formal Development of Programs. IBM Hursly report TR12 117, June 1973.

Jones, C.B.: Program Specifications and Formal Development. Manuscript. 1976.

Hoare, C.A.R.: Proof of Correctness of Data Representations. Acta Informatica, Vol. 1, pp. 271-281, 1972.

SAM-IV Dömölki B., Farkas Zs. és Sántáné-Tóth E.: Software elemek formális leirásának egy módszeréről. SzÁMKI 1696/76. 3. kötet, Budapest, 1976.

AZ EGYSZERŰ MAKROPROCESSZOR HATÁSAI EGY KISEBB TELJESITMÉNYŰ PROGRAMOZÁSI RENDSZERBEN

Áts László
"Pollack Mihály" Müszaki Főiskola

Bevezetés

A "Pollack Mihály" Müszaki Főiskolán egy EMG 830/20 tipusu számológép áll az oktatás szolgálatában. A gép egy szervezési intézettel közös tulajdont képez és az illető intézet üzemelteti. A Főiskola gépidő felhasználását tartalmi oldalról nem az üzemszerü feldolgozások, hanem programfejlesztési tevékenység relativ tulsulya jellemzi.

Néhány évvel korábban megnövekedett programozási feladataink arra kényszeritettek bennünket, hogy eszközt és módszert keressünk munkánk megkönnyitésére. Különféle lehetőségeket mérlegelve egy makroprocesszor kifejlesztése mellett döntöttünk: számológépünk SIMPLE 71 assembleréhez egy meglehetősen egyszerü preprocesszort készitettünk. Saját magunk számára is meglepő volt az a minőségi ugrás, amit a makroprocesszor felhasználása eredményezett. Javult a programok szerkezete olajozottabbá vált a programozók közti kooperáció. A programok belövési ideje nagymértékben csökkent.

Programozási környezet

Az általunk kidolgozott makroprocesszor az EMG 830/20 tipusu számológépeken általánosan használt programozási rendszerben funkcionál. Ez a rendszer tartalmazza az input - output rutinokat, a SIMPLE 71 assemblert, amely a tárba fordit fixcimes formában. Tartalmaz ezen kivül egy un. stop rutint és egy monitort a programbelövés segitésére, továbbá különféle utility

programok állnak rendelkezésre. Az alapszoftvert képező programok nagyobb része mágneslemezről hivható a konzol irógépen keresztül. A rendszer lyukszalag orientált. Nem tartalmaz magasabb szintü programozási nyelvet, a felhasználói programok nem irathatók fel programkönyvtárba.

A programfejlesztési munkát könnyitő rutinok részben tul kevés /stop rutin/, részben tul sok /monitor/ információt nyujtanak a program müködéséről, s a programozót géptermi jelenlétre, s egyben teljes szellemi "ottlétre" kényszeritik, amely egyrészt nagyon megnehezíti, hogy egyidőben több programmal foglalkozzon, másrészt gépidőpocsékoláshoz vezet. Programbelövés közben előforduló gépkezelői hiba, de szoftver és hardver hiba sem elhanyagolható gyakoriságu.

A makroprocesszor tervezési kérdései

A programfejlesztési munka hatékonyságát mindenképpen fokozni kellett. Mivel kézenfekvő volt, hogy az adott programozási rendszerből nincs módunk kilépni, arra a következtetésre jutottunk, hogy a munkafolyamat legrosszabb hatékonyságu fázisát, a programbelövést kell ellátni jól használható eszközökkel. Ezen az uton jutottunk arra a gondolatra, hogy makroprocesszort készitünk. A következő feladatokat kivántuk megoldani a makroprocesszor segitségével:

1. a programbelövésben használható, un. belövési makrókat definiálni, ill. lehetővé tenni a programozó számára, hogy igényei szerint definiáljon ilyen makrókat és elhelyezze a programjában.

2. lehetőséget teremteni általános célu, ill. termatikus makrókönyvtárak létesitésére.

A makroprocesszort - az implementálâshoz szükséges munka és idő mennyiségét csökkentendő - minél egyszerübben akartuk kialakitani, mellőzve a nélkülözhető szolgáltatásokat. A makrót hivását megelőzően definiálni kell, ez a makrónév és a skeleton megadásával történik. Makróidős aritmetika és feltételes kiterjesztés nincs. Egy makró legfeljebb kilenc paraméterrel rendelkezhet, hiányparaméter nem alkalmazható. A makródefiníciókat a makróprocesszor vermeli, igy bármely makró ujradefiniálható. A kiterjesztés fő mozzanatainak sorrendje a következő:

1. Az aktuális paraméter kidolgozása
2. cimkeparaméter generálása
3. az utasitásnév vizsgálata

Ez a sorrend lehetővé teszi, hogy aktuális paraméterként utasitásnév /vagyis esetleg makrónév/ is szerepelhet. Skatulyázott definició nem fordulhat elő, de skeletonban állhat makróhivás, ha ez nem vezet rekurzióra.

Ugyancsak az egyszerü és gyors implementálás érdekében döntöttünk ugy, hogy preprocesszort készitünk, igy nem kellett bolygatnunk a meglévő programozási környezetet. Emiatt a makroprocesszor a definiciókat és a makróhivásokat tartalmazó programot is lyukszalagról várja. Az input akárhány lyukszalagon állhat, az utolsó definiciónak nem kell szükségképpen megelőzni a program első utasitását. A makróprocesszor listát nyomtat a kiterjesztett programról, ez az assembler által megkövetelt szintaxisu, ugyanugy, mint az egyidejüleg készülő lyukszalag, amely az assemblerrel lefordítható.

Belövési makrók az utasitások ujra definiálása

A makroprocesszor segitségével beépithetők a programba olyan speciális makrók /belövési makrók/, amelyek szerepe a belövés megkönnyitése. Sokféle változatuk képzelhető el, közös vonásuk, hogy a programozó által előre meghatározott

pontokon ellenőrző információt adnak a meghatározott programváltozók állapotáról. Az igy nyert viszonylat rövid lista segitségével a hiba gyorsan behatárolható és kijavitható. A javitást forrásnyelvi szinten kell elvégezteni. Mihelyt a program belövésével végeztünk, a belövési makrót ujradefiniálva eliminálhatjuk.

Előfordult olyan eset, hogy a programozó nem élt a belövési makrók beépitésének lehetőségével, de a belövés során sulyos nehézségek merültek fel. Mivel a forrásnyelvü szalag javitása eléggé nehézkes, belövési makrók beirása helyett valamely assembler utasitást definiáltunk makróként ujra. A rekurziót elkerülendő, ügyelni kellett, hogy a skeletonban az ujradefiniált utasitást ne szerepletessük.

Egy más lehetőséget adott tetszésszerinti utasitás ujradefiniálására a makroprocesszor és az assembler egy müködésbeli különbsége. Az assembler terminátorok segitségével ismeri fel a nyelv szintaktikus elemeit, igy az utasitás nevét, az irásmód kötetlen. A makroprocesszor ezzel szemben a sor nyolcadik pozicióján keresi a makrónév első karakterét. Igy a skeleton tartalmazhatja az ujradefiniált utasitást, mindössze pl. a kilencedik sorpozición kell kezdeni.

Utasitások ujradefiniálása nem egyszer olyan esetekben nyujtott segitséget, amelyekben utólag sem tudunk gyorsabb megoldást javasolni a hiba feltárására. Volt eset amikor olyan hardver hibát találtunk, amely a használatos tesztekkel kimutathatatlan volt.

Makrókönyvtárak létesitése

Mint a legtöbb programozási nyelv,a SIMPLE 71 assembler

sem mentes olyan nyelvi konstrukcióktól, amelyek használata kényelmetlen, nagyobb odafigyelést igényel, hibalehetőséget rajt magában. A makroprocesszort felhasználtuk arra, hogy olyan makrókat definiáljunk, amelyek bizonyos meglévőknél természetesebb módon használhatók, vagy hiányzó lehetőségeket pótolnak.

Ilyan gondolatokkal fogtunk hozzá egy általános célu makrókönyvtár kialakitásához. E makrókönyvtárban olyan makrók kaptak helyet, amelyek témakörtől függetlenek, bármely program irásánál egyszerüsitik a programozó munkáját. Mágnesszalagos müveletek, nyomtatóval kapcsolatos tevékenységek nyertek igy pl. logikusabb alakot.

Definiálásra került ezen kivül egy matrixaritmetikai makrógyüjtemény, továbbá bővitettük a SIMPLE 71-ben használatos függvények körét.

Tapasztalatok

A rendelkezésre álló uj lehetőségek lassan megváltoztatják a programozók munkastilusát. Megnyilvánul ez abban, hogy a kialakitott makrókönyvtárak felhasználást nyernek és továbbfejlődnek. A makrókönyvtárak és általában a makrósitás lehetősége becslésünk szerint a SIMPLE 71 nyelvü programok megirásában 20-30 %-os munka- és időmegtakaritást tudtunk elérni. A "kellemetlen" utasitások eliminálása nyomán - de a makrók alkalmazása önmagában is ebben az irányban hatott - nőtt a programok áttekinthetősége, strukturáltsága. Ez önmagában is gépidő megtakaritást eredményez.

Változott a programbelövés folyamata is. Korábban a programozó órákat töltött a gépteremben programbelövés ürügyén. Jelenleg erre semmi szükség. Sőt az üzemeltetési rend szerint igényelhető legrövidebb gépidő /fél óra/ is hosszunak bizonyult egy program, ill. programozó számára. Mivel a programhibát nem a gépteremben kell felderiteni, lehetőség nyilik arra, hogy egy időben több programmal foglalkozzon a programozó, ill. hogy egy félórás gépidőt többen vegyenek igénybe.

Ez ismét jelentős mennyiségü gépidő megtakaritást jelent. Mivel pedig a hibákat könnyebb megtalálni, ezért egy program kevesebbszer is kerül belövés ürügyén a gépterembe. Óvatos becslésünk szerint mindez azt eredményezi, hogy programbelövés során a gépidő megtakaritás legalább 50 %-os.

Áts László
főiskolai adjunktus

PÁRHUZAMOS FOLYAMATOKAT KEZELŐ OPERÁCIÓS RENDSZER

Dr.Bach Iván
tudományos osztályvezető
MTA Számitástechnikai és Automatizálási
Kutató Intézet

A korszerü számitógépes gyakorlatban sokszor van igény olyan dedikált operációs rendszerre, amely valamilyen integrált rendszer számitógépes kiszolgálásának alátámasztására hivatott. Ezek, az általában mai fogalmaink szerint kisszámitógépre készült operációs rendszerek azzal jellemezhetőek, hogy több párhuzamos folyamat kiszolgálását kell ellátniuk - legtöbbször valós idejü müködési módban - és felépitésüket tekintve egyediek, szolgáltatásaikban az adott feladat igényeihez idomulnak.

Felmerült az a gondolat, hogy az ilyen jellegü rendszer alkalmazását és készitését megkönnyitendő olyan operációs rendszermagot definiáljunk, amelyre az ilyen operációs rendszerek ráépithetőek. Amennyiben törekvésünket siker koronázza, akkor az ezen elven felépitett operációs rendszerek egységesebb szerkezetüek lesznek, amely megnöveli azok portabilitását és megkönnyiti implementálásukat. Ezzel a kérdéssel foglalkozik a Purdue Europe nemzetközi szervezet egyik munkabizottsága, amelynek a szerző is tagja. Az alábbi dolgozat messzemenően felhasználja azokat az elveket és megállapitásokat, amelyek ott közös munka eredményeképpen kidolgozásra kerültek.

Célkitüzés és a környezet leirása

A dolgozat olyan könnyen implementálható elveket és ezek alapján készült eszközöket kiván ismertetni, amely alkalmas párhuzamos és egymással kommunikáló folyamatokat kezelő operációs rendszer kialakitására. Az elvek és esz-

közök kidolgozásánál szem előtt tartottuk azt a követelményt, hogy azok nemcsak kváziparallel, hanem fizikailag parallel folyamatok esetében is használhatóak legyenek. A fizikailag párhuzamos müködés feltételezése egyes funkciók megvalósitását lényegesen bonyolultabbá és nehezen kezelhetővé teszi. Amennyiben csak kváziparallel folyamatokról van szó, ott sokszor egyszerüsitésekre nyillik mód. Erre néhány helyen rá is fogok mutatni.

Az általunk tekintett rendszer folyamatokból és processzorokból áll. A folyamat szekvenciálisan végrehajtható tevékenységek sorozata. A processzor az az elem, amely a folyamatok által definiált tevékenységek végrehajtására képes. Egy processzor egyidejüleg csak egy tevékenység végrehajtásán munkálkodhat.

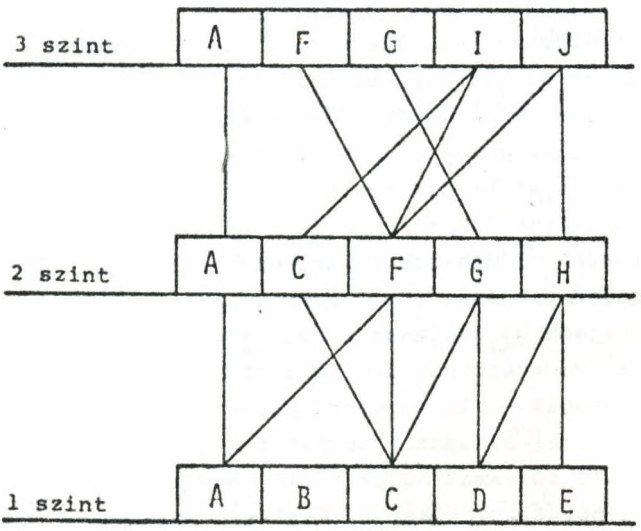
A rendszerben lévő processzorok egymástól különbözhetnek, ami annyit jelent, hogy az általuk végrehajtott tevékenység eltér és ennek következtében utasitáskészületük nem azonos /pl. perifériakezelő processzor, központi processzor/. A rendszerben lehetnek azonos processzorok is, amelyek processzorhalmazt alkotnak, ezekről viszont feltételezzük, hogy egymás között teljesen felcserélhetőek, vagyis a halmaz egyik elemét alkotó processzor által végrehajtható minden tevékenység végrehajtható a halmaz másik processzorán is. Ennek megfelelően utasitáskészletünk is azonos.

Feltételezzük, hogy a folyamat által előirt minden tevékenység egy és csakis egyfajta processzor utján hajtható végre. Az az eset tehát, amikor két központi processzor a memóriának különböző részeihez tud csak hozzáférni, de a két memóriarész metszete nem üres, nem illik bele modellünkbe.

A rendszer megvalósitása, folyamat és processzor állapotok

Elképzeléseink szerint a rendszer magját alkotó software rétegezett felépitésü. A hardware, vagyis a számitógép a legalsó réteg alatt helyezkedik el. Az egyes rétegeket alkotó software különböző funkciókat valósit meg, és minden magasabb hierarchiáju réteg felhasználhatja az alatta lévő réteg függvényeit, vagy változatlanul átveheti azokat.

Ez az egyébként ismert struktura bizonyos modularitást biztosit és ezen kivül lehetővé teszi a rendszer magjának valamilyen határok közötti szabad megválasztását olyan értelemben, hogy egy adott alkalmazásnál bizonyos, számunkra érdektelen funkciók megvalósitásától eltekinthetünk.



Hardware

1.ábra

A felhasználó operációs rendszerę a legfelsőbb réteghez csatlakozhat. A felhasználónak nem áll szükségképpen rendelkezésére valamennyi megvalósitott funkció, lehetnek olyanok, amelyeket csak a felsőbb szintek használnak és a felhasználó számára hozzáférhetetlen.

A dolgozat csak a rendszermagnak azzal a részével foglalkozik, amely az élő, a rendszerben aktiv folyamatokat kezeli.

Ebben a modellben mind a folyamatoknak, mind a processzoroknak három állapotuk lehetséges. Ezeket a 2.ábra mutatja.

Amennyiben egy folyamat *futó* állapotban van, akkor a folyamathoz egy processzor van hozzárendelve, amely a folyamat által előirt tevékenységeket végrehajtja. *Várakozó* állapotban a folyamatnak a processzoron kivül minden erőforrás rendelkezésére áll és csupán processzorra vár, hogy tevékenységét folytassa. *Blokkolt* állapotban a folyamat a processzoron kivül valamilyen más általános értelemben vett erőforrás rendelkezésre állására várakozik és nem vesz részt az esetlegesen szabad vagy felszabaduló processzorok hozzárendeléséért folyó "küzdelemben"

Hozzárendelt állapotban a processzor valamilyen folyamat tevékenységét hajtja végre.

Szabad állapotban a processzor semmilyen tevékenységet nem folytat. A processzor akkor kerül *ébresztett* állapotba, ha a processzorra vagy a processzor fajtára várakozó folyamatok listájába uj folyamat kerül, és rendszerünk preemptiv. Ilyenkor a processzornak meg kell vizsgálnia, hogy az ütemezési stratégia - pl. egyszerü prioritás - alapján nem az ujonnan *várakozó* állapotba került folyamatot kellene-e jelenleg *futó* folyamat helyett előnyben részesitenie.

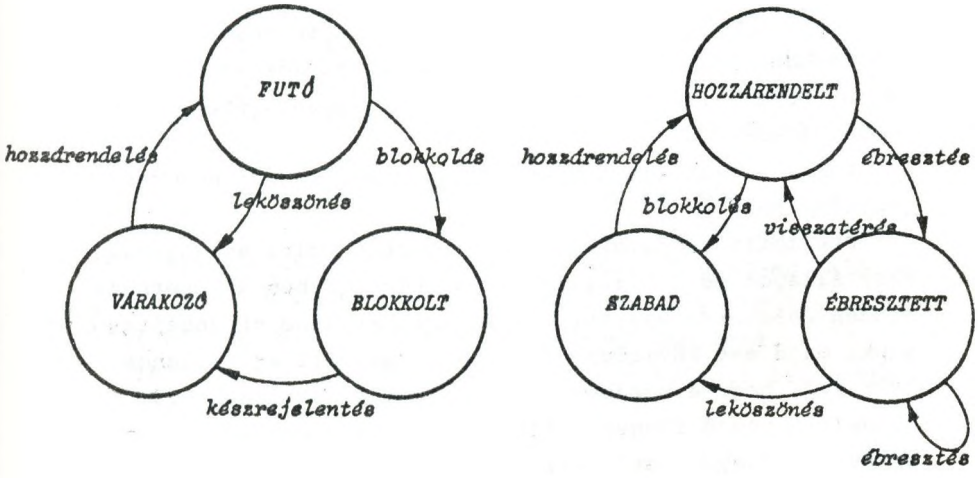
Az állapotváltozásokat a következő primitivek biztositják.

HOZZÁRENDELÉS: Ez a funkció egy *várakozó* folyamathoz egy *szabad* processzort rendel, a folyamatot a *futó*, a processzort pedig *hozzárendelt* állapotba viszi át.

LEKÜSZÖNÉS: Ez a funkció az *ébresztett* processzort leválasztja a *futó* folyamatról és a folyamatot a *várakozó*, a processzort pedig *szabad* állapotba viszi át.

BLOKKOLÁS: Ez a funkció a *futó* folyamatot *blokkolt* állapotba viszi át és ennek megfelelően a hozzárendelt processzor a *hozzárendelt* állapotból *szabad* állapotba kerül.

KÉSZREJELENTÉS: Ez a funkció a *blokkolt* folyamatot *várakozó* állapotba teszi.



2.ábra

ÉBRESZTÉS: Ez a funkció a processzort - általánosabb esetben a processzor halmaz egy elemét - *ébresztett* állapotba viszi. Ebben az állapotban kellett döntenünk afelől, hogy az ébresztést kiváltó ok szükségessé teszi-e az érvényes hozzárendelések megvalósitását. Amennyiben a döntés pozitiv, *leköszönés* majd *hozzárendelés* következik be, amennyiben a döntés negativ, ugy a *visszatérés* függvényt kell végrehajtani.

VISSZATÉRÉS: Ennek a funkciónak a hatására az *ébresztett* processzor visszatér a *hozzárendelt* állapotba.

Ezek a vázolt - itt korántsem teljes részletességgel kidolgozott - primitivek arra az esetre érvényesek, amikor a rendszerben preemciót alkalmazunk. Amennyiben rendszerünkben preemció nincs, akkor elmarad a processzoroknál az *ébresztett* állapot és nincs szükség az *ébresztés*, *leköszönés* és *visszatérés* primitivekre sem.

Ezen függvények végrehajtása kapcsán természetesen gondoskodni kell a folyamatok és az operációs rendszer magját képező függvények kontexusának kimentéséről, illetve betöltéséről.

Ennek részleteit itt nem tárgyaljuk, csupán egy megjegyzést teszünk.

Preemptiv rendszernél is elhagyható volna az *ébresztett* állapot és a *visszatérés* függvény. Ebben az esetben minden *ébresztés* helyett *leköszönést* kellene végrehajtanunk, majd ezt követően *hozzárendelést*. Itt arról lehet csak szó, hogy a korábban *futó* folyamat kerül a hozzárendelést végző függvény által ujra hozzárendelésre. Ez általában nagyobb erőfeszitést igényel, hiszen a folyamatok kontexusát minden alkalommal folyamatkontexussal kell kicserélni, mig az *ébresztett* állapot bevezetésével a csere a kontexusnak csak arra a részére szoritkozik, amelyet a processzor mag függvényei igényelnek, amely általában kisebb.

Szinkronizáció

A párhuzamos folyamatokat kezelő rendszer legfontosabb feladata a folyamatok közötti szinkronizáció lehetővé tétele. Ennek egyik fontos feltétele a kölcsönös kizárás és egyes funkciók oszthatatlanságának biztositása.

A kölcsönös kizárás egyik leggyakoribb formája pl. az az eset, amikor a memória bizonyos részéhez csak egyetlen feladat kapcsán nyulhatunk hozzá. Tipikusan ilyen az előbb felsorolt és az állapotokat megváltoztató primitivek esete. Természetes, hogy amig egy ilyen primitiv végrehajtása kapcsán valamennyi állapotátmenet könyvelését nem fejeztük be, addig ezekről semmiféle információ szolgáltatása nem engedhető meg, hiszen az átmenet során inkonzisztens állapotok is előfordulhatnak. A korszerü gépeknél ezen követelmények kielégitését bizonyos hardware utasitások támogatják /ilyen pl. a TEST and SET/. A továbbiakban feltételezzük, hogy ilyen hardware utasitások lehetőséget adnak két belső primitiv megvalósitására. Ezek a "*Zár*" és "*Nyit*" primitivek.

Zár (V): Amennyiben a V változó nyitott állapotban volt, ugy azt lezárja és a processzor folytathatja tevékenységét. Amennyiben a V változó zárt állapotban találtatott, akkor a processzor mindaddig nem folytathatja munkáját, amig a változó nyitott állapotban nem lesz.

Nyit (V): A zárt állapotu V változó nyitott állapotba kerül.

Ezen bevezető után ismertetjük azt az eszközt, amelyek a folyamatok közötti szinkronizálást megvalósitják. Feltételezzük, hogy ahol szükséges, alkalmas változóra *zár* illetve *nyit* utasitást adunk ki a kölcsönös kizárás biztositására. A szinkronizálás céljaira szinkronizáló elemek szolgálnak. Feltételezzük, hogy az egyes folyamatok más folyamatoknak üzeneteket kivánnak átadni.

Itt feltételezzük, hogy az üzenetek annyiban egyenértékűek, hogy egy szinkronizáló elemnek átadott üzeneteket

- bármely folyamat szolgáltatta is azt - bármely ehhez a szinkronizáló elemhez üzenetért forduló folyamat fel tud használni.

A rendszerben lévő szinkronizáló elemek száma tetszőleges. Praktikusan a teljesen általános szinkronizáló elem alkalmazásaira általában nincs szükség. Amennyiben mindig A folyamat küld B folyamatnak üzenetet, akkor olyan szinkronizáló elemet kell létesiteni, amely A folyamat üzeneteit fogadja és B folyamatnak kiszolgáltat üzeneteket. Több termelő és egy fogyasztó esetén az A1, A2, ... An folyamatok bármelyike szolgáltathat üzenetet, de csak B folyamat fordulhat a szinkronizáló elemhez üzenet elvétele céljából. Hasonlóképpen oldható meg az egy termelő több fogyasztó esete.

A szinkronizáló elem lényegében egy számlálóból és egy listából áll. A listában vagy a feldolgozásra váró üzenetek az üzenetre várakozó folyamatok helyezkednek el. Valahányszor egy folyamat akkor kér üzenetet amikor üzenet rendelkezésre áll, a folyamat az üzenetet megkapja és az üzenetet a listából törli. Valahányszor egy üzenetet szolgáltatunk a szinkronizáló elemnek, amikor van olyan folyamat, amely üzenetre vár, egyik folyamat az üzenetet megkapja, és a folyamatot a listából töröljük. Könnyü belátni, hogy a kétféle - üzenetek és folyamatok - lista közül legfeljebb csak az egyik élhet. Éppen ezért elegendő egyetlen lista.

A számláló abszolut értéke a lista hosszát jelöli, mégpedig pozitiv értéknél üzenetek, negativ értéknél folyamatok állnak a listában.

A szinkronizálásra két funkció szolgál. Mindkettőnek két paramétere van, az egyik a szinkronizáló elem, a másik az üzenet.

INKREMENTÁLÁS (szinkronizáló elem, üzenet):

Ennek hatására a szinkronizáló elem számlálója kerül vizsgálatra. A számlálót 1-gyel megnöveljük és amennyiben ennek értéke pozitiv /vagyis nem volt üzenetre várakozó folyamat/ az üzenetet a listába felfüzzük. /Célszerüen csak az üzenetre mutató pointert./ Amennyiben volt várakozó folyamatunk, azok közül kiválasztunk egyet, az üzenetet a folyamatnak átadjuk és erre a folyamatra végrehajtunk egy *készrejelentés* primitivet. Ezzel ez a folyamat *blokkolt* állapotból a folyamat kontexusában meghatározott processzor fajtára *várakozó* folyamatok sorába kerül.

DEKREMENTÁLÁS (szinkronizáló elem, üzenet):

A második paraméter azt jelzi, hová várja a folyamat az esetleges üzenetet. A számlálót 1-gyel csökkentjük és megvizsgáljuk, vajon negativ-e /ami annyit jelent, hogy nem állt üzenet rendelkezésre/. Ez esetben a folyamatot felfüzzük a listába és végrehajtunk egy blokkolás primitivet. Amennyiben van eddig fel nem használt üzenet, az üzenetet a folyamatnak átadjuk.

Nyilvánvaló, hogy a kényes lépések során gondoskodni kell a kölcsönös kizárásról. Amennyiben rendszerünk csak kváziparallel, akkor csak a megszakitási rendszer okozhat illegális hozzáférést. Ilyenkor a megszakitások letiltása elegendő a kölcsönös kizárás biztositásához.

A vázolt módszerrel a párhuzamos folyamatok kezelésének szokásos eszközei előállithatók. Utalunk itt a szemafor megvalósitására.

Ennek érdekében minden szemaforhoz egy szinkronizáló elemet kell rendelnünk és mivel ebben az esetben információ átadására nincs szükség, az információra mutató pointer NIL lesz.

V (szemafor) = *inkrementálás* (szink.elem, NIL)
P (szemafor) = *dekrementálás* (szink.elem, NIL)

nyilván megvalósitja a szemafor funkcióit.

Természetesen, ha csak szemaforokra van szükségünk, akkor a fenti függvények közvetlenül is megvalósithatók.

A VT50 ÜGYVITELI KISSZÁMITÓGÉPCSALÁD OPERÁCIÓS RENDSZERE

Baffia László, Bontovics Mária, Kovács Tamás, Varga Éva
VIDEOTON Fejlesztési Intézet

Bevezetés

A VT50 ügyviteli kisszámitógépcsalád kifejlesztésének célja egy olyan olcsó adatfeldolgozó gép létrehozása volt, amely adottságainál fogva a billentyüzettorientált /adatrögzitéssel egybekötött/adatfeldolgozásra alkalmas, s igy ujszerü megoldást nyujt.

- kis vállalatok vagy nagy gazdasági egységek részlegei adatfeldolgozási feladatainak komplex megoldására,

- decentralizált adatfeldolgozó rendszerek megvalósitására.

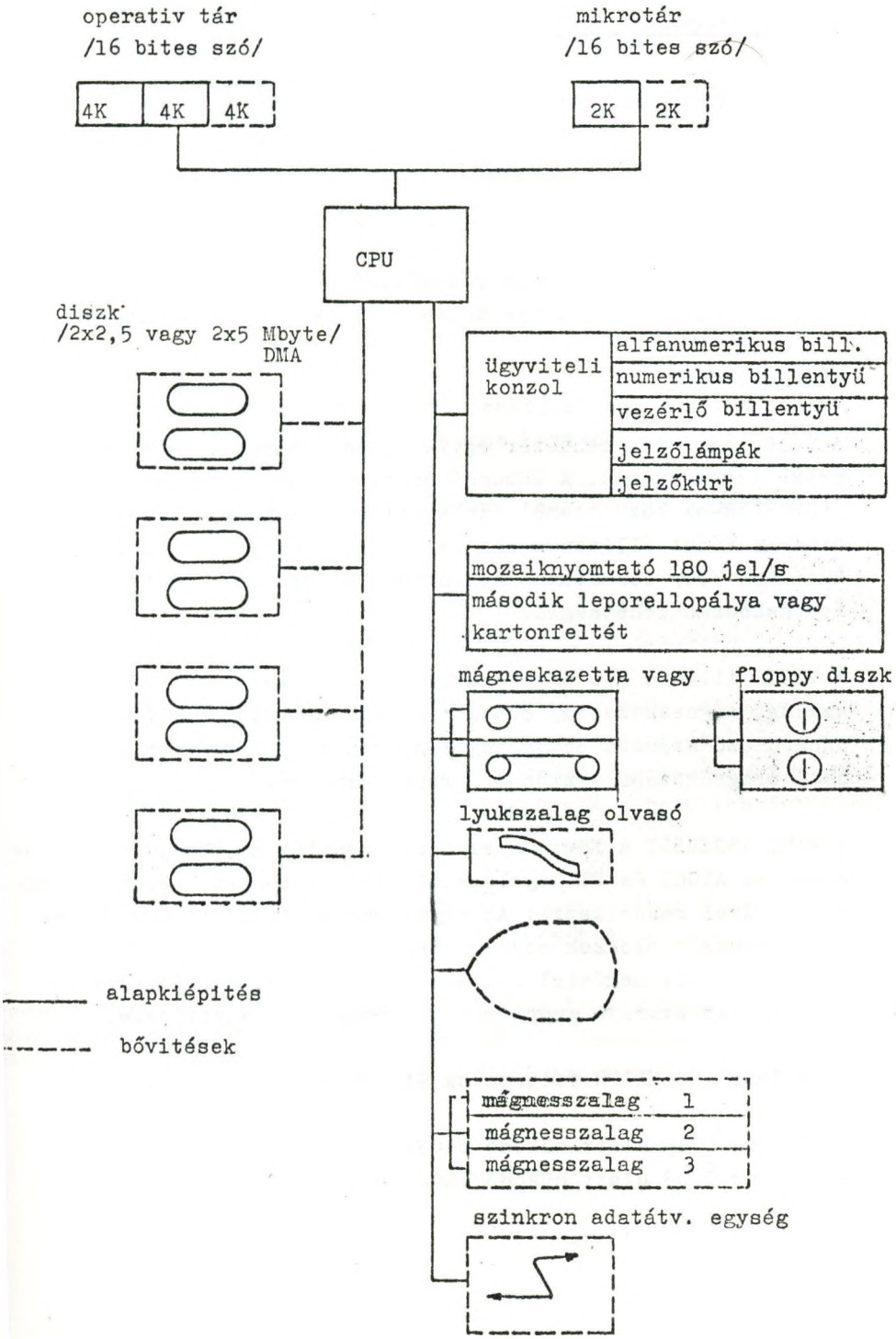
Az utóbbi esetben a gép vagy további VT50-es gépekkel, vagy valamilyen idegen géppel müködhet együtt.

1./ Hardware

A VT50 ügyviteli kisszámitógépet maximális kiépitettségében a következő ábra szemlélteti.

A gép utasitáskészlete mikroprogramozott. Az utasitások között szerepel a négy matematikai alapmüvelet 14 jegyü decimális számokkal, valamint néhány karakterkezelő müvelet.

Az alapkiépitésü gép jelölése VT51, mig a diszk-kel kiegészitett gép a VT54-es modell.


operativ tár
/16 bites szó/
4K
4K
4K
mikrotár
/16 bites szó/
2K
2K
CPU
diszk
/2x2,5 vagy 2x5 Mbyte/
DMA
ügyviteli konzol
alfanumerikus bill.
numerikus billentyü
vezérlő billentyü
jelzőlámpák
jelzőkürt
mozaiknyomtató 180 jel/s
második leporellopálya vagy
kartonfeltét
mágneskazetta vagy
floppy diszk
lyukszalag olvasó
alapkiépités
bővitések
mágnesszalag 1
mágnesszalag 2
mágnesszalag 3
szinkron adatátv. egység

2./ Programozási nyelvek

A VT50 gépcsalád két nyelven programozható:

- ASSEMBLER
- RPG II

Megfelelő forditók állnak rendelkezésre R10 számitógépen /Cross compiler/, illetve megfelelő kiépitettségü VT50-en.

2.1 ASSEMBLER

A VT50 operációs rendszer egyik fő sajátossága, hogy mindössze három szintü. A középső szintről /assembly/ az ASSEMBLER-ek közvetlenül tárba tölthető nem relokálható bináris kódot állitanak elő /a legfelső szint az RPG II/. Közbenső /BT/ formátum tehát nincs. Mindennemü szerkesztés forráskódban lehetséges.

A VT50, illetve R10 nagykapacitásu háttértárai /diszk, mágnesszalag, mágneskazetta/ ezt a megoldást lehetővé teszik, a programozó számára pedig igy lényegesen egyszerübben kezelhető programozási eszköz áll rendelkezésre.

A VT50 ASSEMBLY a forrásnyelvi szerkesztés megkönnyitése érdekében az ALGOL és PL1 nyelveknél használatoshoz hasonló blokkstukturával rendelkezik. Az egymásbaskatulyázott vagy egymás mellé rendelt blokkok száma 256-nál nem lehet több. Az ASSEMBLY nyelv megfelelő direktivákat tartalmaz, továbbá overlay strukturáju programok irásának megkönnyitésére.

Az alábbi ASSEMBLER változatok állnak rendelkezésre:

- R10-en OS10 alatt müködő CROSS-ASSEMBLER
- R10-en IDOS alatt müködő CROSS ASSEMBLER

- VT51-en müködő ASSEMBLER
  /Kazettáról vagy lyukszalagról kazettára fordit. A szimbolumtáblázatot az operativ tárban létesiti, ezért 8K operativ tár mellett csak korlátozottan használható./

- VT54-en müködő ASSEMBLER
  /Diszkről diszkre fordit, a szimbolumtáblákat is diszken tárolja. Gyakorlatilag korlátlanul használható./

2.2 RPG II

AZ RPG II nyelvü programozást lehetővé tevő eszközök ugyancsak két változatban állnak rendelkezésre:

- R10-en müködő RPG II rendszer, mely két fő komponensből áll:

  - RPG II interpreter, mely R10-en lehetővé teszi RPG II programok futtatását tesztadatokkal /belővés/

  - RPG II fordító, mely VT50 ASSEMBLY kódot állit elő, s igy lehetővé teszi ASSEMBLY rutinok beépitését RPG II programokba. A fordítás során keletkező forrásanyagból, valamelyik ASSEMBLER-rel állitható elő a betölthető bináris program.

- VT54-en müködő RPG II fordító. A fordító az R10-en futó változathoz hasonlóan ASSEMBLY kódot állit elő.

  Tekintettel arra, hogy a VT50-hez lyukkártyaolvasó nem illeszthető, egy RPG II EDITOR áll rendelkezésre, mely display-en lehetővé teszi RPG II programok rögzitését és javitását.

3./ Filekezelő rendszerek

A VT50 operációs rendszerében két filekezelő áll rendelkezésre:

- szekvenciális filekezelő /mágneskazettán, mágnesszalagon és floppy diszken/ tárolt adatok kezelésére

- diszkes filekezelő.

3.1 Szekvenciális filekezelő

A szekvenciális filekezelő az ISO/4341 szabvány "Compact System" strukturájának megfelelő /fejcimke 32 byte/ fix hosszuságu blokkositott vagy blokkositatlan /max. blokkhoz 256 byte/ többkötetes adatállományok kezelését teszi lehetővé szekvenciális adathordozón. /A floppy diszket szándékosan szekvenciális adathordozóként tekintjük./

3.2 Diszkes filekezelő rendszer

A diszkes filekezelő rendszer a VT54 monitorok szerves része. Jelentőségére való tekintettel egy külön előadás ismerteti. A diszkes filekezelő szempontjából egy file egy megadott diszk terület, mely bizonyos feltételek mellett bővithető, s mely rendszerprogramot, felhasználói programot, forrásszöveget, illetve fix/változó /blokkositott/blokkositatlan/ rekordokból álló adatot tartalmazhat.A felhasználó a diszket csak a filekezelőn keresztül használhatja.

4./ Monitorok

Két alapvetően különböző monitor áll a VT50 gépeken rendelkezésre:

- kazettás változat /VT51 monitor/
- diszkes változat /VT54 monitor/

Mindkét változat tartalmazza a felhasználó által igénybevehető főbb perifériák handler-jeit.

4.1 VT51 monitorok

A VT51M- monitorok bináris programok töltését mágneskazettáról teszik lehetővé. Két változatban állnak rendelkezésre:

- alapváltozat programok üzemszerü futtatására,
- diagnosztikai változat programbelövést elősegitő szolgáltatásokkal.

4.2 VT54 monitorok

A VT54M- diszkes monitor overlay strukturáju bináris programok töltését és futtatását teszi lehetővé diszkről.

Rendelkezik minden lényeges diagnosztikai szolgáltatással.

Bizonyos, a felhasználói program által ritkán igényelt terjedelmes monitormodulokat swappingelten kezel. A monitor operátori kommunikációval kapcsolatos része overlayezett. Igy helyfoglalása nem több 3.5K-nál.

5./ Utility-k

5.1 Kazettás utility-k

- BIB bináris könyvtárkezelő
- LIST kazetta listázó
- PRCOPY bináris programot lyukszalagról kazettára másoló
- SORT szekvenciális rendezőprogram /két egységgel müködik/.

5.2 Diszkes rendszert támogató utility-k

- SYSGEN rendszergeneráló
- KATINI katalógus inicializáló
- LISTD diszk katalógus listázó

- RENAME file átnevező
- DELET file törlő
- REGEN diszk ujraszervező

- COPY bináris programokat, forrásszövegeket és fix/változó hosszuságu rekordokból álló adatállományokat különböző perifériákról diszkre /és viszont/ másoló.

5.3 Diszkes rendezőprogram

A diszkes rendezőprogram diszkről diszkre rendez. A rekordon belül maximum 9 kulcs jelölhető ki. A rendező program speciális üzemmódban lehetővé teszi:

- ADDROUT file létrehozását
- indexelt szekvenciális szervezésü file létrehozását
- indexelt szekvenciális szervezésü file ujraszervezését.

5.4 Adattárkezelő programcsomag

Az adattárkezelő programcsomag két változatban áll rendelkezésre:

- szekvenciális adathordozókon müködő
- diszkes rendszerben müködő

Az adattárkezelő programcsomag formátumozott szekvenciális szervezésü adatállomány kezelésével kapcsolatos alapmüveletek végrehajtását teszi lehetővé programozás nélkül, mint:

- adatállomány rögzitése,
- adatállomány listázása vagy megjelenitése /esetleg kulcshatárokon belül, illetve egyedi rekord lekérdezése/,
- adatállomány másolása,

- rendezett adatállományok egybeválogatása,
- adatállomány mezőinek módositása.

Az adattárkezelő programcsomag mozgásállományok rögzitését és kezelését, valamint törzsadatállományok teljes kezelését mindenféle programozás nélkül lehetővé teszi, s ezzel adatfeldolgozó rendszerek létrehozásánál jelentős programozói munka takaritható meg.

ADATFELDOLGOZÓ SZÁMITÓKÖZPONT ÜZEMELTETÉSÉT ELŐSEGITŐ PROGRAMFEJLESZTÉSI MUNKÁK

Bándy Imréné
FÜTI

Vállalatunk - a Fővárosi Épitőipari Üzemgazdasági és Ügyviteltechnikai Iroda - adatfeldolgozási rendszerek kifejlesztésével és üzemeltetésével foglalkozik. Számitógépparkunk gerincét egy ESZR-40-es és egy ESZR-20-as gép alkotja, melyeket DOS operációs rendszerekkel üzemeltetünk.

Operációs rendszereink generálásánál a következő jellemzőket kellett figyelembe vennünk:

- a géppark 24 órás folyamatos üzemben müködik,
- a gépidő felhasználásában jelentős tétel a programtesztelés,
- az adatfeldolgozási munka fontos részeként nagy volumenü és rövid határidős nyomtatási feladat jelentkezik,
- a gépi konfigurációkban a központi tár kapacitásához viszonyitva aránylag kevés a lemezkapacitás.

Az R-40-es gépre generált háromparticiós DOS felépitése például a következő:

- állandóan POWER II futtatás egy 60K-s F1 particióban,
- 210K-s F2 partició: elsősorban kitesztelt adatfeldolgozó rendszerek magán fáziskönyvtárból való futtatásához, másodsorban szerkesztési feladatok részére - magán modulkönyvtár segitségével,
- 218K-s BG partició: elsősorban magán forrás- és modulkönyvtárral programforditás és tesztelés számára, másodsorban állandó aktuális szerkesztéssel adatfeldolgozó rendszerek futtatásához.

A három particiót kiszolgáló rendszerlemez csak fázis-könyvtárat és munkaterületeket tartalmaz.

Az ESZR-40 és ESZR-20 általános rendeltetésü számitógépeken kivül a FÜTI gépparkjához még két kisebb off-line adatfelviteli és litálási feladatot ellátó gép tartozik: egy MDS 2400 tipusú és egy - hardware átalakitással és software támogatással erre alkalmassá tett - UNIVAC 1005-ös.

A gépparkhoz tartozó négy számitógép off-line együttmüködésének és hatékony üzemeltetésének biztositása több software és hardware problémát vetett fel. A software úton megoldott problémák közül az alábbi témakörökben megvalósitott fejlesztéseinket ismertetjük:
- az on-line és off-line nyomtatás,
- a mágneslemezcsomagok gazdaságosabb kihasználása és
- az adatfeldolgozási munkák job-kártyáinak feldolgozási időszak szerinti update-elése.

Számitóközpontunk jelenlegi feladatösszetételéből adódóan a határidős adatfeldolgozási munkák kulcskérdése a listálás-nyomtatás. A számitógépekkel való gazdálkodási lehetőség kiszélesitése érdekében elsődlegesen biztositottuk, hogy a listák a programokból választhatóan on-line vagy off-line technika szerinti formában boosáthatók ki.

A POWER segitségével és felügyelete alatt - lista outputra - a következő megoldásokat biztositottuk:
- a felhasználói programokból közvetlenül lehet nyomtatni;
- lehetséges a listák SPOOLING-ja mágneslemezre, vagy mágnesszalagra;
- végül az MDS vagy az UNIVAC kisszámitógépeken kinyomtatható listaállományokat is lehet készitení mágnesszagra.

Gondot jelentett a listák példányszáma; sornyomtatóink maximum három példányos nyomtatásra képesek. Bizonyos listákat azonban háromnál több példányban is el kell készitenünk. Ez lemezen spoololt és off-line nyomtatásra kerülő listáknál nem jelent problémát, viszont a POWER-rel kinyomtatásra kerülő szalagon spoololt állományokhoz a POWER-ben ilyen lehetőség nem volt. Ezért a * $$ PRT kártya formátum operandusának kezelésénél beépitettük az illetékes POWER-fázisba, hogy ezt az operandust ismétlési tényezőként lehessen használni.

A sornyomtatók kapacitásával való gazdálkodás a továbbiakban azt a problémát is felvetette, hogy egy, a fentiek valamelyike szerinti adott formátumban kibocsátott listaszalagot mégsem az eredetileg diszpécselt számitógépen és sornyomtatón kell kinyomtatni, hanem egy másikon. Ezt két irányban hidaltuk át, egyrészt segédprogrammal lehetővé tettük a POWER-SPOOL szalagok MDS vagy UNIVAC konvertálását, másrészt a POWER kiiró fázisának bővitésével biztositottuk, hogy MDS-re és UNIVAC-re készitett lista-szalagok POWER-ben is kiirhatóak legyenek.

A nyomtatás problémakörében felmerült egy kompatibilitási probléma is: az MDS berendezés közvetlenül alkalmas az R-40-es és R-20-as gépeken felirt lista-szalagok kinyomtatására, a UNIVAC 1005-ös gép hardware-jét viszont át kellett alakitani, és emellett az itt kinyomtatni szándékozott adatsorokat másképpen kell összeállitani, mint az R-40, R-20 és MDS adatsorokat.
A UNIVAC berendezésre kerülő lista-szalagok felirása során gondoskodni kell a megfelelő - a UNIVAC printeren értelmezhető - kód használatáról, mivel a konvertálás elvégzéséhez

a UNIVAC főtároló kapacitása kevés. A megkivánt kód biztositható lett volna a felhasználói programokból, viszont többszáz programot kellett volna kibőviteni és újrafordítani.
A kódkonverziót ezért a DOS logikai input/output vezérlő rendszer /LIOCS/ megfelelő moduljába helyeztük el, igy mentesitettük a felhasználói programokat ennek a problémának a kezelésétől.

A FÜTI tevékenysége során mintegy 500 mágneslemez-csomagot használ, amelyek együttes értéke körülbelül 5 millió forint. Ezt az erőforrást a már futó és évenként növekvő számban az új, induló adatfeldolgozó rendszerek használják.
Felvetődött az a kérdés, hogy a lemezcsomagok hozzárendelése az adatfeldolgozó rendszerekhez, az állományokhoz mennyire gazdaságos, mennyire intenziv.
A megoldás keresése egy mágneslemez nyilvántartó és kezelő komplex rendszer kifejlesztéséhez vezetett.

A rendszer szolgáltatásai az alábbiak:
VTOC listát készit a lemezcsomagokon lévő állományokról sávnövekvő sorrendben. Szekvenciális állományokra vonatkozóan fölszabaditja a védett, de adatokkal el nem foglalt területet és a védettséget a ténylegesen felhasznált területre vonatkozóan jegyzi be a VTOC-ba. Olyan állományoknál, ahol a területet utólag bővitették és a bővitések az eredeti területtel fizikailag egybefüggőek, a VTOC-beli védett terület bejegyzéseit egy bejegyzéssé vonja össze.

Kinyomtatja a területfelszabaditás és összevonás utáni állapotot, külön a szabad területek kimutatásával, és az eredeti és a módositott - pontosabban tényleges - lemezkihasználtság abszolút és százalékos mutatói.
Az egymás után következő fölszabaditott sávokról készül egy-egy "format-5" bejegyzés a VTOC-ba, amelyek a továbbiakban az automatikus területbiztositási funkcióhoz fognak információt szolgáltatni.
Minden, a fentiek szerint "kezelt" lemezcsomagról kerül egy bejegyzés egy lemeznyilvántartó törzsállományba, ez a lemezekkel kapcsolatos költségek elszámolását teszi lehetővé, és lemezgazdálkodási információt ad.

A lemez-kezelő rendszert állandó időközönként futtatják az összes adatlemezre vonatkozóan.

A fenti "karbantartó" funkcióra épül az automatikus lemezterület-biztositási rendszer /space management/, amelyet a DOS Job Control programjába épitettünk be.
Az automatikus területbiztositás úgy müködik, hogy a hagyományos EXTENT kártya helyett az alábbi formát is megengedjük:

// EXTENT SYSxxx, vol.ser.numb., T = nnnn

A T = nnnn operandusban azt kell megadni, hogy hány sáv szükséges a tervezett állomány felirásához. A módositott Job Control a format-5 cimkék segitségével tényleges területhatárokat biztosit az állomány részére. Ha a megadott lemezcsomagon a kért /nnnn/ sávszámnál csak kevesebb üres sáv áll rendelkezésre, akkor a SYSLOG logikai egységen /a konzolon/ a következő hibaüzenet jelenik meg:

"AUTOMATIKUS TERÜLETKÉRÉS NEM ELÉGITHETŐ KI X'onn',
vol.ser.numb."

Számitóközpontunk üzemvitelében kiemelt szerepe van elszámolási szempontból a job-név rendszernek és a ciklikus feldolgozások következtében a file-azonositó neveknek - a file identificator nevek - rendszerének.
A job-nevek hónapról-hónapra változóan fejezik ki az aktuális elszámolási információt, a file-ok azonositására vonatkozó jelsorozatokba szintén bekerül az aktuális hónap decimális sorszáma 01-től 12-ig .
Ez a rendszer nagyon jó áttekinthetőséget biztosit, viszont sok manuális munkával jár; a job-nevek és a file-nevek havi újralyukasztása miatt.

A manuális munkát úgy küszöböltük ki, hogy fölvittük az adatfeldolgozó rendszerek job-control kártyáit egy törzsállományból állitjuk elő az aktuális hónapnak megfelelő job-control-kártya sorozatot egy paraméterkártya vezérletével. Az egyetlen paraméterkártyában csak a módositás jellegét kell megadni, és a törzsállományt kezelő program ennek alapján akár többszáz // JOB, // DLBL és // TLBL kártya aktuális értékét állitja elő, és a "leszabott" job-kártya sorozatot /jc.stream/ egy mágneses adathordozóra helyezi.
A SYSIN logikai egységet a fenti fizikai egységhez rendelve a feladat futtatható.

ADATBÁZISKEZELÉS Rlo COBOL-BÓL*

Bánkfalvi J.-Bánkfalvi Zs.-Békéssy P.-Buzder J.-Horváth J.-
,Simonfai L.
Számitógépalkalmazási Kutató Intézet

Bevezetés

Az utóbbi évek során a SzÁMKI-ban kifejlesztésre került egy kisgépes,interaktiv adatbáziskezelő [1] és egy viszonylag jól kiépitett COBOL [2][3] rendszer. Tekintettel arra, hogy:
- az Rlo felhasználások jelentős része adatfeldolgozás orientált, továbbá
- a kisgépes adatbáziskezelő rendszerek jelentősége egyre nő,

kivánatosnak látszott a megszerzett tapasztalatok újraértékelése és az eddig független fejlesztési irányok egyesitése. Ezzel egyidejüleg Magyarországon is hozzáférhetővé vált egy Rlo alapú, interaktiv adatbáziskezelő és tranzakciófeldolgozó rendszer. Ennek súlyos hiányossága volt az, hogy nem rendelkezett procedurális interface-szel. A MADAM-X project azt a célt tüzte ki, hogy e rendszer /alaprendszer/ szolgáltatásait hozzáférhetővé tegye COBOL nyelvből. A COBOL adatbáziskezelő funkciókkal való bővitése során a lehetőségekhez képest maximálisan épitettünk a CODASYL [4] [5] ajánlásokban lefektetett elvekre és módszerekre.
Az alaprendszer olyan szinten valósitja meg az adatbáziskezelést, amely egybeesik korábbi elképzeléseinkkel arra nézve, hogy mi az az ésszerü határ, ameddig a CODASYL ajánlás megvalósitásában kisgépes rendszer esetén el kell és szabad mennünk. Ilyen előzmények után láttunk hozzá a MADAM-X rendszer megvalósitásához, mely egyrészt megtartja az alaprendszer szolgáltatásait, másrészt pedig CODASYL szellemü hozzáférést biztosit az adatbázishoz COBOL-ból /későbbiekben más nyelvekből is/ a COBOL szolgáltatásainak változatlanul hagyása mellett.

* A munka a VIDEOTON megrendelésére készült.

A rendszer megvalósitásának általános elvei

Az alaprendszer alapvetően interaktiv üzemü kisgépes adatbáziskezelő célrendszer. Az adatok közötti kapcsolatok rendszere - a séma - külön nyelven irandó le. A leforditott séma és a kezeléséhez szükséges rutinkészlet egy célszerüen módositott MMT monitor részeként van jelen a tárban. A COBOL futtatórendszer egy indexszekvenciális file kezelőt tartalmazó RTDM monitort használ és a mágnesszalagos file-ok kezelését a felhasználói programhoz hozzászerkesztett FMST segitségével oldja meg. Az emlitett software komponensek méreteinek ismeretében nyilvánvaló, hogy a COBOL rendszer és az adatbázis kapcsolatának megvalósitásához a két terjedelmes rendszer helyfoglalásának jelentős mértékü csökkentésére van szükség. Ezt megoldandó a következő taskositásokat kellett elvégezni:

- MMT monitor
- SGBD adatbáziskezelést végző monitor szekciók
- CODASYL interface-t megvalósitó program
- FMST mágnesszalagos file kezelő.

Az adatbáziskezelő funkciókat a COBOL nyelvbe CALL-ok formájában iktattuk be, mely megoldás lehetővé tette számunkra e forditóprogram módositásának mellőzését.

Az alaprendszer áttekintése

Az alaprendszer adatbázisleiró nyelve a bázisban tárolt rekordok szerkezetét és egymással való kapcsolatát az alábbiak szerint rögziti. Az azonos szerkezetü rekordok egyedi névvel ellátott maximált méretü file-okban nyernek elhelyezést. A rendszer a következő elemi adattipusokat ismeri, illetve különbözteti meg: kulcs, lista, szöveg, hivatkozások, dátum, óra, szó, BCD, BCDE, numerikus érték. Ezek közül az első négy pointerrel tárolt érték, amin azt értjük, hogy az ismétlődő szövegek helyfoglalása csökkentése céljából a rendszer ezeket táblákban és szótárakban fogalalja össze és az adatbázis rekordokban csak pointereiket szerepelteti. A táblákban a keresés hash-algoritmus szerint a szótárakban lineárisan történik. A kulcs tipusú érték alkalmas arra, hogy segitségével kulcs szerint azonositsunk valamely rekordot. A kulcs értékeknek természetesen egyedieknek kell lenniük, s igy mód van arra, hogy az őket gyüjtő tábla minden eleme tartalmazza a kulcsot magábafoglaló rekord sorszámát.

Az elemi adattipusok egy része mellett ismétlési tényező is állhat, melynek hatására a rendszer ennek megfelelő számú azonos tipusú érték számára tart fenn helyet az egyes rekordokban.

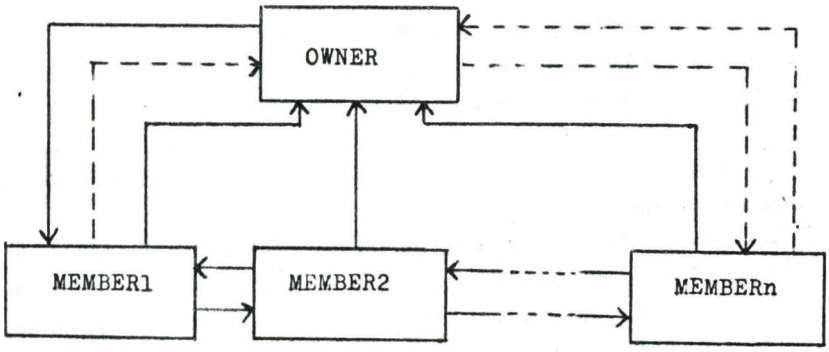
A különböző rekodtipusokat megtestesitő file-ok apa-fiu, illetve fiu-apa kapcsolatban állhatnak egymással. Nyilván minden ilyen kapcsolat egy-egy CODASYL értelemben vett halmazt állit elő. Az alaprendszer a halmazok egyedi névvel való azonositásáról nem gondoskodik. A file-ok közötti kapcsolatok megvalósitásában fontos szerepet játszik a hivatkozás tipusú elemi adat, amelynek neve mindig egy apa file nevével kell hogy egybeessen, aktuális értéke pedig automatikusan a tényleges apa kulcs tipusú elemi adatának értéke. A hivatkozás tipusú értékek tárolásával párhuzamosan megtörténik az általauk kijelölt kapcsolat létesitése is.

A CODASYL elemek megvalósulása

A rész-sémák alkalmazását elvetettük de - amint ez a továbbiakban kiderül - a MADAM-X rendszer adatleirásának a COBOL programok DATA DIVISION-jába való maximális beiktatása módot nyújt egy sor rész-séma feladat automatikus megoldására.

A MADAM-X rendszer - a CODASYL ajánlás kisgépes megvalósitásának hagyományait követve - az AREA-t a FILE-lal helyettesiti. Tekintettel arra, hogy a MADAM-X rendszer az adatbázisban tárolt rekordok helyét nem képes megváltoztatni, s ilyen irányú fejlesztését nem is tervezzük, adott FILE esetén a rekord sorszám veszi át a DATABASE-KEY szerepét. A CODASYL ajánlás legfontosabb vivmánya a SET fogalom bevezetése, mely lehetővé teszi, hogy az adatbázisba a felhasználóktól független módon kerüljenek beépitésre a tárolt adatok között fennálló kapcsolatok.
Az alaprendszer apa-fiu kapcsolata és annak implementálása néhány lényegtelen részlettől eltekintve egybeesik a CODASYL által javasolt CHAIN tipusú halmazokkal. Az a körülmény, hogy az alaprendszer esetében az apa rekordból csak az első fiu érhető el közvetlenül, szükségképp asszimetrikussá teszi a megfelelő adatkezelő utasitásokat is. /1. ábra/

A halmaz fogalmából természetes módon adódik a halmaz által előirt kapcsolat tényleges létesitésének és az ebből folyó rendezés felhasználásának kérdése. A CODASYL ajánlás a halmazok e jellemzőinek meghatározását elsődlegesen a séma /részséma - tehát az adatleiró nyelv - hatáskörébe utalja. Minthogy az alaprendszer erre közvetlenül nem ad módot, a MADAM-X rendszernek ugyanezt a feladatot részben automatikusan, részben az adatkezelő nyelv keretein belül kell megoldania.



1. ábra. A halmazok MADAM-Xbeli implementálásának összevetése a CODASYL javaslatával, ahol a CODASYL mellőzött kapcsolatait szaggatott vonal jelzi.

A MADAM-X rendszerben az egyes rekordok hivatkozás tipusú elemi adatainak tárolásával egyidőben történik a MEMBER rekordoknak a már létező OWNER-hez való hozzákapcsolása.
A deklarált kapcsolatok tényleges megvalósitása egyben az általuk egy-egy apához kapcsolt fiuk halmazainak rendezését is maga után vonja. E rendezés vezérlése céljából a MADAM-X rendszer adatkezelő nyelvében külön utasitás rendeli el, hogy egy MEMBER OWNERjéhez kapcsolásakor a lánc elejére, vagy adott MEMBER-t követően kerüljön beiktatásra. Abban az esetben, ha a kapcsolat megvalósitása hivatkozás tipusú elemi adat hatására automatikusan megy végbe, az adott rekord mindig a szóbanforgó lánc első eleme lesz.

Az adatbázis rekordok leirása a COBOL DATA DIVISION-jában

Az adatbázis és a COBOL programok közötti információ-átadás a DATA DIVISION-on keresztül történik. Noha magát az információátadást közvetlenül az adatkezelő utasitások hajtják végre, ezek tevékenységét döntő módon meghatározza az adatbázis rekordoknak a DATA DIVISION-ba beiktatott felhasználói leképezése. A leképezés általános szabályai a következők:

- a névhasználat független az adatbázisbeli nevektől;
- minden rekordtipusnak egy COBOL csoportmező felel meg, melynek legalább annyi közvetlen részmezője van ahány elemi adatot a szóbanforgó rekordtipusból figyelembe kivánunk venni;
- au ismétlési tényezővel ellátott elemi adatoknak ugyanannyiszoros OCCURS deklaráció kell, hogy megfeleljen;
- az információátadásnál érintett adatbázisbeli elemi adatok listáját az irást, olvasást és update-elést előiró adatkezelő utasitások határozzák meg;
- az információátadást a közvetlen COBOL részmezők és az elemi adatok listájának elemenkénti párbaállitása vezérli. melyben a rövidebb lista határozza meg az átvitel terjedelmét és a párt képező listaelemek esetében a COBOL-mező tipusúnak - pontos, itt nem részletezett szabályok szerint - igazodnia kell a neki megfelelő elemi adat adatbázisbeli tipusához.

A fenti leképezés igen rugalmas, és sok tekintetben a rész-séma által nyújtottakhoz hasonló lehetőséget biztosit a felhasználó számára. A megoldás gyengéje, hogy - mivel a fordító nem fér hozzá a sémához - a felhasználó elképzelései helyességének ellenőrzésére nem nyújt lehetőséget.

A MADAM-X rendszer adatkezelő utasitásai

A felhasználó a MADAM-X rendszer adatkezelő szolgáltatásait a COBOL program PROCEDURE DIVISION-jába beiktatott CALL-ok felhasználásával veheti igénybe. A CALL utasitások a MADAM-X rendszert feltételező COBOL programok mindegyikéhez hozzászerkesztendő DBCSC modulra vonatkoznak, mely számára a hivás első paramétere a kivánt tevékenységet azonositja, mig a további paraméterek a kiválasztott tevékenység specifikálásához szükséges információt közvetitik.

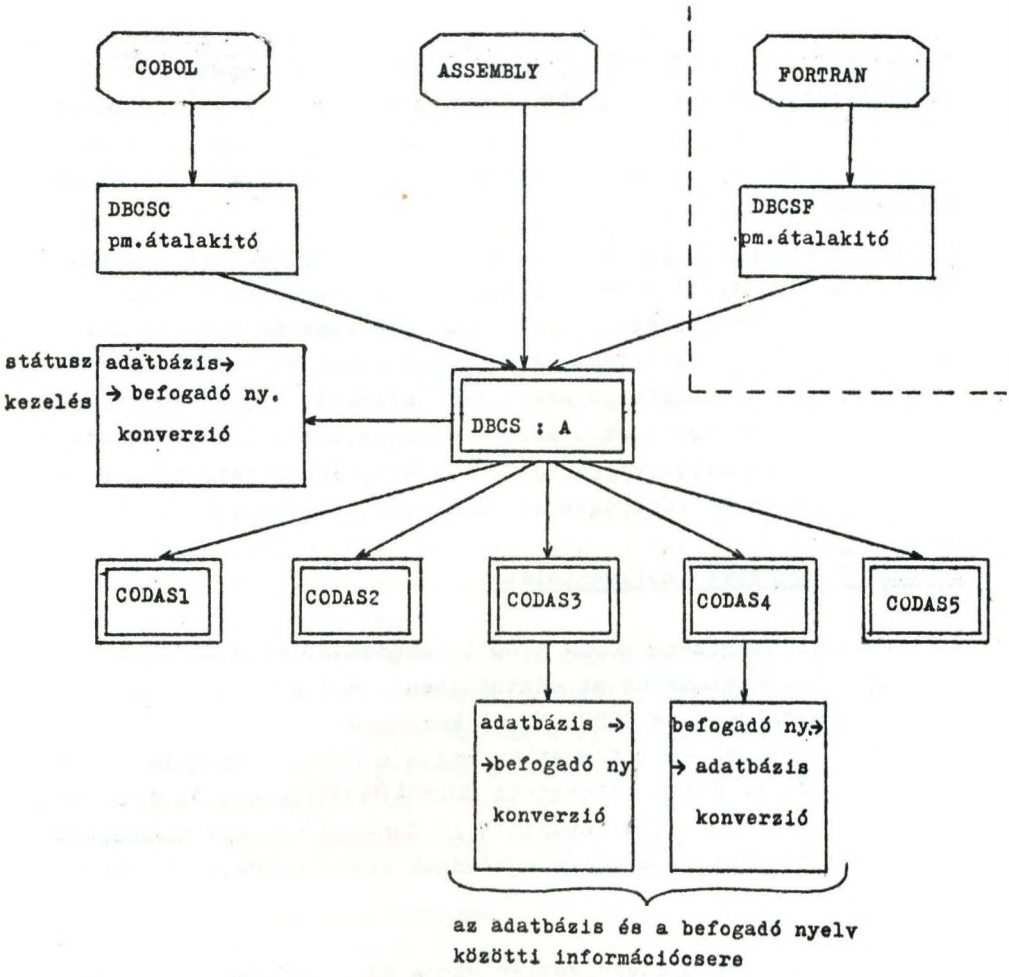
Az egyes funkciók megalkotásában és elnevezésében a lehetőségeket figyelembe véve a CODASYL ajánlást követtük. Az egyetlen elviekben is jelentős eltérés abban áll, hogy valahányszor egy utasitásban rekord kiválasztására van szükség, minden logikailag lehetséges rekordazonositási módszer /kurrensség, kulcs, sorszám/ alkalmazását megengedjük.

A MADAM-X rendszer adatkezelő utasitásainak végrehajtása során fellépő rendellenes, illetve különleges körülményekről a felhasználó egy, az adatbázist megnyitó OPEN utasitásban specifikált mezőben kap felvilágositást. A négyjegyü státusz információ első két jegye a szóbanforgó adatkezelő utasitás tipusát adja meg, utolsó két jegye pedig az utasitás végrehajtásáról tájékoztat. Az utasitás szabályos végrehajtását lehetetlenné tevő hiba felfedezése esetén az adatbázis tartalma változatlan marad.

A MADAM-X rendszer implementálása

A rendszer kidolgozása során arra törekedtünk, hogy - lehetőség szerint - különválasszuk az adatbázissal való kapcsolat megvalósitásának általános és COBOL-függő komponenseit.
A MADAM-X rendszer nyelvfüggetlen magja a DBCS:A assembly nyelvü szekcióból és az általa élesztett CODASi (16 i 45) taskokból áll, mely utóbbiak a rendszer által nyújtott 17 funkció egy-egy alkalmasan megválasztott csoportja végrehajtásának vezérlésére szolgálnak. /2. ábra/

A MADAM-X rendszer nyelvfüggetlen magja és a befogadó nyelv közötti kapcsolat alapját képező mezőleirót magasszintü nyelv esetén a megfelelő nyelvfüggő rutin automatikusan, assembly nyelv esetén pedig a programozó manuálisan hozza létre. A mezőleiró alakja a COBOL tábláinak felépitését követi. Ez a döntés lényegesen megkönnyitette a jelen rendszer kidolgozását anélkül azonban, hogy a nyelvfüggetlen mag általános használhatóságát veszélyeztetné, hiszen a mezőleiró előállitásához szükséges adatokról joggal feltételezhető, hogy bármely magasszintü nyelv esetében elérhetők egy assembly nyelvü futtatási rendszerbeli modul számára.



2. ábra. A MADAM-X nyelvfüggő és nyelvfüggetlen komponenseinek kapcsolata.

Az adatbázisnak és a befogadó nyelvnek a mezőleirón keresztül realizálandó kapcsolatát az ún. konverziós rutinok valósitják meg.
A fentiek alapján nyilvánvaló, hogy a rendszer egyedüli COBOL-függő tevékenységei:

- a COBOL-ba ágyazott CALL-ok paraméterezésének átalakitása a nyelvfüggetlen mag által megkivánt alakra
- a COBOL-ban előforduló értéktipusok és az adatbázisbeliek közötti konverzió megvalósitása.

A COBOL számára létrehozott eszközök tehát jó alapot biztositanak arra, hogy az adatbázis más nyelvekből is - elsősorban FORTRAN-ból - hozzáférhető legyen.

IRODALOMJEGYZÉK:

1. Simonfai, L;Horváth,J.: MADAM - an experimental data base management system on a minicomputer.
   Preprints of 2nd Hungarian Computer Science Conference Budapest, 27 June-2July, 1977. pp 769-796.
2. Bakos T.: Rlo COBOL felhasználói kézikönyv SzÁMKI, 1974.
3. Bánkfalvi,J; Bánkfalvi,Zs.: Determinisztikus top-down elemző implementálása az UTRA rendszerben.
   Információ Elektronika 1975/4 pp 284-289.
4. CODASYL Data Description Language
   Journal of Development June 1973.
5. CODASYL COBOL
   Journal Development 1976.

NYELVRESZABOTT INTERAKTIVITÁS AZ ANSWER PROGRAMFEJLESZTŐ RENDSZERBEN

Bárány Sándor - Bolgár Gábor
Számitógépalkalmazási Kutató Intézet

1. Bevezetés

Az ANSWER rendszer általános ismertetése [1] során láttuk a nyelvi szerkesztő és az interaktiv próbapad helyét, feladatát a nyelvi rendszerben. Itt most röviden összefoglaljuk az ANSWER rendszer azon alapvonásait, amelyek a nyelvi szerkesztővel és az interaktiv próbapaddal kapcsolatosak. A részletes tárgyalásban mutatunk majd rá arra, hogyan valósulnak meg ezek az alapelvek.

2. A nyelvi szerkesztő

2.1 A nyelvi szerkesztő helye a nyelvi rendszerben

Az ANSWER rendszer dipslay-eket használ a felhasználókkal való interaktiv kapcsolathoz. Ez teszi lehetővé, hogy a program megirásának kezdetétől ne kelljen papirt /kódlapot, kártyát, listákat/ használni. Ehelyett a rendszerben a forrásszövegek létrehozására, megőrzésére, visszakeresésére, javitására egyszerü és gyors mód van.

Az ANSWER rendszer, interaktivitása révén, a felhasználó munkájához kellemes, a programozói munka hatékonyságát megsokszorozó környezetet próbál teremteni. A nyelvi szerkesztő hozzájárulása ehhez:

- használata egyszerü, nem vonja el a programozó figyelmét a munkától bonyolult parancsokkal

- az elkövetett hibákat nagyon hamar, többnyire a leirás pillanatában jelzi: ez a javitásnál nagy könnyebbség

- ahol lehet, a felesleges irásmunkától megszabadit:
  - ha egy szimbólum egy adott helyen előre várható, akkor azt a programozó helyett kiirja
  - egy, a kezdetéből már egyértelmü szimbólumot /röviditést/ teljes alakjával helyettesiti.

2.2 A nyelvi szerkesztő fogalma és viszonya a hagyományos szövegszerkesztőkhöz

A hagyományos programszöveg-javitó eszköz a karakteres szövegszerkesztő. Ennek hátrányai:

- nem ad segitséget a forrásprogram megirásánál ahhoz, hogy az irásmunka végén már szintaktikusan helyes programunk legyen: egy-egy javitással a már helyes programot el lehet rontani.

- a javitások megfogalmazásánál, a javitások helyének megkeresésekor nem támaszkodhatunk a forrásprogram szerkezetének, szintaxisának ismeretére, hanem csupán karakter-sorozatok megadása lehetséges /ennek veszélyességére a klasszikus példa: cseréld ki mindenhol az AIMA betüket KÖRTE betükre - az eredménye: HATALMA azonositóból HATKÖRTE lesz - ez a "jutalma", hogy nem a nyelv jól ismert fogalmaiban, hanem semmitmondó karakterekben gondolkodtunk/.

- a javitás mindig az egész karakterhalmaz ide-oda másolásával, majd az egész forrásprogram újraelemzésével jár.

A nyelvi szerkesztő olyan szövegszerkesztő, amely egyetlen nyelvhez kötődik /más nyelvü szöveget lehetetlen vele kezelni!/: ettől nyelvi. Ennek az egy nyelvnek ismeri /felimeri, ellenőrzi/ a szintaxisát is. Programot javitani és programot irni egyaránt lehet vele, és müködése során állandóan vigyáz arra, hogy csak szintaktikusan helyes programrész keletkezhessen. A szintaxis ismerete azt is lehetővé teszi, hogy egy hiba helyének megkeresése, ill. meghatározása során a nyelv fogalmaival dolgozhassunk karaktersorozatok helyett. /Igy pl. azonositók cseréje esetén a fent emlitett AIMA-HATALMA összekeveredés elkerülhető./

A nyelvi szerkesztő működését, használatát a továbbiakban egy egyszerü példán mutatjuk be. A példa az alábbi, képzelt szintaxisú nyelv egy kis programja lesz:

<program> ::= {<utasitás>.}*
<utasitás> ::= <egyszerü utasitás> | stb...
<egyszerü utasitás> ::= <azonositó> + {<paraméter>}*
<paraméter> ::= <azonositó>

A konkrét program pedig:

ALFA. BÉTA+X+Y. GAMMA+X. DELTA.

kezet egy részét kitüntetjük, ez a kijelölt egység.

A szerkezet szemléltetése a nyelv szintaxisán alapszik: egy nagyobb szintaktikai egység, amely kisebb egységekből épül fel, ezeknek a felépitő egységeknek a röviditéséből áll össze, a felépitő egységek teljes szövege nem látható. A kisebb egységek közül a kijelölt egység aláhúzással van kiemelve.

Kezdetben a kijelölt egység az egész program. Ahhoz, hogy a programozó ennek valamely részletét kiválassza /és ezzel láthatóvá tegye/, a helykijelölő program segitségével a kivánt részletet meg kell keresnie. Ezt a helykijelölő utasitásokkal érheti el, amelyek a kijelölt egységet változtatják meg, állitják át. Minden helykijelölő utasitás után a forrásprogram-visszalistázó újra meghivódik, igy a "látvány" /a forráslista röviditett képe/ is módosul. Ezért beszélünk a helykijelöléssel történő "nagyitásról", stb.

Lássunk végre konkrét példákat:
Kis programunk kezdetben igy fest a képernyőn:

<program>

Részletezni a ↓ helykijelölő utasitással tudjuk, ez egy összetett szintaktikai egységet elemeire bont és az első elemet kiválasztja:

<egyszerü utasitás>.
<utasitás-lista>

2.3 Forrásprogram-kezelés az ANSWER nyelvi rendszerében

A nyelvi rendszer számos programja müködése során forrásprogram-kezelést is végez /szerkesztő, magas szintü linkage-editor, próbapad, hibakezelő, stb./. Hagyományos nyelvi rendszerekben az ezeknek megfelelő programok /kérésre/ forráslistákat készitenek, amelyekben általában sorszámozással segitik az eligazodást. A listában a tájékozódás a sorszám, a hibára vonatkozó üzenetek, stb. alapján történik. A javitandó részletet is a sorszámmal és a soron belüli karakterpozició vagy a kezdőkarakterek, stb. megadásával szokás kijelölni. Ezek a módszerek sokszor fárasztóak, lassúak és sok hibát okozhatnak.

Az ANSWER rendszerben a papiron lévő listák helyett a szintaxison alapuló szerkezeti felbontást és a display lehetőségeit használjuk fel a programban való eligazodásra. A képernyő természetesen sokkal kisebb annál, hogy akár egy jellemző programrész is ráférne. Ezt az ellentmondást a forrásprogram-visszalistázó és a helykijelölő programok oldják fel.

2.4 A forrásprogram-visszalistázó és a helykijelölő

Teljes program a képernyőn egyszerre sosem létezik, hanem csak a programnak egy szerkezeti kivonata. Igy viszont a részletekhez nem férhetnénk hozzá sohasem, ha a szerkezet egy-egy részletét nem lehetne "kinagyitani". Ezért a szer-
Ha most nem az első, hanem a második utasitást akarjuk látni, akkor a → utasitással léphetünk rá, amely az ugyanazon a "szinten" lévő szintaktikai egységek közül a következőt teszi kijelölt egységgé:

<egyszerü utasitás> .
<egyszerü utasitás> .
<utasitás-lista>

Egy újabb ↓ utasitás hatására az egyszerü utasitás belsejébe lépünk be. Az előzőek analógiájára a részletezésnek ilyennek kellene lennie:

<azonositó>
<paraméter-lista>

Az ilyen mélységü szinteken azonban a rövidités már túlzott részletezéssé fajulna; sőt, sokszor hosszabb lenne, mint az eredeti forrásszöveg. Ezért ilyenkor a "rövidités" elhagyjuk. A forrásszöveg ilyen kis darabjai a képernyőn is jól elférnek és jól áttekinthetők. A kijelölt egység kiemelése természetesen nem változik. A példa tehát igy látszik:

<egyszerü utasitás>.

BÉTA

+X+Y. <utasitás-lista>

A ↓ és a → utasitásoknak természetesen az ellenpárjuk is használható. A ← utasitás "visszaforditja" a → utasitás hatását: a kijelölt egységet a megelőzőre állitja vissza. A ↑ utasitás visszatér a megelőző szintre, a kijelölt egységet tartalmazó szintaktikus egységre.

A fenti, "nyil"-utasitások használata első látásra hosszadalmasnak, nehézkesnek tünik. A programok szintaktikus szintjeinek a száma azonban nem nagy, és igy egyedi, egyszeri eljutás valahova nem túl lassú, viszont az állandó ellenőrizhetőség miatt igen biztonságos.

Ha egy helyre többször is el kell jutni a javitási folyamat során, vagy egy szintaktikai egységet valamilyen más szempont miatt kell megjelölni /lásd később/, akkor használhatjuk az ún. cimkéket. Egy cimkét egy kijelölt egységhez lehet hozzárendelni, "kitenni", és ezután bármikor a cimkére való hivatkozás ezt az egységet jelenti.
Példa: ha a fenti egyik példa, az

<egyszerü utasitás>.
<utasitás-lista>

esetén az első-cimke cimkét kitesszük, akkor a legutóbbi példánkból a @ első-cimke helykijelölő utasitás visszaállitja a cimke kitevésekor érvényes állapotot. Ugyanezt a ↑ és ← utasitásokkal is elérhettük volna.

2.5 A nyelvi szerkesztő utasitásai

A nyelvi szerkesztő mindig a kijelölt egységen végzi el az utasitások által megadott müveletet. Ha az utasitásnak csak egy operandusa van /pl. DELETE/, ez elegendő is. Ha további operandusra van szükség, akkor azt vagy egy cimkehivatkozással, vagy a karakteresen leirt új forrásprogram-részlettel lehet megadni.

A nyelvi szerkesztőnek igen kevés utasitása van, ezek működése is igen egyszerü.

Az INSERT utasitás a kijelölt egység mögé beszúrja az operandusaként megadott másik egységet.

A BEFORE a kijelölt egység elé szúr be.

A DELETE a kijelölt egységet törli.

A REPLACE a kijelölt egységet felülirja az operandusként megadott másik egységgel.

Az EXCHANGE a kijelölt egységet és a cimkehivatkozással megadott operandusát kicseréli.

A CORRECT utasitás a kijelölt egység karakterszintü javitását teszi lehetővé. Ez a "kiskapu", amikor a fenti utasitások egyikével sem lehet, vagy csak nehézen lehet a kivánt javitást elvégezni. Ilyenkor a javitás befejezésekor az egész egység újraelemződik és csak a szintaktikusan hibátlan programrész kerül vissza a programba.
A LOOK FOR utasitás ciklusban az operandusként megadott azonositó előfordulásaihoz állitja a kijelölt egységet. Igy lehet szisztematikus javitásokat könnyen elvégezni.

3. Az Interaktiv Próbapad /ITB = Interactive Test Bench/

3.1 Az Interaktiv Próbapad feladata

Az ANSWER operációs rendszer próbapadja arra szolgál, hogy készülő szoftver termékeinket - esetleg még hiányos állapotban is - működés közben interaktiv módon meg tudjuk figyelni. Az ITB tehát az ANSWER rendszerben készülő programok bejáratását és hangolását segitő interaktiv eszköz.

A programok a rendszer egyes elemei /nyelvi szerkesztő, magas szintü összeszerkesztő program/ által végzett átalakitások után kerülhetnek a próbapadra. A nyelvi szerkesztő biztositja, hogy a program moduljai környezetfüggetlen szintaktikus értelemben hibátlanok legyenek, a magas szintü összeszerkesztő program pedig a teljes program környezetfüggő szintaktikus, ill. bizonyos tipusú szemantikus hibáktól való mentességét garantálja. Természetesen ettől még nem biztos, hogy a program valóban azt csinálja, amit elvárnak tőle, és az sem, hogy működése gazdaságos. Ezért a próbapadon a programot bejáratjuk, közben az észrevett hibákat a rendszer hibakezelő programjának a listájára felvesszük, majd a hibakezelő programot használva a hibákat kijavitjuk. Amikor a próbapadon úgy látjuk, hogy a program azt csinálja, amit kell, akkor hozzáláthatunk a hangoláshoz. Mindezen tevékenységek megsegitésére az ITB három fontos lehetőséget biztosit:

i/ Hiányos programok futtatását is lehetővé teszi /interaktiv szimuláció segitségével/.

ii/ A programok működése szakaszokra bontható: a futás /majdnem/ tetszőleges helyeken megszakitható, majd továbbinditható. A továbbinditás előtt a program alkatrészei megvizsgálhatók. A program lépésenként is végrehajtható.

iii/ A programok működéséről adatokat lehet gyüjteni minőségvizsgálat céljára.

3.2 Hiányos programok végrehajtása a próbapadon

Az ITB-n olyan programok is futtathatók, amelyekből egy vagy több teljes modul hiányzik. Ezzel az ITB lehetővé teszi, hogy részlegesen elkészitett programokat /programrészeket/ is kipróbáljunk, vagy hogy több programozó által irt program esetén az egyes programozók saját részeiket önállóan bejárathassák, stb. Ehhez az ITB a programozót kéri fel a program hiányzó részeinek szimulálására.

3.3 Az ITB-n futó programok müködésének megfigyelése

Az ITB-n futó programok müködése bizonyos pontokban felfüggeszthető. A felfüggesztés logikailag egy töréspont a vezérlésben, fizikailag egy csapda a program belső alakjában. A töréspontokban a programozó újabb utasitásokat adhat az ITB-nek, amelyek a következő lehetőségeket nyújtják:

a/ A program végrehajtása abbahagyható.
b/ Ujabb töréspontok jelölhetők ki /újabb csapdák elhelyezésével/.
c/ Csapdákat lehet megszüntetni.
d/ A program alkatrészeit meg lehet vizsgálni.
e/ A csapdákról listát lehet kérni.
f/ A képernyő tartalma felvehető a hibalistára.
g/ A program végrehajtása folytatódhat.

Ezen parancsok legnagyobb részét /vagy legalábbis hozzájuk nagyon hasonló funkciókat/ sok más rendszerben is találhatunk. Itt csak néhány olyan vonásra térünk ki, ami az ANSWER operációs rendszernek sajátja.

- A csapdák kijelölése is speciális módon történik, nem sorszámmal határozzuk meg a csapda helyét, hanem ún. ITB-cimkéhez tehetjük le azt. Az ITB-cimkéket a helykijelölő program segitségével tehetjük le. Ha a helykijelölővel közöljük, hogy egy cimkét csapda helyének kijelölésére akarunk használni, akkor a helykijelölő program arra is ügyel, hogy a cimke olyan helyre kerüljön,ahova valóban értelmes dolog csapdát állitani. Az ilyen cimkék kitevéséről nem kell feltétlenül az ITB-n való futtatás megkezdése előtt

gondoskodni, mert bármely töréspontban a helykijelölő program az ITB-ből is aktivizálható.

- Minden töréspontban, tehát minden olyan pillanatban, amikor a vezérlés felfüggesztődik, a képernyőre az ITB a visszalistázó program segitségével kivetiti az ún. vezérlési képet, ami a forrásszövegnek azt a részét tartalmazza, ahol a vezérlés éppen tartozkodik, s a kijelölt egység a soron következő algoritmust mutatja. Ezáltal a futó programot a képernyőn nyomon tudjuk követni.

- Bármely töréspontban a vezérlési képről át lehet váltani az ún. kereső képre, amelyben a helykijelölő program eszközeivel manipulálhatjuk a forrásszöveget, s igy a programnak bármelyik részét fellapozhatjuk, vagy pl. csapda elhelyezése céljából cimkét tehetünk ki.

- Ha a vezérlési képről átváltunk a kereső képre, a kijelölt egység az az algoritmus lesz, amelyiknek a végrehajtása éppen következik. Ha a vezérlési képre visszakapcsolunk, akkor attól függetlenül, hogy a kereső kép mit tartalmazott, a vezérlési kép ismét a soron következő algoritmust mutatja kijelölt egységként.

- A képernyő tartalmát /a kijelölt egységet/ és egy hozzáfüzött tetszőleges magyarázó szöveget fel tudjuk venni a programhoz tartozó hibalistára, hogy ezzel a program futtatása utáni hibajavitást megkönnyitsük. /A hibák kijavitását a hibakezelő programmal végezhetjük el a hibalista alapján./

- Az ITB-n a programokat lépésenként is végre lehet hajtani. A "lépés" fogalma is szorosan a rendszer nyelvéhez kapcsolódik. A lépésenkénti futtatást egy kis programrészleten szemléltetjük. Tegyük fel, hogy a vezérlési kép a program futásának a következő állapotát mutatja:

ALFA:
BÉTA, GAMMA, DELTA.

Ez azt jelenti, hogy a vezérlés az ALFA nevü szabályban a BÉTA nevü algoritmus hivása előtt áll. Ha most kiadjuk a → /STEP/ parancsot, akkor a BÉTA algoritmus lefut, s a következő vezérlési képet látjuk:

ALFA:
BÉTA, GAMMA, DELTA.

Ezután, tegyük fel, hogy átváltottunk keresőképre, s ott a forrásszövegben lapozgatva megállapitottuk, hogy a GAMMA algoritmus futását érdemes volna megnézni. Ezért a ↓ ENTER/ parancsot adjuk ki, aminek eredményeképpen a vezérlés belép a GAMMA algoritmusba, s a következő vezérlési képet kapjuk:

GAMMA:
EPSZILON, FI.

Ekkor kiadjuk az ↑ /OFF/ parancsot, aminek hatására az a legszükebb algoritmus, amiben a vezérlés éppen tartozkodik, /esetünkben GAMMA/, lefut:

ALFA:
BÉTA, GAMMA, DELTA.

Láthatjuk tehát, hogy az ITB használatakor nem sorszámokban, hanem a rendszer nyelvének fogalmaiban gondolkodunk.

3.4 Adatok gyüjtése minőségvizsgálat céljára

A logikailag helyes programot hatékonyságvizsgálatnak vethetjük alá. A fejlesztőgépen olyan hatékonysági kérdéseket vizsgálhatunk, amelyek a célgéptől nem, vagy csak alig függenek. Ezekhez a vizsgálatokhoz az ITB /külön kérésre/ adatokat szolgáltat: statisztikát készit a program dinamikus viselkedéséről. Az ITB számlálja, hogy a vezérlés hányszor haladt át az egyes hivásokon, ugrásokon, csoportokon, szabálydefiníciókon, makródefiníciókon, cimkéken. Logikai értékkel rendelkező algoritmusok esetében a sikeres és a sikertelen kilépéseket is számlálja. Az ily módon nyert információk alapján megállapithatjuk, hogy melyek azok a szabályok, amelyeket érdemes volna átirni, esetleg alternativákat felcserélni, stb. Természetesen az ITB által szolgáltatott információk is a programozó számára jól érthetően, forrásnyelven olvashatók.

4. Hivatkozások

[1] Bedő Á.-Langer T.: A programozás technológiája az ANSWER operációs rendszerben. SZEGED, PROGRAMOZÁSI RENDSZEREK'78.

[2] ANSWER nyelvi rendszerének felhasználó szintü leirása SzÁMKI, SOFTTECH sorozat, D 3 kötet.

INTERAKTIV PROGRAMCSOMAG 3D TESTMODELLEK FELÉPITÉSÉRE ÉS AZOK KEZELÉSÉRE

Báthor Miklós
MTA Számitástechnikai és Autómatizálási
Kutató Intézete

I. Bevezetés

Az Intézetben,egy intelligens ipari robot kidolgozására irányuló munka kapcsán szükségessé vált egy olyan programcsomag elkészitése,amely egyszerü geometriával leirható testek számitógépes modelljének a kialakitására,majd a modell különféle manipulálására képes. Az interaktivitást a szöveges kapcsolaton túlmenően igen széles körü grafikai szolgáltatásokkal is segitjük,igy a modellek felépitése könnyen végrehajtható,a programcsomag kevéssé képzett felhasználó számára is jól áttekinthető,a kezelése egyszerüen elsajátitható.

A hardware-t szándékosan minél egyszerübbre,olcsóbbra választottuk. R-10 gépet használunk,egy 800 Kbyte kapacitású,MOM gyártmányú minidiszkkel kiegészitve. A programok a szintén az Intézetünkben kifejlesztett IDOS /Interactive Disc Operating System/ monitor felügyelete alatt futnak, kihasználva annak széleskörü file-kezelő és -könyvtárazó szolgáltatásait. Az alfanumerikus kapcsolatot Videoton gyártmányú display-en bonyolítjuk le,míg grafikus megjelenitőként egy TEKTRONIX-613 típusú,tárolócsöves display szolgál. Az utóbbi 256x256 pont felbontású képernyővel rendelkezik,ez a legtöbb esetben elegendő a kezelni kivánt test-modellek megfigyelésére.Grafikus inputként mindössze egy - külső utasitásokkal pozicionálható - cursor van,amellyel a képernyő tetszőleges pontjának azonositása érhető el.

II. A testek matematikai modellje,adatstruktura

Viszonylag egyszerü geometriával leirható testek kezelését tüztük ki célul,ilyenek például az iparban szokásos megmunkálásokkal előállitott alkatrészek.A testmodelleket hierarchikusan épitjük fel, egy előre rögzitett /azonban szükség esetén bővithető/ alapelem-választék felhasználásával.A test élei kétfélék lehetnek: egyenesek és max. 180° középponti szögü körivek.Az élek lehetnek valóságosak, vagy ugynevezett fiktivek; ez utóbbiakra főként a testet határoló felületek definiálásánál lesz majd szükség.Egyenes élet a két végpontja már meghatároz, köriv leirásához az iv egy tetszőleges közbülső pontját is meg kell adni.

A testet határoló felületek esetében négyféle alapfelületet engedünk meg: sikot,hengert,egyköpenyü forgáskupot és gömböt,illetve ezek tetszőleges kisebb darabkáit.Minden felület csak egyszeresen összefüggő lehet,többszörösen összefüggő /pl. "lyukas"/ felületeket a már emlitett fiktiv élek segitségével lehet egyszeresen összefüggő darabokra felbontani.Ugyancsak fiktiv élekre van szükség,ha két különböző tipusú vagy paraméterü felület úgy találkozik,hogy ott a felületnormális-vektor folytonosan megy át /"sima" átmenet,lekerekités,stb./.Az emlitett másodrendü felületeket gyakori előfordulásuk miatt definiáltuk alapelemekként; a szokásos - sikokkal való - approximálásuk az adatstruktura és a feldolgozási idő szükségtelen megnövekedését eredményezné.Az alapelem-választék az esetek legnagyobb részében elegendőnek bizonyult,azonban az algoritmusok egyszerüen kiterjeszthetők tetszőleges másodrendü felület kezelésére; az alapválaszték könnyen bővithető.

A test-modellt alkotó adatokat az IDOS operációs rendszer által könyvtárazott file-ban helyezzük el, ennek az azonositója egy tizbetüs név.A file-on be-

lül több adattáblát épitünk fel.A test csucspontjait és az ugynevezett "köriv közbülső pontokat" a testhez rögzitett,egyébként tetszőlegesen választott koordinátarendszerben, 3-3 koordinátával adjuk meg.A csucspontoktól ezenkivül még pointerek hivatkoznak az onnan kiinduló élekre is.

A következő táblában.az élek leirása található.Egy élet a két végpontjára illetve a köriv közbülső pontjára mutató referenciák definiálnak.Pointerek mutatnak ezen kivül az élhez csatlakozó két felületre is /helyesen modellezett test esetében minden élnek pontosan két felülethez kell tartoznia/.

Felület leirásához szükség van egyrészt a felület egyenletére,másrészt a felület tényleges lehatárolását adó élek megnevezésére.A felület egyszeresen összefüggő lévén,határát egy összefüggő él-lánc alkotja; pointer-sorozat segitségével adjuk meg a határoló éleket,tetszés szerinti körüljárási irányban. A felületegyenleteket a következő táblázatban tároljuk,erre ugyancsak pointer mutat.

A felületegyenleteknél a minimális adatmennyiségre való törekvés a következő egyenlet-ábrázolást eredményezte: sikot az

$$Ax + By + Cz + D = 0$$

egyenlet definiál; az /A,B,C/ vektor merőleges a síkra. Sík esetében az A,B,C,D számokat tároljuk. Gömb leirásához tárolandó a gömb középpontjának három koordinátája és a gömb sugara. Hengernél a tengely egy tetszőleges pontját, a tengely irányvektorát és a henger sugarát tároljuk. Kúp esetében ugyanezek az adatok szükségesek,annyi finomitással, hogy a kúp csúcsát a tengelyvektor végpontja jelöli ki,míg a kúp "sugarát" a tengelyvektor talppontjára illeszkedő,rá merőleges sik által a kúpból kimetszett kör sugara adja. A felületeket irányitással is ellátjuk: kész test-modellnél a tér pontjairól egyértel-

müen eldönthető,hogy a testen belül vagy kívül helyezkednek-e el,így a test minden felületének megnevezhető a külső és a belső oldala. Ez az információ a modell további felhasználásánál lényeges lehet, ezért sík esetében az A,B,C,D számokat úgy adjuk meg,hogy az /A,B,C/ vektor a testből "kifelé" mutasson,a másodrendü felületeknél pedig jelezzük, hogy a felület kívülről nézve konvex vagy konkáv.

Az adatstruktúra későbbi,a speciális felhasználástól függő,előre esetleg nem látott bővítését azáltal tettük lehetővé,hogy minden felülethez és a teljes testhez is megengedtük úgynevezett pót-információk hozzárendelését. Az utolsó adatmező ezeket a lehetőségeiben adott,később definiálandó pót-információkat tartalmazza.

III. Programszerkezet,a program használata

A program alapállapotban egy test modelljén dolgozik,amelyet az IDOS könyvtárából lehet a memóriába tölteni,majd a munka tetszés szerinti fázisában újból könyvtári nyilvántartásba venni,így a munka bármikor megszakitható és később ismét folytatható. A modell a felépítési folyamat közben a TEKTRONIX display képernyőjén vizuálisan megfigyelhető: a test csomópontjait és éleit drótváz-szerü ábrán jelenítjük meg. Lehetőség van a perspektív képet előállító virtuális "kamera" helyzetének,távolságának,látószögének a változtatására.

A kamera által alkotott kétdimenziós drótváz-ábrán a testet felépítő alapelemeket a grafikus display cursor-jének a segítségével azonosíthatjuk. Az alfanumerikus tasztatúra cursor-mozgató billentyüivel mozgathatjuk az ábrán a cursor-t,majd csúcspont ill. él választás esetén a cursor az ábrán a hozzá legközelebbi megfelelő elemre ugrik /a távolságot a test vetülete és a cursor között a képernyő síkján értelmezzük/. Felületet valamelyik határoló éle által azo-

nosíthatunk,illetve ha a kijelölt élhez már két felület tartozik,akkor még egy további határoló él is kijelölendő. Természetesen lehetőség van csúcs,él vagy felület közvetlen azonosítására is,a lista szerinti aktuális sorszámának a megadásával.

A képi megjelenítéstől eltekintve,a program szervezése fa-struktúrájú;az operátor egymást követő kiválasztások során határozhatja meg az elvégzendő feladatot. A fa elágazási pontjaiban a kiválasztás lehetőségét úgynevezett "menü"-elven biztosítottuk: a szóbajövő folytatási irányok az alfanumerikus display-en egymás alá kiíródnak és az operátor a cursor segítségével választhatja ki a megfelelőt. A program bejelentkezése és a file-választás után megjelenő alap-menü választást enged,hogy csúcspontokkal,élekkel vagy felületekkel kívánunk-e foglalkozni,illetve transzformálni akarjuk-e a testet,vagy a pót-információban elhelyezett szub-struktúrával fogunk dolgozni. Az első három esetben generálni,módosítani vagy törölni lehet az alap-elemeket. Az egyes operációk során a program az elvégezni kívánt tevékenységet részletes vizsgálatnak veti alá és hibás testmodellt lehetőleg nem enged létrehozni /például egy él két végpontja nem lehet azonos; egy felület határoló éleiként csak egymással lánc-szerüen kapcsolódó élek jelölhetők ki,stb./ Felületgenerálás után henger és kúp esetében még a tengely irányát is meg kell adni,majd a program legkisebb négyzetes módszerrel illeszti a megfelelő felületegyenletet. Könyvtárazás előtt a modellről különböző információk kérhetők, mint például: van-e olyan él,amelyikhez nem két felület tartozik; milyen az egyes felületek irányítása; teljesül-e a testre az Euler-egyenlet,stb.

Az eddigieken túlmenően lehetőség van két test-modell adatstruktúrájának az egyesítésére,valamint tetszőleges lineáris transzformáció elvégzésére. Az utóbbi esetben a transzformációs mátrix megadásán kívül külön kérhető a modell térbeli eltolása vagy elforgatása tetszőleges tengely körül.

Mint már említettük,a program a kialakítás alatt levő modellt alap-állapotban drótváz-szerü ábrán jeleníti meg. Ez az ábrázolás gyorsan elkészíthető és egyidejüleg tartalmazza az összes csúcspont és él vetületét,azonban egyrészt semmi felvilágosítást nem nyújt a már generált felületek elhelyezkedéséről és irányításáról,másrészt nem is eléggé szemléletes. Az egyszerübb vizuális ellenőrzés érdekében a programcsomag tartalmaz takartvonalas megjelenítést is; a kamera bármely pozíciójából külön kérésre elkészíttethető az adott vetület takartvonalas változata. Itt a már létező felületek takarhatják a modell többi részét,ezért a modell körüljárása révén az operátor jóval pontosabban érzékelheti,hogy nem követett-e el hibát. Az alkalmazott takartvonalas algoritmus a csúcs-él-felület hierarchikus leírás szerint dolgozó algoritmusokkal rokon,a megjelenő ábra - a grafikus display és a project egyéb szempontjainak megfelelően - a látható élek illetve éltöredékek hálózatát tartalmazza. A hasonló algoritmusokkal ellentétben azonban a modell nem csak síklapú felületekből állhat,hanem megoldottuk a másodrendü felületek feldolgozását is. Első lépésben kiszámítjuk a görbült felületek kontúrját /kontúr alatt értjük a felület azon pontjainak mértani helyét,ahol a felületnormális vektor merőleges a vetítési sugárra/,majd a modell-leírást ezekkel a kontúrokkal ideiglenesen kiegészítve határozzuk meg az egyes élek takartságát.

A programcsomag elkészítésénél az alapvető szempontunk a minél kisebb memóriafelhasználás volt. Ennek érdekében az alap-menü utáni programrészeket,valamint a takartvonalas algoritmust overlay-szervezéssel alakítottuk ki. A memóriában egyidejüleg mindig csak az a legszükebb szubrutin-együttes tartózkodik, amelyik az adott funkció elvégzéséhez éppen szükséges. Lényeges méretcsökkenést lehetett elérni azáltal is,hogy a programok által kezelt szöveges információkat nem tároljuk a memóriában. A program,interaktiv lévén,igen sok szöveges üzenetet tartalmaz

/pl. a menük,figyelmeztető szövegek,stb./; ezeket a diszken tartjuk,virtuális memóriakezeléssel érhetők el. A programcsomag 12K tárkiépítésü R-10 gépen már használható.

A programozási munka megkönnyítésére az Intézetben kifejlesztett PLM programozási nyelvet használtuk. A nyelv fordítását az MP/3 makroprocesszor végzi, makrodefiníciókból álló fordítóprogram alapján. A PLM nyelv lényegileg az assembly kódú programok egyszerü,gyors és hibamentes előállítására szolgál, kényelmesen és áttekinthetően fogalmazhatók meg ciklikus és feltételes utasítások,összetett aritmetikai kifejezések,szubrutinhívások,stb. Külső makrodefiníciók hozzáadásával a PLM könnyen bővíthető,az adott feladathoz leginkább alkalmazkodó,jól olvasható nyelv hozható létre.

Irodalom

1./ Braid,I.C.: Designing with Volumes
Cantab Press, Cambridge, England 1974

2./ Kimura,F.,Hosaka,M.: Program Package GEOMAP
Reference Manual, Univ.of Tokyo 1977

3./ Newman,W.,Sproull,R.: Principles of Interactive
Computer Graphics McGraw-Hill,New York,1973

4./ Sutherland,I.,Sproull,R.,Schumacker,R.: A Characterization of Ten Hidden-Surface Algorithms
ACM Comp.Surveys Vol.6, 1974

5./ Báthor M.: Interactive Picture Manipulating
2. Magyar Szám.tud. Konf. 1977

6./ Baumgart,B.: Geometric Modeling for Computer
Vision Stanford AI Lab.,Memo no. AIM-249
Stanford University 1974

7./ MODBUIL interaktiv 3D testmodellező programcsomag
felhasználói leírás MTA-SzTAKI /belső anyag/

8./ Interactive Disc Operating System IDOS
kezelési útmutató MTA-SzTAKI /belső anyag/

9./ PLM programozási nyelv leírás
MTA-SzTAKI /belső anyag/

10./ MP/3 makroprocesszor általános ismertetés
MTA-SzTAKI /belső anyag/

A PROGRAMOZÁS TECHNOLÓGIÁJA AZ ANSWER OPERÁCIÓS RENDSZERBEN

Bedő Á. - Langer T.
Számitógépalkalmazási Kutató Intézet

1. Szoftver-technológia

A számitástechnika elterjedése kezdetén a szoftver-készitésre mint munkafolyamatra az volt a jellemző, hogy egy-egy tehetséges mérnök, matematikus vagy jogász nekilátott és egyedül létrehozott valamilyen programot. Az igy elkészült programok egyedi alkotások voltak. Elkészitésükhöz komoly mesterségbeli tudás igényeltetett, s ha ez meg is volt, akkor létrejött a szoftver, amelyet ezután jóleső érzéssel a saját egyedi alkotásának tekinthetett minden programozó.

Ahogy egyre komolyabb feladatok nagyobb igényü megoldására kivántuk használni a számológépeket, úgy egyre inkább nőtt a feladat megoldásában szerepet játszó szoftver-elemek mérete és száma. Ma már általában nem egyedi szoftver programokra van szükségünk, hanem szoftver rendszereket épitünk.

Mostanában szivesebben veszünk egy nem túlságosan ötletesen megirt, azonban a rendszerekben jól használható szoftver-elemet.

A szoftver-készitésnek mint közösségi munkafolyamatnak a megismételhetősége alapvető követelménye szakmánknak. Ennek érdekében ki kell fejlesztenünk, illetve tudatosan alkalmaznunk kell valamiféle technológiát. A szoftver rendszer akkor tekinthető ipari terméknek, ha előre meghatározott minőségi követelményeknek eleget tesz és a javitása, karbantartása, valamint használhatósága biztositva van.

Megállapithatjuk azt, hogy a szoftver-technológia azon módszereknek és eszközöknek az együttese, amelyek alkalmazása lehetővé teszi a szoftver-készitésnek és -alkalmazásnak, mint közösségi munkafolyamatnak célszerü szervezését és megismétlését.

2. A szoftver-készitès folyamata

A szoftver-készitès folyamán célszerünek látszik az alább felsorolandó hat lépést megkülönböztetnünk egymástól.

Tudjuk, hogy a szoftver-készitès folyamata nem abból áll, hogy a hat lépést egymásután végrehajtjuk. A legtöbb lépés megtétele után ellenőrzésnek kellene következnie, s ha ekkor úgy látjuk, hogy céljainktól valamilyen nem elviselhető mértékben eltértünk, akkor az előző lépések valamelyikétől ismételnünk kell a már végrehajtott lépéseket. /Van, amikor teljesen elölről kell esetleg kezdenünk a munkát./

A fent emlitett hat lépés a következő:

1. A megoldandó feladat megértése és megfogalmazása.
2. A feladat és a rendelkezésre álló számitástechnikai eszközök elemzése és a szoftver rendszer tervezése.
3. A rendszer megvalósitása menetének megtervezése, a munka megszervezése.
4. A rendszer elemeinek elkészitése.
5. A rendszer felépitése.
6. A rendszer müködésbe helyezése, karbantartása, fejlesztése.

3. A szoftver-készitès folyamán alkalmazandó módszerekről

Abból adódóan, hogy a szoftver-készitès közösségi munkafolyamat és valamilyen terméket eredményez, a termék előállitása folyamán bizonyos átfogó - minden lépésben alkalmazandó - módszerekre is szükségünk van.

Ilyenek:

- az irásbeliség módja
  /mit kell leirni, hogyan, mikor, kinek/
- a közlések módja
  /a munkatársak hogyan tájékoztatják egymást, illetve hogyan tájékozódnak a társuk munkájáról, eredményeiről, bizonyos változó követelményekről/
- az irányitás módja
  /vezetők és vezetettek kapcsolata, az irányitás eszközei/

- az ellenőrzés módja
  /a termék minősége milyen, hogyan dolgoznak a munkatársak, határidővel hogy állunk/

E módszereken kivül természetesen a szoftver-készitás fent leirt folyamata minden lépésében módszeresen kell tevékenykednünk.

4. Alaptétel: A módszerek alkalmazásához azokat segitő eszközökre van szükség.

5. Az ANSWER operációs rendszer kifejlesztésének célja

Tevékenységünk több éve arra irányul, hogy a szoftver-készités folyamatát a tervezéstől a fejlesztésig számológéppel segitsük.

Munkánk során arra az alapvető ellentmondásra bukkantunk, hogy a jelenleg használatban lévő operációs rendszerek /OS, DOS, stb./ nem igazán alkalmasak programok, illetve programrendszerek fejlesztésére.

Véleményünk szerint ennek az a természetes oka, hogy e rendszerek nem a programfejlesztés segitését tüzték ki maguk elé célként, hanem a számológép mint nagyértékü berendezés erőforrásainak optimális felhasználását kivánták lehetővé tenni; bizonyos feladatokat megoldó programrendszerek hatékony futtatása volt e rendszerek elkészitésének célja.

6. A programfejlesztést segitő operációs rendszerekkel szemben támasztandó és összehangolandó követelmények

A. Lehetővé kell tenni az assembly programozás teljes elhagyását.
B. A rendszernek meg kell oldania a szoftver hordozásának a problémáját /Főképpen a nagygépről kisgépre való átvitelt/.
C. A programozói adatfeldolgozási munkát gépesiteni kell.
D. A rendszer lehetővé kell tegye azt, hogy benne előre megadott minőségü programokat lehessen készitени.
E. A programfejlesztés párbeszédes /interaktiv/ folyamat kell hogy legyen.
F. A rendszernek a csoportmunkát kell támogatnia.
G. A rendszer módositható és bővithető kell legyen; magának a rendszernek is fontos a hordozása.

7. Az ANSWER programfejlesztő operációs rendszer fő részei

A. Nyelvi rendszer

Tapasztalataink azt mutatják, hogy a programkészitás szinvonalának emelése érdekében nem elegendő újabb és újabb magasszintü, de hatékony nyelvek forditóprogramjainak az elkészitése. A forditóprogramot olyan szoftver-környezetbe kell beágyaznunk, amellyel együttesen a nyelv megvalósitása a felhasználó számára egységesen és egymással tökéletesen összehangolt módon adja

- a programirás, programjavitás,
- a programkipróbálás,
- a programtárolás,
- a programok mérése és
- sokféle kód generálása lehetőségét.

Az ilyen szoftver-csomagot /vagy alrendszert/ nevezzük nyelvi rendszernek.

Az ANSWER rendszerben CDL2 nyelvi rendszer lesz.

B. Dokumentációs rendszer

A dokumentációs alrendszer célja az, hogy mindenféle szöveges /nem programszöveg/ információt tároljon, készitését, módositását olvasását lehetővé tegye. Az ANSWER rendszerben az irásbeliség azt jelenti, hogy a dokumentumoknak a dokumentációs rendszer file-jaiban van a helyük. Minden papiron lévő dokumentum a file-ok másolata.

Ezenkivül a dokumentációs rendszer postaszolgálatot is ellát, segitve a munkatársak kapcsolatát.

A dokumentációs alrendszer az ANSWER élete során információs alrendszerré bővül. A bővités legfontosabb lépése az lesz, hogy a vezetési információk kezelését is el fogja látni.

C. A rendszer magja

A munka viszonylag korai fázisában kiderült, hogy rendszerünk nem tud felépülni a jelenleg használatos operációs rendszerek szolgáltatásait használva. Ennek két oka van:

- mint már emlitettük e rendszereket más célból hozták létre igy számunkra bizonyos szolgáltatások hiányzanak, illetve a rendszerek belső felépitése, logikája ellenkezik a mi követelményeinkkel;
- e rendszerek rengeteg, a programfejlesztés szempontjából teljesen fölösleges részt tartalmaznak.

Ezért elhatároztuk, hogy önálló rendszert készitünk. Az önálló rendszer felépitését lehetővé tevő alapvető tevékenységeket, illetve alapfogalmakat e rendszer magja /supervisor+file-kezelő/ valósitja meg.

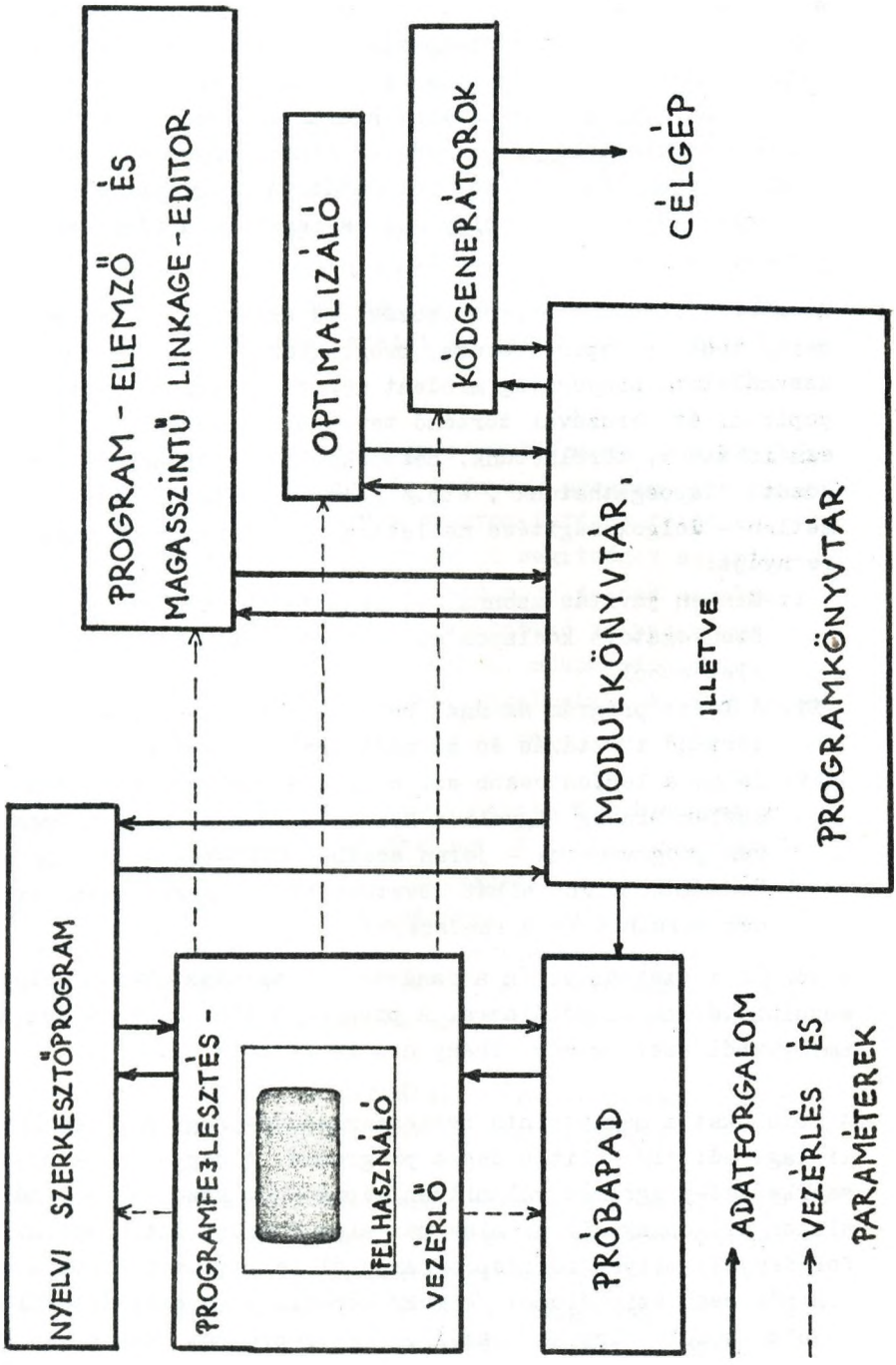
Ezek a fogalmak, illetve tevékenységek:
program, processz, perifériák, file-ok, interaktivitás /milyen lehet a párbeszéd/, I/O, gépek kapcsolata, naplózás, stb.

A rendszer magjáról a SOFTECH-sorozat megfelelő köteteiből lehet többet megtudni.

A dokumentációs alrendszerről ezen a konferencián két előadás is elhangzik: Szőke Péter: Az ANSWER dokumentációs rendszere és Aszalós János: Az ANSWER dokumentációs rendszerének axiomatikus leirása.

Az ANSWER rendszer első változatának kifejlesztésével párhuzamosan folynak a majdani szoftver-tervezési alrendszer kialakitására irányuló kisérletek. Jelenleg a PROLOG nyelvnek ilyen használatát vizsgálják az SzKI-ban és a NIM IGÜSzI-ben.

8. Az ANSWER nyelvi rendszerének felépitése



9. A programozás folyamata az ANSWER nyelvi rendszerében

Induljunk onnan, hogy a feladatot megfogalmaztuk valahogy, és a programterv is elkészült. A programterv lényegileg a program nagybani felbontását tartalmazza. Meghatározza a programot alkotó modulokat, rétegeket és szekciókat - ezek a CDL2-fogalmak a különböző szempontok szerint hierarchikusan létrejött, logikailag összetartozó programrészek elnevezésére szolgálnak -, valamint ezen alkotórészek feladatát. A programterv ismeretében a programozó leül a display elé és megkezdi a tényleges programozást, begépeli a moduljait.

Az ANSWER rendszer a programozás eme szakaszától kezdve feleslegessé teszi a papir, ceruza, programkártyák és programlisták használatát. Lényegileg mindent megcsinálhatunk benne, ami a papirral és ceruzával történő tevékenységben előnyös /folyamatosan irhatunk, törölhetünk, beszúrhatunk, kész programrészeink között "lapozgathatunk", stb./ Ezen - egyébként eléggé kellemetlen - dolgok segitése mellett a nyelvi rendszer számos előnyt is nyújt:

i. Minden javitás azonnal végrehajtódik, igy az összefirkált, áthuzogatott kódlapok és ezek másolgatásai kimaradnak az életünkből.

ii. A leirt program azonnal bekerül a gépbe, igy a kódlapról történő tisztázás és az adatrögzités lépése elmarad.

iii. Talán a legfontosabb az, hogy a számológép a program irása közben mindig figyeli a kezünket. Tudja, hogy milyen nyelven programozunk - jelen esetben CDL2-ben - és azonnal szól, ha szintaktikus hibát követünk el. Szintaktikusan hibás modul nem kerülhet be a rendszerbe.

A kódolási szakasz végén a rendszerben szintaktikusan helyes modulok várnak kipróbálásra. A programot több modul alkotja, amelyekből esetleg még néhány nem is készült el.

A modulokat a magasszintü összeszerkesztő-program /high level linkage editor/ állitja össze programmá. A hagyományos összeszerkesztő-programok valamilyen gépikódhoz közel álló bináris alakon dolgoznak. Ez az alak már alig tükrözi azt, hogy milyen forrásnyelv milyen forrásprogramjából keletkezett. Ezen a szinten már csak szimbólumok /cimek/ képezik a kapcsolatot különböző modulok között. Igy az összeszerkesztő-program csak olyan inter-

face-hibát képes jelezni, hogy a modulban használt szimbólum sehol sincs definiálva, vagy hogy egy szimbólum többszörösen van definiálva. Ráadásul ezek a szimbólumok valamely belsőkódú azonositók, igy az összeszerkesztő-program hibaüzeneteiből elég nehéz következtetni, hogy mi is volt a baj a forrásprogramban. Olyan finomabb interface-hibát a szerkesztőprogram csak igen körülményesen képes felderiteni, mint például az, hogy valamely átadott eljárás tipusa, paramétereinek száma vagy paramétereinek tipusa nem egyezik az alkalmazás és a definiálás helyén. Makrók, illetve konstansok átadása is igen nehézkes már ezen a szinten. A fentiekben felsorolt hiányosságokat küszöböli ki a magasszintü összeszerkesztő-program, amely még az összes forrásinformációt tartalmazó és a forrásprogram szerkezetét teljesen tükröző belső alakon dolgozik.

A magasszintü összeszerkesztő-program müködésének eredményeképpen tehát összeáll az /esetleg még hiányos/ program. Programunkat úgy szeretnénk kipróbálni, hogy a program futását teljesen ellenőrzésünk alatt tartsuk. Ehhez nyújt segitséget az interaktiv próbapad. Rajta a program hiányzó moduljainak az algoritmusait szimulálni lehet, futását különböző helyeken meg lehet szakitani, adatait le lehet kérdezni, müködéséről adatokat lehet gyüjteni. Mindezt a forrásnyelv fogalmainak és a program azonositóinak használatával bonyolitjuk le.

Igy a próbapadon kiderithetünk számos hibát. Moduljainkat a nyelvi szerkesztőprogrammal javithatjuk. A nyelvi szerkesztőprogram olyan szintaxisvezérelt szövegszerkesztő, amelyben a programozási nyelv fogalmai segitségével jelölhetjük ki a javitandó részt, és hajthatjuk végre a javitást. Csak szintaktikusan helyes javitások hajthatók végre, a nyelvi szerkesztő nem hagyja a már szintaktikusan helyes modulokat elrontani.

A javitás-összeszerkesztés-kipróbálás ciklus néhányszori megismétlése után elkészül egy logikailag helyes program.

Eddig nem tértünk ki arra a kérdésre, hogy milyen gépre akarunk programot fejleszteni. Az ANSWER rendszerrel szembeni alapvető elvárás, hogy vele valamely nagyobb fejlesztőgépen tetszőleges célgépre lehessen programot fejleszteni. Eddig csak azt értük el, hogy a program müködik a fejlesztőgépen, ahol az ANSWER

rendszer üzemel. Ezután a már logikailag helyes programból kódot kell generálni a megfelelő célgépre, és az igy nyert kódot kell a célgépre átvinni. A megfelelő kódgenerátor természetesen rendelkezésünkre áll az ANSWER rendszerben.

A fejlesztőgépen bejáratott program csak nagyon ideális esetben lesz teljes egészében megfelelő a célgépen. Átvitel előtt a program gépfüggő részeit a célgépnek és a célgépen futó környezetnek megfelelően le kell cserélni. Ez a csere általában a program 1o-2o %-át érinti, és ha a programozási nyelv és a programozási módszer megfelelő, akkor ezen gépfüggő részek a program többi részétől jól elkülönülnek, s cseréjük nem nagy munka.

Természetesen az igy átvitt programot még a célgépen is be kell járatni. A kódgenerátoroknak van olyan üzemmódjuk, amelyben az átviendő kódba begenerálnak bejáratást segitő információkat. Igen lényeges, hogy az átvitt program jelentős részének a helyességéről már meggyőződtünk a fejlesztőgépen. Tehát a célgépen a bejáratás lényegesen kevesebb erőfeszitést igényel, mintha teljesen új programmal kezdenénk. Sőt igy a program fejlesztésével töltött munkamennyiség is várhatóan kisebb, mert a fejlesztőgépen a programfejlesztést segitő szoftver rendelkezésünkre áll, szemben a célgépen létező helyzettel.

Végezetül ismertetjük rendszerünk további néhány vonását, amely a programfejlesztés fentebb vázolt folyamatának több szakaszát is jellemzi.

i. A programozó teljes interaktiv kapcsolatban áll a rendszerrel. A rendszer "karakterenként" figyeli a felhasználó tevékenységét, és a felhasználó minden utasitásának azonnal látja a hatását.

ii. A rendszer egyetlen nyelven történő programozást segit maximálisan. Igy lehetővé válik, hogy a programozó a programjaival kapcsolatos minden utasitását a forrásnyelv fogalmaival és a forrásprogram azonositóival fejezze ki. A rendszer programjai pedig tudják, hogy az általuk kezelt objektumok CDL2 programok, igy ennek megfelelően ellenőrizhetik a programozó tevékenységét.

iii. Az ANSWER rendszer az optimalizálással kapcsolatos alábbi szemléletet támogatja. Amikor a programjainkat irjuk, akkor csupán algoritmusaink globális hatékonyságával törődjünk, - ez tervezési kérdés -, és arra törekedjünk, hogy minél áttekinthetőbb, jól strukturált programot irjunk, mellőzzünk minden csábitó optimalizálási trükköt. A hatékonysági szempontokat félretéve törekedjünk arra, hogy minél gyorsabban hozzuk létre a logikailag helyes programot. A programozó által alkalmazandó legtöbb optimalizálást a forditási rendszer úgyis elvégzi. Ha még sem vagyunk megelégedve programunk minőségével, mérjük ki a hatékonytalan, leggyakrabban futó részeket. Tapasztalat szerint a programok a futási idő 9o-95 %-át a programszöveg 5-1o %-ában töltik. Ezeket a részeket optimalizáljuk, most már minden lehetséges trükköt felhasználva. Ezek kicsiny részek, igy a program szerkezete már nem romlik el. Igy megkiméljük magunkat attól, hogy olyan részek optimalizálásába fektessünk energiát, amelyek alig futnak. A memóriaigényt pedig éppen a nem bolygatott részekben csökkenthetjük a futási idő rovására.

Információk

A nyelvi szerkesztővel, az interaktiv próbapaddal, az interaktivitás és a nyelvre szabottság kérdésével foglalkozik Bárány Sándor és Bolgár Gábor jelen konfe4encián elhangzó "Nyelvre szabott interaktivitás az ANSWER programfejlesztő rendszer" cimü előadása.
A szoftver-technológia témán dolgozó munkatársaink tanulmányaikat és munkadokumentumaikat a SOFTECH sorozatban jelentetik meg. Ezek kötetei a Programozási Rendszerek Főosztályán megtalálhatók, illetve igényelhetők.

DIL FORDITÓPROGRAMOK KÉSZITÉSÉNEK TAPASZTALATAI

Biró Ágnes - Komor Tamás
Számitógépalkalmazási Kutató Intézet

A DIL /DATASAAB Interpretative Language/ a D5 gépcsalád magasszintü programozási nyelve. A DIL programból D5 gépi kódot előállitó rendszer tartalmaz egy DAL assembly nyelvre fordító programot, egy DAL nyelven megirt értelmező programot, a DAL assemblert és egy szerkesztő programot. E rendszert először a DATASAAB cég készitette el, majd 1974-ben megbizta az INFELOR-t /ma SZÁMKI/, hogy készitse el a rendszer új változatát. Ennek elkészülte után 1976-ban a VIDEOTON Fejlesztési Intézet szerződést kötött a Számitógépalkalmazási Kutató Intézettel a DIL fordítóprogram R1o gépre történő kidolgozására.

Jelen dolgozatban a DIL fordítóprogramok kidolgozásának néhány tapasztalatát kivánjuk közzétenni. Ehhez nincs szükség a DIL nyelv, illetve a rendszer ismertetéséhez, igy erre előadásunkban nem is térünk ki. Tájékoztatásul közöljük, hogy a DIL rendszer rövid jellemzését adja [2], és a DAL assemblerről már az 1975-ös konferencián elhangzott egy előadás [3].

1. DTS-DIL fordítóprogram

A DIL rendszer eredeti változatát a DATASAAB cég a D22-gépek ALGOL-GENIUS nyelvére dolgozták ki. /Az ALGOL-GENIUS lényegében ALGOL 6o nyelv füzérek kezelésével kiegészitett változata/. Később felmerült a rendszer IBM gépekre történő átvitelének igénye. Ezért határozták el a rendszer újraprogramozását az IBM és a D22 gép közötti hordozhatóságot biztositó COBOL nyelven.

1.1 Fordítóprogramot irni COBOL-ban? Első hallásra megdöbbentő volt számunkra is. Utólag vizsgálva a kérdést azonban azt mondhatjuk, hogy programozástechnikai szempontból a feladat nem volt nagyon nehéz. A meglehetősen dinamikus táblakezelést is COBOL

nyelven programoztuk, igaz, hogy ehhez egy egész szubrutinkészletet kellett irni. Assembler nyelven csak két kis modult kellett irnunk, amelyek az egész fordító 1 %-át alkotják. Természetesen a fordítóprogram hatékonyságát már erősen rontotta az, hogy COBOL nyelven készült, ezt majd az RIoDIL-lel való összehasonlitás fogja jellemezni /lásd a 2.1 pontot/.

Ugyanakkor bebizonyosodott, hogy a COBOL valóban elősegiti a hordozhatóságot, bár természetesen nem oldja meg teljesen a problémát. A COBOL nyelv IBM-en és D22-n használt változatainak eltéréseit két csoportra oszthatjuk.

a, "Lexikális" eltérések. Ilyenek például:

D22-változatban:	IBM-változatban:
"	,
COMP-O	PIC....COMP
ENTER LINKAGE.	
RETURN.	GOBACK.
ENTER COBOL.	

Ezeknek a cseréjét a D22-ről IBM-re való áttéréskor a DAISY szövegszerkesztő program [1] segitségével automatizáltuk.

b, Eltérő, vagy csak az egyik COBOL változatban megengedett utasitások, illetve szintaktikus konstrukciók.
Ezeknek a használatát igyekeztünk elkerülni /pl. REPORT WRITER FEATURE, COBOL belső overlay, eltérő korlátozások a szóköz használatára relációjelek mellett és pont előtt, stb./ A fennmaradó esetekben /pl. SELECT, ENTER, CURRENT-DATE utasitások/ a megfelelő változtatásokat kézzel kellett átvezetni, ez azonban a program csekély részét érintette.

1.2 A DTS-DIL fordító COBOL programjának elkészitéséhez a megrendelő DATASAAB cég rendelkezésünkre bocsájtotta a fordító ALGOL-GENIUS nyelvü mintegy 2oooo soros, 4o modulból álló programját. Ezt a "dokumentációt" csak a bemenő nyelv felhasználói szintü leirása, a programban alkalmazott mintegy 15o közös változó listája, valamint a fordító által felépitett táblázatok és munkafájlok strukturájának leirása egészitette ki. A fordítóban al-

kalmazott algoritmusok felderitéséhez és az output specifikálásához meg kellett fejteni az ALGOL-GENIUS programot!
A méreteket figyelembe véve ez még jól strukturált program esetén is igen kemény feladat, ez pedig sajnos nem volt az.
A munkában Bánkfalvi Zsolt vezetésével több programozó vett részt. A munka stilusa nem volt teljesen egységes.
A "kódorientált" programozók a modulok algoritmusát csak vázlatosan deritették fel, és a COBOL programot részben az eredeti program mechanikus átirásával állitották elő. A "tervorientált" programozók részletekbe menően megfejtették az eredeti programot, és csak ezután kezdtek hozzá az újraprogramozáshoz. Az első csoportba tartozók gyorsabban jutottak el a programok megirásához, de ezt az előnyt elveszitették a belövés során. A két módszer összehasonlitásánál figyelembe kell venni még a következőket. A tervorientált programozók moduljaikról használható dokumentációt is készitettek /amelyet az RloDIL készitésnél fel is használtunk./ A tervorientált módszer lehetővé tette az eredeti programstruktúra javitását. Mindezek alapján a tervorientált módszert jobbnak tartjuk.

Ezt a következtetést megerősiti a több mint két éve folyó karbantartás tapasztalata is. Egyrészt a tervorientált modulokban lényegesen kevesebb hiba jelentkezett utólag, másrészt ezek javitása általában könnyebb, mint a kódorientált részek hibáinak javitása. A karbantartást a megrendelővel évenként megújitott szerződés alapján végezzük. A felhasználók /svéd, amerikai és norvég cégek/ hibajelzéseit a DATASAAB kapja meg, ellenőrzi, és listával dokumentálva küldi meg nekünk. Az első hónapoktól eltekintve évente mintegy 5 hibajelzést kaptunk. Ezek többsége hibás DIL-forrásprogram feldolgozására vonatkozott.

1.3 Végezetül néhány adat a munka méretéről és a ráforditásokról.
A DTS-DIL forditóprogram 2oooo forrássorból áll. Az IBM változat gépi kódja 4ooK byte, amelyet erős overlay-ezés következtében 14oK-s particióban lehet futtatni /ebből mintegy 5oK a táblázatok számára fenntartott terület/. Hasonló nagyságú a D22-es változat is. A programhoz 3oo oldalas angol nyelvü dokumentáció készült. A munkát tapasztalt rendszerprogramozók végezték, az összes ráforditás 6,5 ember/év /karbantartás nélkül/. A gépidőről sajnos nincs adatunk, de ennek értékelését amúgyis megnehezitené az, hogy a forditó belövését részben a D22 gépen végeztük, álta-

lában napi egyszeri fordulással. Ezenkivül néhány hétig módunk volt egy kártyaolvasóból és sornyomtatóból álló egyszerü RJE terminálon keresztül használni a megrendelő IBM gépét /ezalatt óvatos becslések szerint is az itthonihoz képest négyszeres teljesitményt értünk el/.

2. Az RloDIL forditóprogram

Az RloDIL fordító kidolgozásában a fő problémát a DIL fordító mérete jelentette, mert az eddigieknél lényegesen kisebb memóriájú gépre kellett kidolgozni. Továbbá kérdéses volt, hogy milyen módszerekkel tudjuk elfogadható idő alatt elkésziteni a viszonylag nagy programot a nem éppen programfejlesztésre orientált Rlo gépen.

2.1 A megoldást a megfelelő implementációs nyelv és a forditóprogram újratervezése hozta meg.
Implementációs nyelvnek a CDL-t /Compiler Description Language/ választottuk, amely igen alkalmas forditóprogramok irására.
Az Rlo gépen évek óta jó tapasztalatokkal használják a CDL modulfordító változatát, amely a mi munkánkat is nagymértékben megkönnyitette.
Az újratervezés során első célunk az volt, hogy a régi kódorientált modulokat is "tervorientált" módon irjuk le úgy, hogy a CDL szöveget könnyen elő tudjuk állitani. Ekkor már rendelkezésünkre állt a DTS-DIL üzemszerü használata során szerzett jónéhány tapasztalat és hibajelentés. Ezeket is figyelembe vettük az újratervezésnél.

Az újratervezésnél a második cél a modulok és a munkaterület /nagyszámú táblázat tárolására/ méretének minimalizálása volt. Ugy döntöttünk, hogy sem az operációs rendszer file kezelőjét, sem a CDL input-output rendszerét nem használjuk, mert mindkettő igen memóriaigényes és számunkra nem kényelmes szolgáltatásokat nyújtanak. Azokat a központi modulokat, amelyek a legtöbbet használt rutunokat /táblázatok kezelése, input-output tevékenységek, nyomkövetési funkciók/ tartalmazzák, assembly nyelven irtuk meg. Ezzel nemcsak a modulok mérete csökkent, hanem gyorsabbak is lettek /más módszerek, szótárkezelési eljárások bevezetésével is igyekeztünk a futási időt csökkenteni/.

A munkaterület méretét a táblázatok gazdaságosabb kitöltésével esetenként több táblázat átszervezésével, összeolvasztásával igyekeztünk csökkenteni. A D22 és IBM gépeken a táblázatokban szereplő numerikus adatok nagyságuktól függetlenül, mindig egy gépi szóban nyertek elhelyezést, ez adatonként 3, illetve 4 byte-ot jelentett. Az Rlo változatban a táblázatok numerikus adatait nagyságuktól függően a minimálisan szükséges számú byteban tároljuk, ezzel jelentős memória megtakaritást értünk el.

További problémát jelentett az Rlo szerkesztő programja, amely maximum 256 szekcióból álló programot tud összeszerkeszteni /a CDL által generált assembly program igen sok szekciót tartalmaz/, ezért a forditóprogramot eleve 8 különálló, szekvenciálisan végrehajtható programra bontottuk. Ezen programok további overlay-ezésére az előzetes tervekkel szemben nem volt szükség.

A feladatot végeredményben sikerrel oldottuk meg. Megrendelőink az RloDIL átvétele előtt IBM gépen a DTS-DIL forditóprogrammal dolgoztak, igy az RloDIL üzemszerü használatának kezdetén alkalmunk volt összehasonlitó vizsgálatokat végezni a két forditóprogramra vonatkozóan. Egy kb. 2oooo soros forrásprogram leforditása IBM gépen lo perc CPU időt és 23 perc teljes időt vett igénybe, ugyanez a program Rlo-en 22 perc alatt fordult le, illetve a forráslista letiltása esetén az RloDIL forditóprogram futásideje mindössze 9 perc volt.

2.2 Az RloDIL tervezésekor figyelmet forditottunk a tesztelésre is. Felmértük, hogy milyen módon segithetnénk elő az egyes modulok és az összeépitett forditóprogram tesztelését. A CDL nyelv beépitett nyomozási segédeszköze, a TRACE funkció, önmagában nem oldotta meg a problémát, a következők miatt:

i. Csak a meghivott szabályokról generál információt, a makrókról nem.

ii. A paraméterek értékét belépéskor irja ki, a kimenő paraméterek esetében ez semmitmondó. A vektorok tartalmáról semmi információt sem biztosit.

iii. Egy teljes modul nyomozása általában felesleges, viszont különböző részek nyomozásához újra kell forditani a CDL programot. Ezért úgy döntöttünk, hogy egy külön modult irunk a tesztelés elősegitésére, ezt NYOMOZ-nak neveztük el. Ez a modul a hivási paraméterektől függően különböző információkat

ir ki, pl. egy vagy több változó értéke, egy vektor tartalma hexadecimálisan vagy karakteresen, stb. A NYOMOZ modul megfelelő helyről való hivásával a i. és ii. probléma megoldódik.

A iii. probléma megoldására vezettük be a nyomozási szint fogalmát. A NYOMOZ modul hivásának egyik paramétere a kérés szintje. A kért információt a modul csak akkor szolgáltatja, ha ez a szint kisebb egy globális szintnél, amelyet az RloDIL fordító egyes programjainak hivásakor lehet beállitani. Bevezettük azt a konvenciót, hogy az egyes modulok közötti interface-t jellemző adatokat a legkisebb szinttel /1 v. 2/ nyomoztatjuk, a modulok vezérlő strukturáját középszinten, a részletes algoritmusokat pedig a legmagasabb szinteken nyomoztatjuk. Ilymódon a globális szint megadásával újrafordítás nélkül tudtuk vezérelni a kivánt információ mennyiségét. A gyakorlat azt mutatta, hogy a NYOMOZ modul használata megkönnyiti a modulok belövését, és különösen hasznosnak bizonyult a fordító integrálása során. Legnagyobb hasznát azonban a karbantartás során vettük.
Ugyanis a NYOMOZ alacsony és közepes szintü modul hivásait nem távolitottuk el a már müködő forditóprogramból sem, és igy az üzemelés során jelentkező hibák behatárolása a megfelelő globális nyomozási szint megadásával nagyon könnyü volt. Ugyanakkor a müködő fordító hatékonyságát a NYOMOZ modul hivásai nem befolyásolják lényegesen. Ennek a módszernek az alkalmazását feltétlenül javasoljuk viszonylag nagy programok batch környezetben történő kidolgozása esetén.

2.3 Az RloDIL forditóprogram "tervorientált" újratervezése, a COBOL nyelvü DIL fordító üzemeltetési tapasztalatainak felhasználása a forditóprogram megirásakor jó programtermék megszületését eredményezte. Az eddigi üzemeltetési tapasztalatok azt mutatják, hogy az átlagos felhasználók által irt programok esetében az RloDIL diagnosztikai képessége megfelelő, és a forditóprogram helyes kódot generál. Ezt mutatja az is, hogy a VIDEOTON FI féléves üzemeltetés során egyetlen hibáról küldött feljegyzést. Ugyanakkor Intézetünkben egy III. éves programozó szakos hallgató diplomafeladatként az RloDIL fordító minőségvizsgálatát végezte el. Körülbelül 150-200 szintaktikusan hibás tesztprogramot forditott le. A hibajelzések döntő többsége helyes és elegendő információt nyújt a felhasználónak a hiba kija-

vitásához. A hibajelzéseknek mintegy 5 %-a redundáns vagy félrevezető oly módon, hogy a hiba kijavitása nem triviális a hibaszöveg alapján. A leforditott programokban mindössze 2 vagy 3 olyan hiba volt, amelyekre nem adott a forditóprogram hibajelzést, vagy a hibaszöveg félrevezető volt. Ezekben az esetekben a forditó hibás kódot generált. A félrevezető, redundáns vagy súlyosabb esetekben hiányzó hibajelzések főként azokra a modulokra jellemzők, amelyekbe az eredeti ALGOL-GENIUS programbeli hibajelzéseket változtatás nélkül ültettük át.

2.4 Végezetül néhány adat az RloDIL méretéről és a ráforditásokról összehasonlitásul a DTS-DIL-lel. A forditóprogram 12ooo CDL sorból és 9ooo assembly sorból áll. A gépi kód 15oK byte. 64K byte-os gépben a táblázatok számára 2o-25K byte-nyi terület marad. A forditóprogramhoz 5 kötet, összesen 6oo oldalnyi dokumentáció készült. Az összes ráforditás karbantartás nélkül 8 ember/év, a munka során 5oo óra gépidőt használtunk el.

IRODALOMJEGYZÉK:

Gyárfás A.-Jónás G. DAISY. CDC 33oo Felhasználói ismertetők 4/1974.

Komor T. A DIL programgeneráló rendszer. Információ Elektronika 1977. 6. sz.

Szentes R. A DAL assembly nyelv és megvalósitása magasszintü programozási nyelven. Programozási Rendszerek '75, Szeged, 1975.

T R A N S F O R M
ESZ/OS OFF-LINE INPUT PROGRAMRENDSZER

Bölcsföldi József
Egyetemi Számitóközpont

1. A TRANSFORM rendszer célja

Az OS operációs rendszer nincs kellőképpen felkészülve a rendszeridegen kódú információk fogadására.
A TRANSFORM rendszer célja ennek a hiánynak a pótlása.

2. A TRANSFORM rendszer funkciója

A TRANSFORM programrendszer

- tetszőleges adathordozón /lyukkártyán, lyukszalagon, mágnesszalagon, mágneslemezen, stb./ elhelyezett
- tetszőleges paritásu vagy paritás nélküli, 5,6,7 vagy 8 csatornás shift nélküli, vagy legfeljebb 2 shiftes kódban lévő
- mező illetve rekord terminátorokkal vagy azok nélkül rögzitett
- karakteres

információt transzformál

- mágnesszalagra vagy mágneslemezre
- EBCDIC kódba

- az input rekordtartalom változtatása nélkül vagy az input rekordtartalom alábbi változtatásaival
  - transzformálandó fix számu karakterből álló mező
  - kihagyandó fix számu karakterből álló mező
  - kihagyandó adott mezővégjelig tartó változó hosszuságu mező
  - transzformálandó adott mezővégjelig tartó, adott maximális hosszuságu mező, feltöltés nélkül
  - transzformálandó adott mezővégjelig tartó mező adott maximális hosszuságura /ha szükséges, adott karakter segitségével balról történő feltöltéssel/
  - transzformálandó adott mezővégjelig tartó mező adott maximális hosszuságura /ha szükséges, adott karakter segitségével jobbról történő feltöltéssel/

3. Megszoritások, korlátozások

A TRANSFORM rendszer az alábbi megszoritásokkal, korlátozásokkal müködik.

3.1 Az input kódrendszer bármilyen paritásu vagy paritás nélküli 5,6,7 vagy 8 csatornás, shift nélküli vagy legfeljebb 2 shiftes kódrendszer lehet.

3.2 Maximum 256 féle rekordkép ill. maximum 256 féle mezőkép adható meg egy transzformálandó állományhoz.

3.3 Valamely mezőképben megadott mezőhossz 256-nál nagyobb nem lehet.

3.4 A transzformálandó állomány csak karakteres lehet. A pakolt decimális vagy bináris inputot a rendszer nem értelmezi.

3.5 Az input bizonyos jelei, ha előfordulásuknak nincs jelentősége /pl. csupalyuk kombináció lyukszalag javitására/, az outputban már nem szerepelnek.

3.6 Az input rekord végjele nem kerül az outputba.

3.7 Az input rekord végét egynél több rekordvégjel is jelezheti; hatásuk egy rekordvégjel hatásával azonos.

3.8 Ha az input lyukszalag, akkor a rekordok darabolása nem megengedett /azaz egy rekord különböző részei nem lehetnek különböző lyukszalagokon, mert a kettéválasztás helyére rekordvégjel kerülhet/.

3.9 Ha az input rekord rövidebb, mint a neki megfelelő rekordkép által definiált hossz /azaz az input rekord végjele hamarább jön, mint ahogy a neki megfelelő rekordkép előírja/, akkor a soronlévő mezőt és rekordot lezártnak tekinti.

3.10 Fixnek specifikált mezőnél az adott hossznak megfelelő darabszámu karakter kerül az output mezőbe, kivéve, ha közben - illegálisan - rekordvégjel érkezik.

3.11 Ha egy mező hosszabb a neki megfelelő mezőképben megadott hossznál, akkor az egész rekord hibásnak minősül. /Ilyenkor a rekord nem az outputba, hanem hibalistára kerül./

3.12 A transzformált állomány mindig EBCDIC kódu és VB rekordformátumu.

4. A TRANSFORM rendszer felépítése

A rendszer az alábbi programokból és a hozzájuk tartozó JOB CONTROL eljárásokból illetve állományokból áll. /1.ábra/

4.1 A TRANSFORM rendszer programjai

A program

neve	funkciója	futtatandó
katalógus-program	a katalógusnak - a karakter - - karakter megfeleltetés szótárának - elhelyezése lemezkönyvtárban	kódrendszerenként egyszer
rekordképsor-program	a transzformálandó állománynak megfelelő rekordképsor elhelyezése lemezkönyvtárban	transzformálandó állományonként egyszer
transzformátor-program	a transzformáció elvégzése	transzformációnként egyszer

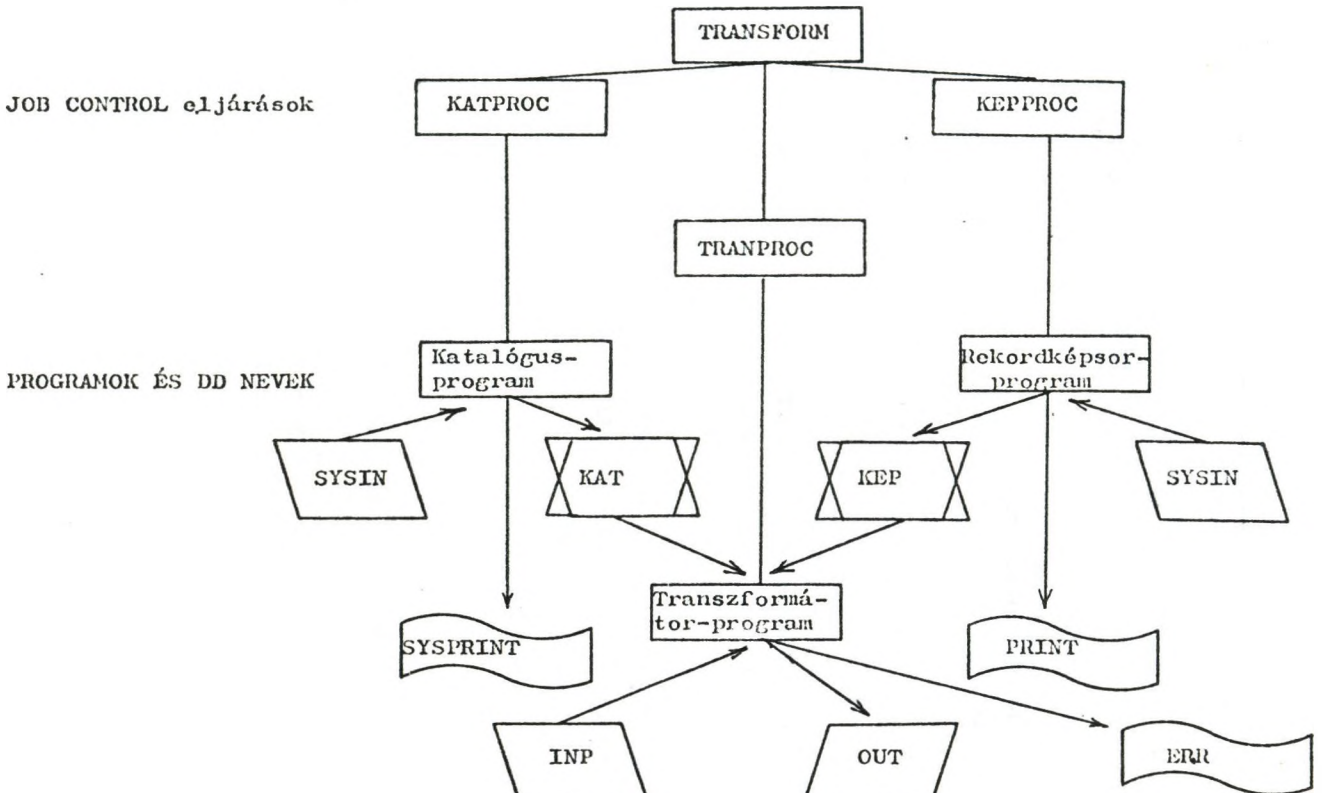
4.2 A TRANSFORM rendszer JOB CONTROL eljárásai

A JOB CONTROL eljárás

neve	funkciója
KATPROC	a katalógus-program aktivizálása, az általa használt állományok leírása
KEPPROC	a rekordképsor-program aktivizálása, az általa használt állományok leírása
TRANPROC	a transzformátor-program aktivizálása, az általa használt állományok leírása

1.ábra

A TRANSFORM rendszer felépitése



4.3 DD nevek a TRANSFORM rendszerben

A program neve	DD név	A DD névhez tartozó állomány tartalma
katalógus-program	SYSIN	a katalógus-program input adatai
	KAT,	a katalógus-könyvtár
	SYSPRINT	a katalógus-program normál- és hibaüzenetei
rekordképsor-program	SYSIN	a rekordképsor-program input adatai
	KEP	a rekordképsor-könyvtár
	PRINT	a rekordképsor-program normál- és hibaüzenetei
transzformátor-program	KAT	a katalógus-könyvtár
	KEP	a rekordképsor-könyvtár
	INP	a transzformálandó állomány
	OUT	a transzformált állomány
	ERR	a transzformátor-program normál- és hibaüzenetei

4.4 A TRANSFORM rendszer kommunikációs könyvtárai

A könyvtár

neve	egy memberjének		
	neve	mérete/byte/	tartalma
ESZKKAT	név	512	egy katalógus azaz a karakter-karakter megfeleltetés szótára: a transzformálandó karakter bináris értéke által meghatározott relativ címen a transzformált karakter áll.
ESZKKEP	név	max. 1552	
		ebből 4	prefixum szó
		256	bytenként 1 relativ szócim KEPSOR-1-től
		256	bytenként 1 transzformált inputrekord-végjel
		4	prefixum szó
	KEPSOR	1024	szavanként 1 mezőkép

Egy mezőkép tartalma a következő:

Ø.bit	1, ha ez az utolsó mezőképe a rekordképnek Ø, különben
1-7.bit	a mező tipusa /egy szám/
8-15.bit	a mező transzformált terminátora /ha fix hosszu mező, akkor a transzformált input-rekordvégjel/
16-23.bit	a mező feltöltő karaktere /ha nincs feltöltés, akkor definiálatlan/
24-31.bit	a mező hossza, ha fix hosszu mező a mező maximális hossza, ha változó hosszu mező

TIME SHARING SZOLGÁLTATÁS DOS/VS KÖRNYEZETBEN

Dalos Mihály, Gerl Zsolt, Simonyi Sándor, Tringer Éva
Számítástechnikai Koordinációs Intézet

BEVEZETÉS

A számítástechnika növekvő térhódítása, amit a felhasználói kör intenzív bővülése, a hazai géppark növekedése és a homogenizálódás, valamint a különböző számítástechnikai szolgáltatásoktól elvárt követelmények szintjének emelkedése egyértelműen mutat, a software tevékenység új formáinak, módszereinek kialakulását is elősegítette.

Az egyik ilyen irányzatot leginkább az jellemzi, hogy egy adott alkalmazói rendszer, szolgáltatás létrehozása nem egy programrendszer megírása útján, hanem többé-kevésbé általános kész programelemek, univerzális keretrendszerek felhasználásával és a konkrét igényekhez való alakításával lényegében rendszer építésnek nevezhető tevékenységgel történik.

Az előadás a Számítástechnikai Koordinációs Intézet IBM 370/125 rendszerének alkalmazásfejlesztési gyakorlatából mutat be egy tipikusnak tekinthető példát.

FELHASZNÁLÓI IGÉNYEK, RENDSZER LEHETŐSÉGEK

Számítógépünk felhasználói köre döntően intézeten belüli, az átlagosnál talán nagyobb szakismerettel rendelkező, elsősorban program, ill. rendszerfejlesztéssel foglalkozó munkatársaink közül kerül ki. Igy természetes, hogy igényeik a rendszerrel szemben egyre fokozódó mértékben és egyre türelmetlenebb formában jelentkeztek. Legkritikusabbnak a job fordulási idők csökkentése, ill. általánosabban a felhasználó és a rendszer kapcsolatának javítása, interaktivitás biztosítása tünt.

Számítógépünk alap operációs rendszere a Disk Operating System/Virtual Storage /DOS/VS/, számos előnyös tulajdonsága mellett azzal jellemezhető, hogy tipikusan batch üzemmódot biztosít, a felhasználó - rendszer kapcsolat terén szerény lehetőséget nyújt. Az IBM operációs rendszerei közül csupán a lényegesen nagyobb rendszerekre kidolgozott OS/VS2 rendelkezik a szóbanforgó problémát megoldó time sharing opcióval, ill. speciális lehetőségeket biztosít még a VM/370 Conversational Monitor System komponense. Meghatározott alkalmazási területekre léteznek jó megoldást adó IBM keretrendszerek /DB/DC rendszerek/, ezek azonban a vázolt igényeket csak egy-egy területen elégítik ki.

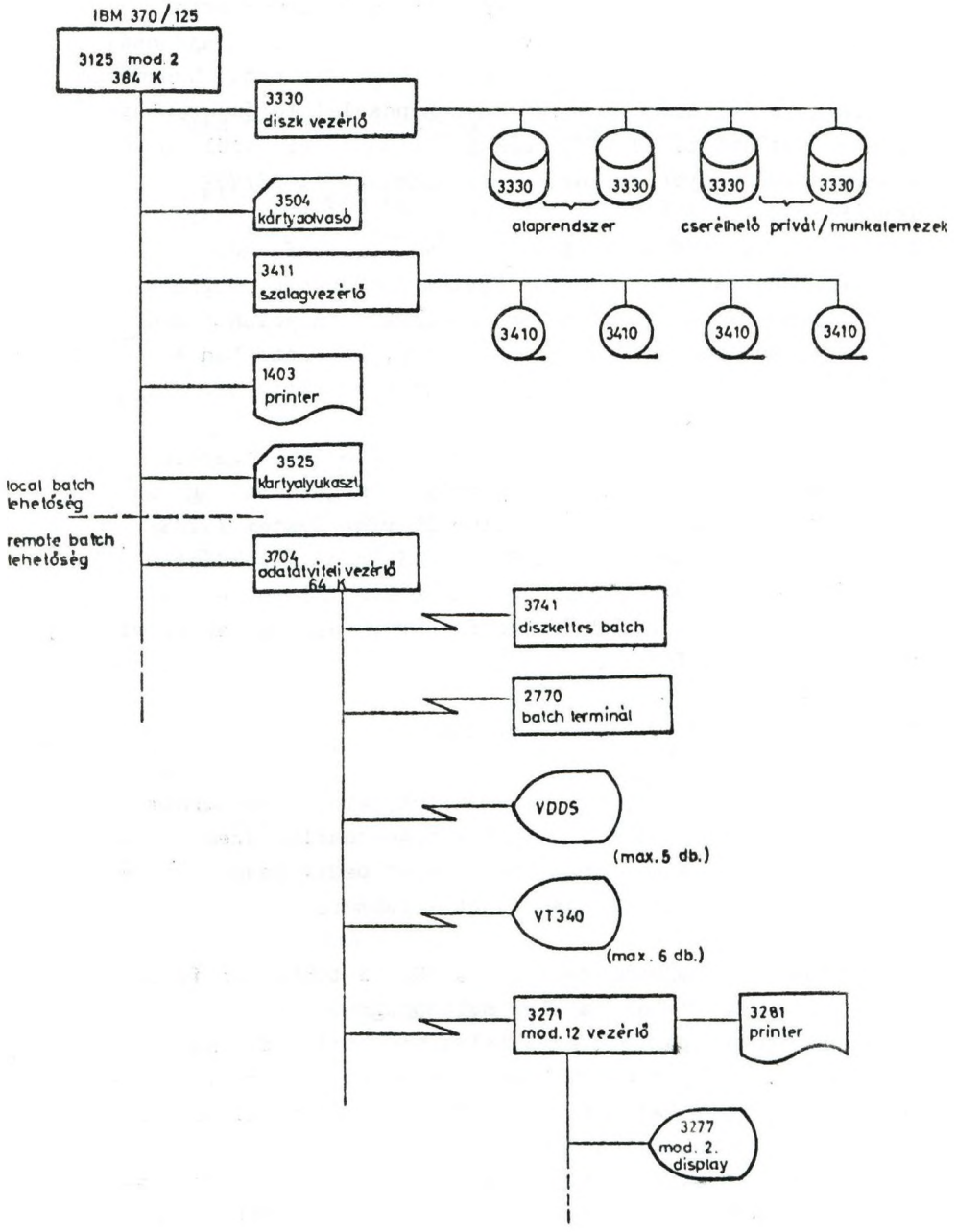
A DOS/VS saját erőből történő kiegészítése lehetőségeinket messze meghaladó feladat. A helyzet megoldását egy új IBM program product, az Entry Time Sharing System /ETSS/ bejelentése jelentette. Az ETSS általában jó megoldást adott, néhány vonatkozásban azonban eleve fel kellett készülnünk módosítására és kiegészítésére, valamint az üzemi feltételek biztosítására.

AZ ETSS MINT A DOS/VS KITERJESZTÉSE

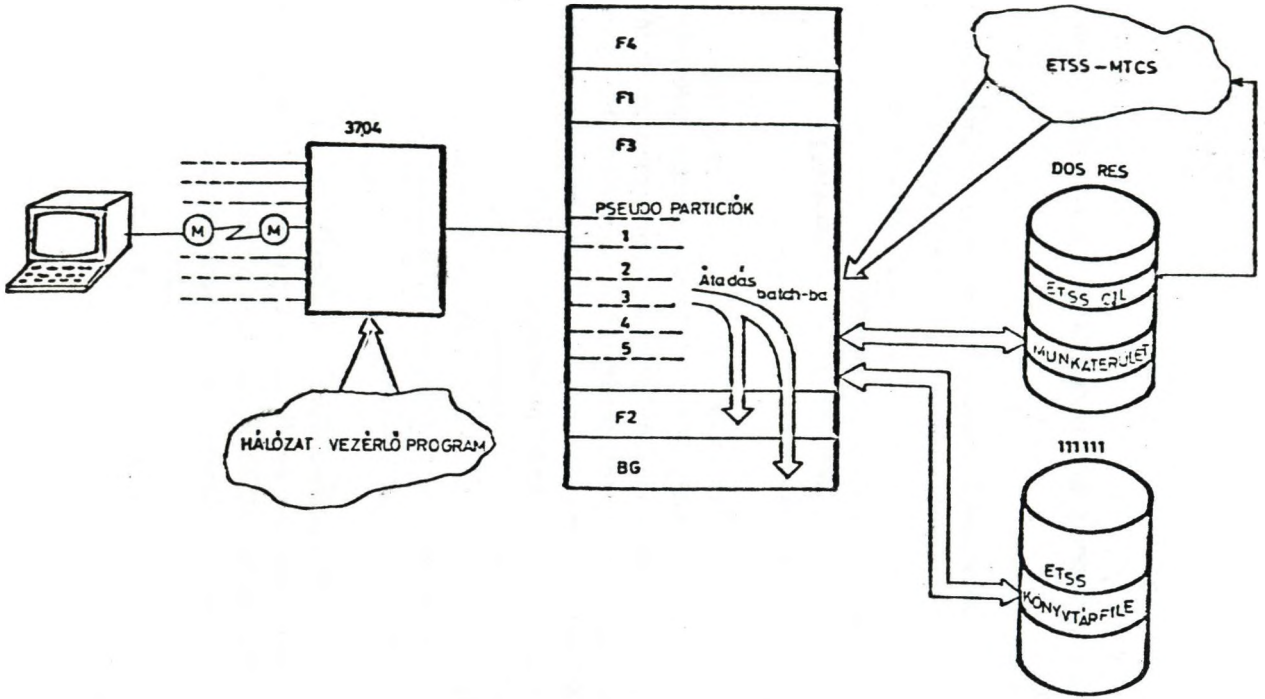
Az ETSS a DOS/VS egy partíciójában fut, mint rendszerkomponens, és ezen belül biztosítja a time-sharing üzemet mind saját parancsainak végrehajtására, mind pedig saját futtatási környezete - pseudo-partíciói - számára.

Igen lényeges tulajdonsága, hogy a DOS/VS többi partíciójának munkáját - tehát az alap multiprogramozott üzemet - funkcionális szempontból egyáltalán nem, teljesítmény szempontból pedig csupán minimális mértékben korlátozza. Hasonlóan kiemelkedő tulajdonsága a DOS/VS partíciókkal való hatékony, kétirányú közlekedési lehetőség.
Ez biztosítja a megfelelő felhasználó - rendszer kapcsolatot olyan feladatok esetén is, amik valamilyen okból a DOS/VS partíciókhoz kötődnek, ETSS-en belül nem jól kezelhetők.

Az Intézet IBM 370/125 rendszerének főbb software paramétereit és a hardware konfigurációt a következő ábrák mutatják.



1. ábra HARDWARE KONFIGURÁCIÓ.



2.ábra AZ ETSS MŰKODÉSI VÁZLATA

Partíció-név	Virtuális tár	Reális tár	A partíció felhasználása
F1	206 K	30 K	POWER
F2	600 K	-	Batch
F3	600 K	100 K	CICS, ETSS, MTCS
F4	600 K	100 K	Batch
BG	294 K	40 K	Batch

A TIME SHARING RENDSZER FELHASZNÁLÓI SZEMPONTBÓL LÉNYEGES JELLEMZŐI

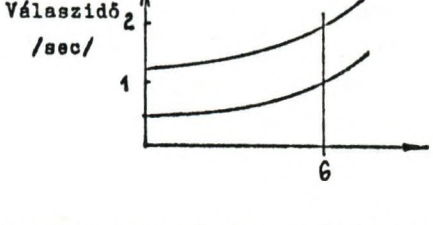
Az ETSS általános célú interaktív on-line programrendszer. A felhasználó kapcsolatban áll a számítógéppel, adatokat, parancsokat adhat meg, melyre azonnali válaszokat, eredményeket kaphat. A számítógép nagy feldolgozási sebességéhez képest lassú terminálok megosztják a számítógép I/O és feldolgozási erőforrásait.

Az ETS számos termináltípust támogat. A programozási nyelvek közül az ASSEMBLER, PL/I, FORTRAN, BASIC, COBOL és RPG használható. A felhasználói adatokat, programokat vihet be terminálról, ezeket megjelenítheti, szerkesztheti, javíthatja. Kérhet programvégrehajtást a time sharing rendszerben, futtathat interaktív programokat, vagy az általa összeállított jobokat, job stream-eket átadhatja batch feldolgozásra. A felhasználónak saját könyvtárak állnak rendelkezésére. Ezeket karbantarthatja, elemeit átírhatja a DOS/VS könyvtárakba és viszont.

Tömör, sokoldalú parancskészlet szolgált az adatok bevitelére, módosítására és egyéb szolgáltatások elérésére. A parancsokból parancslisták, paraméterezhető eljárások állíthatók össze. A felhasználók, a könyvtárak és az adatfile-ok védelme több szinten biztosított.
A rendszerhez utility programok nyújtanak további szolgáltatásokat.

A KIALAKITOTT RENDSZER ÉS ÜZEMELTETÉSI TAPASZTALATAI

A time sharing rendszer három hónapi próbaüzem után került általános intézeti bevezetésre. A próbaüzem alatti teljesítménymérésekből kiderült, hogy az ETSS az alaprendszer terhelését úgy növelte, hogy annak batch feldolgozó képességét nem rontotta számottevően. A terminálokról adott rendszerparancsokra a válaszidő elsősorban az aktív terminálok számától, és kis mértékben a batch particiók leterhelésétől függ. A parancsok jellegétől függően a válaszidő 0,5-1,5 másodperc 1-2 aktív terminál esetén. Erősen romlik a válaszidő 6 terminál egyidejü müködésekor:



A felhasználó számára gyakoribb hozzáférés, gyorsabb fordulások váltak lehetővé. Egyszerübbé vált az adatrögzítés, a javítási tevékenység. Kevesebb terhelés hárul az operátorokra. A gépteremtől távoli felhasználók géphez való hozzáférése alapvetően megváltozott.

A próbaüzem során olyan változások átvezetésére volt szükség, melyek
- az alaprendszerhez és a bővebb felhasználói körhöz való alkalmazkodást segítették
- a rendelkezésre álló terminál park támogatását tették lehetővé
- funkcionális szempontból alkalmazkodtak a helyi igényekhez.

AZ ALAPRENDSZERHEZ ÉS A BŐVEBB FELHASZNÁLÓI KÖRHÖZ VALÓ ALKALMAZKODÁS

Ki kellett alakítani a time sharing rendszer üzemeltetési

környezetét. A meglévő üzemeltetési rendbe az ETSS akkor tud beilleszkedni, ha a batch és az on-line üzem csatlakozási pontjai - indítás, lezárás, batch átadás, stb. - jól definiáltak.

Számos adminisztratív tevékenységet be kellett építeni az indítási eljárásba. Az operátorokat ki kellett képezni az ETSS rendszer kezelésére. Meg kellett teremteni a folyamatos ETSS üzem feltételeit /archiválási rendszer, file-karbantartások/.

Tekintettel a nagyszámú terminál-felhasználóra, különös hangsúlyt kapott az adatbiztonság, adatvédelem kérdése. Egy felhasználó profilját /jogait és lehetőségeit/ úgy határoztuk meg, hogy lehetőleg minden ETSS, ill. DOS/VS szolgáltatást elérjen, de korlátozva legyen más felhasználó adatainak elérésében, főleg módosításában. Korlátozva van a felhasználók pseudo-partición belüli futási ideje és egy-egy könyvtári elem mérete. Definiáltunk olyan könyvtári elemeket, melyeket bárki elérhet. Ilyen sajátosságokkal rendelkeznek az általános célú procedurák és egyéb közös adatok.

Olyan körülményeket kellett teremteni, hogy az on-line és a batch feldolgozások aránya kedvező legyen. Az előbb említett idő és méretkorláton túl is a felhasználók rá vannak kényszerítve munkáik batch feldolgozásra való átadására akkor, ha eredményeikre hardcopy formában van szükség, vagy akkor, ha kevés terminál-idő áll rendelkezésükre a pseudo-particióbeli futás kivárására.

Be kellett vezetni a batch elszámolási rendszerrel összhangban levő gépidő számlázást. Erre főként azért van szükség, mert így jó adatokat nyerhetünk az egy-egy témára fordított gépidőről, sőt - mivel az elszámolási rendszer a felhasználói aktivitást is méri - a témára fordított emberi tevékenységről.

A RENDELKEZÉSRE ÁLLÓ TERMINÁLPARK TÁMOGATÁSA

A meglévő VIDEOTON /VT 340, VDDS/ display terminálok az ETSS rendszerben mint IBM kompatibilis terminálok funkcionálnak. A VT 340 típusú display-ek IBM írógép jellegü terminálként működnek. Mivel a rendszer számára a terminál írógép volt, nem tudta kihasználni a display nyújtotta többletlehetőségeket. Átalakításaink nyomán lehetővé vált a képernyő jobb kihasználása, pl. mód van arra, hogy az egyszer már beírt, vagy megjelenített adatot újra és újra felhasználjuk, mintegy bufferként használva a képernyőt, stb.

Az átalakított VDDS terminál kód- és karakterkészlete, program access és program function kezelőgombjai némi különbséget mutattak az ETSS által várt IBM display terminálhoz képest, ezeket ennek megfelelően módosítani kellett.

HELYI IGÉNYEKHEZ VALÓ FUNKCIONÁLIS ALKALMAZKODÁS

Az ETSS önmagában is alkalmas a programfejlesztés bonyolult, összetett tevékenységének támogatására. Az intézetünkben még meglévő, a programfejlesztés bizonyos lépéseit támogatá eszközeinkről azonban nem kívántunk lemondani. Az ETSS módot nyújt arra, hogy rendszerparancsaiból tevékenységi láncokat, procedurákat lehessen összeállítani. A procedurák rugalmasságát paraméterehetőségük biztosítja. Ilyen procedurákból egy gyüjtemény készült, melyet bármelyik felhasználó elérhet. A procedurák segítségével olyan ismétlődő tevékenységek végrehajtása egyszerüsödik egy utasítás kiadására, mint pl. egy batch-ba átadott job kijelölt particióban történő futtatása, vagy a pseudo-particióban előállított lista printerre íratása, vagy egy könyvtári elem tetszőleges szempontok szerinti rendezése, stb.

Az ETSS HELP funkciója arra szolgál, hogy a gyakorlatlan felhasználó számára egy teljes terminál kapcsolatot bemutatva demonstrálja a rendszer használatát. Mi a HELP funkciónak tanító, oktató, referencia-szolgáltató funkciót adtunk. A felhasználó néhány kulcsszó megadásával általános, vagy specifikus ismereteket szerezhet a rendszerről, ill. egy-egy parancsról. Pl. a /HELP USERS parancs megadásával a felhasználókról, azok profiljáról, idő- és méretkorlátairól szerezhet információkat, és a parancsban adott válasz még különböző utalásokat, további hivatkozásokat, "étlap"-ot is tartalmaz. Mód van valamennyi HELP információ printeren való kiíratására is, ezt gyakorlatilag kézikönyvként lehet használni.

Az ETSS nem biztosította a monitor lehetőségeket, a felhasználó batch feldolgozásra adott munkáiról további információkat nem szerezhetett. A megvalósított DIAL funkció segítségével mód van a sorbaállított, input queue-ban lévő job-okról, vagy az elkészített, ki nem nyomtatott list-queue-ban lévő listákról információt szerezni.

TOVÁBBI FEJLESZTÉSI IRÁNYOK

A rendszer eddigi üzemeltetési tapasztalatai alapján megfogalmaztuk azokat a továbbfejlesztési célkitűzéseket, melyeket rendszerünk második változatában kívánunk megvalósítani. Felhasználói szempontból ezek közül a következők érdemelnek említést:

1. a display tulajdonságok fokozottabb kihasználása;
2. néhány "új alkalmazási terület" számára további specifikus szolgáltatások biztosítása;
3. egységes terminálkezelő rendszerbe való integrálás.

1. Az eredeti ETSS lényegében írógépes terminálokra orientált. A modernebb display-k támogatása csupán csökevényes formában biztosított. Kissé durva általánosítással azt mondhatjuk, hogy input müveleteknél a display képernyőjének csupán 1 sorát kezeli a rendszer és output funkciói sem használnak ki minden lehetőséget.
A rendszer lényeges javításaként mindkét irányban teljes képernyő támogatást kívánunk biztosítani, ami különösen update jellegü müveleteknél jelenthet minőségi változást. Más oldalról ez a javítás megteremti a különböző speciális alkalmazások kényelmes megvalósításának és minimális szakismeretet igénylő kezelésének lehetőségét.

2. Az elsősorban megcélzott alkalmazási területek:
   - A programfejlesztési munka belövési fázisa. Itt a rendszer új komponensekkel való kiegészítését tervezzük.
   - Szövegszerkesztés, különböző dokumentációs munkák számítógépes támogatása. Ezen a területen a rendszer szerkesztési funkcióinak javítását és meglévő szövegszerkesztési és dokumentációs rendszerekkel való hatékony kapcsolat megteremtését irányoztuk elő.
   - Az adatbeviteli funkciót különböző ellenőrzési lehetőségek biztosításával tervezzük javítani.
   - Kisebb volumenü nyilvántartási feladatok, valamint a számítógépes oktatás irányában való kiegészítés lehetőségként szerepelnek terveinkben.

3. Tekintettel a DOS/VS alaprendszer batch jellegére a terminálok kezelését az alaprendszeren kívül kell biztosítani. Az ETSS ennek megfelelően rendelkezik egy saját terminálkezelő alrendszerrel. Mivel azonban más interaktív, ill. on-line - tehát terminálokra épülő -

alkalmazói rendszer is előkészítés alatt áll, szükségessé válik a terminálkezelés egységesítése.

Összefoglalva: úgy gondoljuk, hogy az ETSS üzembehelyezése és vázolt továbbfejlesztése hasznosnak bizonyult, jelentősen megnövelte rendszerünk teljes job áteresztő képességét, és ami talán sokkal fontosabb, tényleges munkaeszközzé tette a számítógépet a fejlesztési gárda számára.

PROLOG ALAPU GYÓGYSZERTERVEZÉSI PROGRAMRENDSZER

Darvas Ferenc*, Futó Iván**, Szeredi János*,
Bendl Judit**, Köves Péter*

Összefoglalás

Az előadásban gyakorlati gyógyszertervezési feladatok megoldására szolgáló programrendszert ismertetünk.

Mivel szerves vegyületek biológiai hatása és kémiai szerkezete között csak kevéssé ismeretes az összefüggés, a gyógyszerek és növényvédőszerek kifejlesztésével foglalkozó vegyészek munkájukban gyakorlati szabályokra és statisztikai jellegü számitásokra támaszkodnak. Az ismertetendő programrendszer a következő két alrendszer révén segiti a gyógyszertervezéssel foglalkozókat:

/1/ Matematikai statisztikai, kémiai szerkezetkezelési és kvantumkémiai számitásokat automatizáló programrendszer.
/2/ Következtetések automatikus levonására szolgáló programok.

A rendszer PROLOG és FORTRAN nyelvü programokat tartalmaz. A rendszer több programja még kisérleti fázisban van, néhány program azonban már több mint két éve müködik és az eredményekről többször beszámoltunk gyógyszerkémiai szakfórumokon.

---

* Számítástechnikai Koordinációs Intézet, Budapest

** NIM IGÜSZI, Budapest

A rendszer célja

A programrendszert számitógépes gyógyszertervezéssel egyébként is foglalkozó szakemberek fejlesztették ki munkájuk hatékonyságának fokozására. A rendszer felhasználói majdnem kizárólag számitástechnikában járatos vegyészek. Ennek következtében a rendszer kiépitésénél nem kellett nagy gondot fordítani a felhasználóval való érintkezés megfelelő kialakitására, mig a hasonló programrendszereknél /MYCIN, egyes szerves szintézistervező programok /1/, /2// ez a feladat komoly erőfeszitést igényelt a program készitői részéről.

A programrendszerrel szemben támasztott legfőbb követelményünk a gyakorlati használhatóság volt. Azt várjuk, hogy a rendszer komoly segitséget nyujt a gyógyszertervezéssel kapcsolatos feladatok automatikus megoldásában. Ez a követelmény magában foglalja azt is, hogy a feladatokat a rendszerrel elfogadható gépidőfelhasználás mellett oldjuk meg és a rendszer programjai könnyen módosithatóak legyenek: a gyógyszertervezés gyorsan fejlődő terület, évente 2-5 új számitási eljárást publikálnak.

A rendszer áttekintése

A programrendszer vázlatát az 1. ábra ismerteti. A nyilak információáramlást jelentenek. A gyógyszerinterakciók előrejelzésére szolgáló program jelenleg még a rendszer izolált

része. Míg a rendszerben szereplő egyéb programok a vegyületek farmakológiai teszteredményeinek előrejelzésére szolgálnak, addig e program célja klinikai szintü hatás prediktálása.

A rendszer ICL 1903/A és ICL 1905 tipusu gépeken müködik /mindkettő 64 K szó memóriával van ellátva/. Az interaktiv programok ICL System 4 és Siemens BS 2000 rendszereken müködnek /az utóbbi 5 Mbyte virtuális memóriával rendelkezik/.

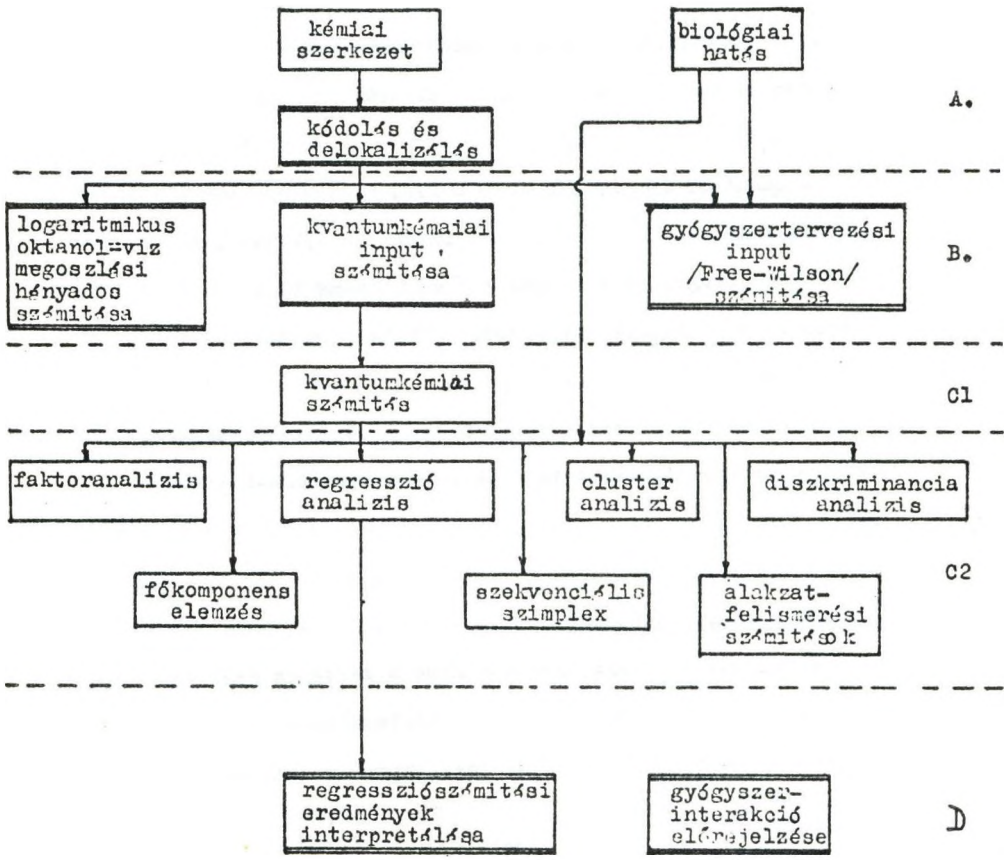
A PROLOG programok max. 40 K szó memóriát igényelnek, az interpreter 14 K memóriát. Az interpreter CDL-ben készült /3/.

Példa a rendszer müködésére

A rendszernek leggyakrabban arra a kérdésre kell választ adni, hogy mi lesz egy adott képletü szerves vegyület hatása egy, a fejlesztők által megválasztott farmakológiai teszten.

A kérdés megválaszolására a szóbanforgó vegyület kémiai képletét kódoljuk és egy PROLOG programmal megkeressük a képletnek megfelelő irányitatlan gráf éleit. Ezután kiválasztjuk az u.n. delokalizálható kémiai kötéseket. A kódolási fázis /1. ábra "A" részlet/ inputja a vegyület kémiai képlete, outputja a vegyület gráf-élek formájában kódolt képlete.

A kódolt képletből kiindulva három további PROLOG programmal a vegyület három fontos jellemzőjét számitjuk ki. /1.ábra, "B" részlet/. A fizikokémiai és kvantumkémiai jellemzők a kémiai szerkezetnek a biológiai hatás szempontjából



1. ábra

A gyógyszertervezési programrendszer logikai vázlata A kettős vonallal határolt téglalapokban lévő programok nyelve PROLOG, a többié FORTRAN.
"A" fázis: a kémiai struktura kódolása és delokalizálása
"B" fázis: a gyógyszertervezési számítások előkészitése
"C" fázis: gyógyszertervezési számítások
"D" fázis: a gyógyszertervezési számítások értelmezése mechanikus következtetéssel. Funkcionálisan ide tartozik a gyógyszerinterakció-előrejelző program.

fontos tulajdonságait adják meg. Az alábbi három paramétert számitjuk ki: a vegyület oktanol-viz megoszlási hányadosának logaritmusát, a vegyület nettó atompopulációját és a Hückel molekulapálya-számitás egyéb eredményeit, végül a Free-Wilson tipusu számitás inputját.

A programrendszer "B" fázisának eredményei szolgálnak a szorosabban vett gyógyszertervezési számitások elvégzésére. /C1, C2 részletek/.

A C1 részlethez tartozó program a kvantumkémiai számitás előkészitését végzi el.

A C2 rendszerrészlethez tartozó programok matematikai statisztikai eljárások segitségével összefüggést állitanak fel a kémiai szerkezet és a biológiai hatás között. A C1-C2 fázisok programjai FORTRAN nyelven készültek.

A hatás-szerkezet összefüggéseket kifejező regressziós egyenletek értelmezését a "D" rendszerrészleten belül egy interaktiv PROLOG program végzi. A program adatbázisa olyan fizikokémiai és biokémiai szabályokat tartalmaz, amelyek alkalmasak arra, hogy a regressziós egyenlet tagjainak fizikai értelmet adjunk. Ez a program számitja ki egyúttal a kiválasztott vegyület várható hatását a farmakológiai teszten.

A szintén "D" fázishoz tartozó interakció_előrejelző program egyidejüleg adott gyógyszerek hatásmodosulására ad becslést és felhivja a figyelmet az esetleges káros szimptomákra.

A rendszer néhány jellegzetessége

A rendszer matematikai statisztikai programokon kivül heurisztikus problémamegoldó és deduktiv következtetéseket automatizáló programokat is tartalmaz. Tudomásunk szerint mesterséges intelligencia-módszereket és statisztikai eljárásokat csak egy kanadai kutatócsoport alkalmazott együttesen /4/.

A rendszeren belül a jól algoritmizálható feladatokat FORTRAN nyelvü programok oldják meg, mig a heurisztikus problémamegoldást PROLOG nyelvü programok biztositják A PROLOG nyelv alkalmasnak bizonyult a kémiai strukturák tárolásával és átalakitásával járó gráfkezelési feladatok megoldására is. A PROLOG programok a nyelv szerkezetéből következően egyszerüek és jól strukturáltak, igy a programokat nagyon könnyü módositani. A PROLOG nagyon alkalmasnak bizonyult a gyógyszertervezéssel kapcsolatos kémiai szabályok természetes nyelvhez közelálló megfogalmazására. A programirási és programmódositási feladatokat nagyban egyszerüsitette az a tény, hogy a PROLOG-ban közös vezérlési mechanizmust használhatunk tételbizonyitásra, fabejárásra visszavezethető feladatok megoldására és adatok visszakeresésére.

Tapasztalatok a rendszer alkalmazásával kapcsolatban

A rendszer egyes programjai 1975. óta müködnek. A rendszer koncepcióját és az elért eredményeket gyógyszertervezési tematikájú kongresszusokon többször ismertettük /5-9/. Néhány beszámoló számitástechnikai fórumokon is elhangzott /10-12/. A gyógyszerinterakció-előrejelző program egyik eredményéről orvosi szimpoziumon számoltunk be. /13/.

Irodalom

/1/ E.H. Shortliffe, R. Davis, S.G. Axline, B.G. Buchanan, C.C. Green, S.N. Cohen: Computer-Based Consultations in Clinical Therapeutics Explanation and Rule Acquisition Capabilities of the MYCIN System.
Computers and Biomedical Research 8 303-320 /1975/ /2/
W:T. Tipke, S.R. Heller, R.J. Fedlmann,-E. Hyde: Computer Manipulation and Representation of Chemical Information.
Wiley, N.Y., 1974.
/3/ P. Szeredi: PROLOG - a Very High Level Language Based on Predicate Logic. Preprints of II. Hung.Comp.Sci.Conf., 1977. Julius, 853-866.old.
/4/ F.A. Miller: Improving Heuristic Regression Analysis. Előadás a 6. Annual Southeastern Regional Meeting of ACM c. konferencián, Chapel Hill, North Carolina, 1967. Jun.15-17.
/5/ Darvas F., Futó I., Szeredi P: Program gyógyszerkölcsönhatások visszakeresésére és uj gyógyszerkölcsönhatások megtalálására mechanikus következtetés segitségével.
"Számitógép az Orvostudományban" c. szimpozium kiadványa, Szeged, 1976. 413-422. /Szerk. Muszka Dániel/.
/6/ Darvas F.: Matematikai módszerek alkalmazása a kémiai szerkezet és a biológiai hatás összefüggésének vizsgálatára. Előadás az MTA közgyülése alkalmából rendezett szimpoziumon, Budapest, 1977. Kémiai Közlemények, megjelenés alatt.
/7/ F. Darvas, I. Futó, P. Szeredi: Some Application of Theroem Prover Based Machine Intelligence in QSAR.
Proceedings of the Symposium on Chemical Structure-Biological

Activity Relationships - Quantitative Approaches, Suhl, NDK. Akadémiai Kiadó, Budapest, megjelenés alatt.

/8/ Darvas F: Biológiai hatások és kémiai szerkezet közötti összefüggések vizsgálata számitógéppel. Magyar Gyógyszeripari Egyesülés, Számitógépes Program - könyvtári Kiadvány, p. 27. /1976/. /Szerk. Horváth Gyula/.

/9/ F. Darvas, I. Futó P. Szeredi: Combination of Theorem Prover-Based Machine Intelligence and QSAR-Methods. Előadás, Gordon Research Conference on QSAR, Plymouth, New Hampshire, USA, 1977.

/10/ Futó I., Darvas F, Szeredi P, Márkusz Zs: Probléma-megoldás és adatbázis-kezelés nagyon magas szintü nyelveken. ESZR-Szeminárium, Moszkva, 1976.

/11/ F. Darvas,I. Futó, P. Szeredi: Logic Based Program System for Predicting Drug Interactions. International Journal of Biomedical Computing, megjelenés alatt.

/12/ I. Futó, F. Darvas, E. Cholnoky: Proctical Applications of an AI Language. Prerpints of the II. Hung.Comp.Sci.Conf., 1977. Julius, 388-399. oldal.

/13/ F. Darvas, I. Futó. P. Szeredi: Expected Interactions of Spironolactone: Predictions by Computer. Proceedings of the Symposium on the Pathomechanism and Clinical Treatment of Hyperaldosteronism, Budapest, 1977. Megjelenés alatt.

A MIKROPROGRAMOZÁS AUTOMATIZÁLÁSÁRÓL

Dávid Gábor, Keresztély Sándor, Sárközy András
MTA Számitástechnikai és Automatizálási Kutató Intézet, Budapest

BEVEZETÉS

Számitástechnikai eszközök programozásánál a munka jelentős része: az algoritmizálás alsóbb szintjei és a kódolás mechanikus, nem kreativ tevékenység. Az ismertetésre kerülő kutatások célja ezen munka automatizálása részint abból a célból, hogy a természetszerüleg elkövetett hibáktól megszabaduljunk és hogy a programozási munka mennyiségét csökkentsük. Jelen dolgozatban a mikroprogramozás automatizálásával foglalkozunk.

A MIKROPROGRAMOZÁS ÉS IGÉNYEI

Mikroprogramozott eszközök azok, amelyekben a felhasználó utasitásai nincsenek közvetlenül - hardware-ben - implementálva, hanem egy egyszerübb berendezés állapotátmeneteinek a sorozataként realizálódnak. Az egyszerübb hardware berendezést mikrogépnek, lehetséges állapotátmeneteit mikroutasitásnak, ezeket az utasitásokat leiró sorozatait pedig mikroprogramnak nevezzük. A mikroprogramozási tevékenység napjainkban egyre fontosabb lesz és szerepe, mennyisége jelentősen növekszik.

Ennek okai: a felhasználói utasitáskészlet szinvonalának folyamatos emelkedése, az alkatrésztechnológia gyors fejlődése /mikroprocesszorok/ által lehetővé vált a célhardware épités, és a felhasználó által változtatható tartalmu mikroprogramtárak megjelenése. Ez a helyzet a mikroprogramozási nyelvekkel kapcsolatban ujszerü követelményeket támaszt, olyanokat, amelyek az assembly nyelvekben fel sem merülhetnek. Ezek a követelmények mind a programok minőségében, mind a programkészités technológiájában jelentkeznek.

A mikroprogramoknak bizonyitottan helyesnek kell lenniük, vagyis specifikációjuknak bármely kezdeti adatrendszerre ki kell terjednie. A technológiai fejlődés következtében a komponensek részleges, vagy teljes lecserélésének, uj elembázison való realizálásának hatását minimalizálni kell a már meglévő mikroprogramokra, vagy az áttérést maximálisan /számitógéppel/ kell támogatni. Másik oldalról: egy felhasználói probléma megoldása során realizált mikroprogramokat célszerü lenne felhasználni az uj elembázison is és igy "társadalmasitani" a software-sek munkáját.

Amig a programkészitési technológia kérdésére egy ideig - és lehet, hogy hosszu ideig - választ adna valamilyen magasszintü algoritmikus nyelv, ez a probléma elodázását jelentené. Perspektivikusabbnak tartottuk a software-készités technológiáját és a mikroprogramok minőségellenőrzési-módszereit

együttesen megoldó automatikus programozás eszközeit alkalmazni a mikroprogramozás igényeinek megfelelően.

AZ AUTOMATIKUS PROGRAMOZÁS

Az automatikus programozás olyan programozási eszköz, amelyben egy, vagy több felhasználó közli ismereteit a programot végrehajtó géprõl, kisegitõ információkkal együtt - ezt az ismeretanyagot nevezzük esetünkben a számitógép leirásának -, majd minden felhasználó a kért programot ugy specifikálja, hogy a számitógép milyen állapotából a program végrehajtása után milyen állapotba kerül: ha S_I és S_O jelöli a végrehajtás elõtti, ill. utáni állapotot /az S_I , S_O állapotokat input/cutput - I/O - relációnak nevezzük/, akkor a P program a lehetséges gépállapotok közötti leképzés:

$$P : S_I \rightarrow S_O$$

Az automatikus programozást egy olyan - matematikai logikán alapuló - eszköz valósitja meg, amely megadja a P programot, mint végrehajtó utasitások egy $p_1 p_2 \ldots p_n$ sorozatát, amelynek $S^o = S_I$, $S^n = S_O$ jelöléssel az S^j közbülsõ állapotokra $j = 1, \ldots, n-1$

$$p_j : S^{j-1} \rightarrow S^j$$

transzformációk igazak és együttesen

$$p_1 p_2 \ldots p_n : S_I \rightarrow S_O$$

azaz a kivánt p programot adják. A $p_1 p_2 \ldots p_n$ utasitássorozatot voltaképpen annak az állitásnak az automatikus programozó eszközünk által végzett konstruktiv bizonyitása során kapjuk, hogy $S_i \rightarrow S_o$ állapottranszformáció /igy a p program/ igaz.

A programozás automatizálása során igy a következőkre van szükségünk:

- a gép leirására, amely magában foglalja a gép strukturális komponenseinek és azok hierarchiájának leirását;
- az utasitások hatásának leirására az érintett komponenseken;
- a felhasználó által közölt ismeretanyagra;
- a kivánt program input/output relációira.

Az adott követelményeket a fentiek szerint csak a matematikai logikán alapuló leirás elégitheti ki, s olyan logikát kellett választani, amelyben mechanikus tételbizonyitási technikát lehet realizálni.

A MIKROPROGRAMOZÁS SPECIÁLIS IGÉNYEI AUTOMATIKUS PROGRAMGENERÁLÁSNÁL

A mikroprogramozandó eszköz jellege a következő követelményeket támasztja egy automatikus generáló rendszerrel szemben:

1. Különböző tipusu objektumokra /pl. bit, byte, szó/ kell müködnie.
2. Kezelnie kell strukturák egyes elemein végzett operációk eredményét a strukturák szintjén is /pl. egy szó egyik byte-jának megváltoztatása után is szóként értelmezhetőnek maradjon/.
3. A kezelendő objektumok igen nagy számára való tekintettel mind a leirásnál, mind a müködés közben csak az érintett objektumok megadására legyen szükség.

A mikroutasitások igen nagy száma miatt, minthogy a mechanikus tételbizonyitási eljárások hatékonyságát a végrehajtó utasitásokat leiró állitások száma döntően befolyásolja, ezért ezek számát minimálisra kell csökkenteni. Mikroprogramozás automatizálásánál erre a következő eszközök állnak rendelkezésre:

1. Horizontális mikroprogramozásu berendezés esetében a mikroutasitások párhuzamosan végrehajtott mikroparancsokra bonthatók, amelyeknek a teljes száma sokkal kisebb a lehetséges mikroutasitások számánál. A programgenerálás során a mikroparancsokat kell utasitásként felhasználni és az eredményként kapott mikroparancssorozat elemeit kell egy postprocesszorral összevonni mikroutasitokká.

2. Azok a mikroutasitások, vagy mikroparancsok, amelyek egy tömb /pl. regisztertömb/ különböző elemein végeznek azonos

operációt, egyetlen állitással irhatók le, ha a tömb elemeinek kiválasztását parametrizáljuk. Ez az eljárás 4-8-szoros redukciót eredményez.

3. A rendszer a müveleteket a hierarchikusan leirt objektumok lehető legmagasabb szintjén értelmezze.

A STRUKTURA LOGIKA

A speciális követelményekre való tekintettel a klasszikus elsőrendü logikán alapuló ujelvü nyelvek helyett erre a célra jobban megfelelő Struktura Logikát vezettük be. A Struktura Logika /a továbbiakban SL/ alapja a többfajtáju logika /many-sorted logic/ és kibővitve a strukturák leirásában hasznos szelektorok fogalmával. /Részletesebb leirása megtalálható [1]-ben./

Az SL-ben egy "program" a következő részekből áll:

- deklarációs rész
- struktura-leiró rész
- mikroparancsokat leiró rész
- a generálandó program I/O relációi.

Egy adott gépre csak az utolsó rész az, amit a felhasználónak le kell irnia, az első három a "rendszerprogramozók" feladata. Az első három részhez a programozó kiegészitéseket füzhet, ha

speciális adatstrukturákat, vagy transzformációkat /pl. a rendelkezésre szubrutinokat/ akar használni.

A deklarációs és struktura-leiró rész mondja meg, hogy milyen tipusu strukturákon irjuk le a gépet. Alaptipusok: bit/n/ /n hosszuságu bit/, selector, instruction, valamint ha $a_1, a_2, \ldots, a_k$ már definiált tipusok, az $s_i, \ldots, s_n$ szelektorok, akkor egy uj t tipust a

$$t \langle s_1 : a_1, \ldots, s_k : a_k \rangle$$

állitással definiálhatunk. Ugyancsak itt mondjuk meg, hogy minden szimbólum milyen tipusu a

reference /tipus/ szimbolum-lista;

utasitásokkal. Ugyancsak itt kell leirni a függvény-szimbólumok tulajdonságait is.

A mikroparancsokat leiró rész az

$$S' \longrightarrow S''$$

tipusu állitások egy halmaza. Egy ilyen egy mikroparancsot definiál:

S' leirja azt az állapotot, amelyben a mikroparancs végrehajtható,

S'' azt, amelyet a mikroparancs végrehajtása után kapjuk.

Az S állapotleirók általános formája

$$S \langle \{ s_j : k_j \} \rangle$$

/$\{ \ldots \}$ halmazt jelöl/, ahol S valamilyen t tipusu struktura, az s_j-k valamilyen szelektorok és k_j az s_j által kiválasztott részstruktura tartalma /amelynek tipusa a t tipusból kiválasztott résztipus/. Az s_j-k mind különbözőek. Az $S\langle\rangle$ azt jelenti, hogy az $\{ s_j : k_j \}$ halmaz üres, azaz a gép az S által leirt, tetszőleges állapotban van.

Ezzel az egyszerü nyelvvel a fenti követelményeket a következőképpen tudjuk kielégiteni:

1./ Az S struktura definiálásakor /a deklarációs részben/ egyszer le kell irni minden összetevőjét, ami általában elég bonyolult a nagyszámu komponens miatt. Azonban egy-egy mikroparancs ezekből az összetevőkből csak kevésre vonatkozik, igy általában 1-2 részstruktura megváltoztatását jelenti. Ezek leirása nagyon egyszerü, ami általában 5-6-szoros redukciót jelent.

2./ A hasonló mikroparancsok egy közös állitással irhatók le. A bevezetett szelektor-változó fogalom segitségével azok a mikroparancsok irhatók le egy állitással, amelyek ugyanazon tipusu komponenseken operálnak ugyanugy, csak a konkrét komponens azonositójában különböznek és ezt az azonositást elvégzi a szelektor-változó, amely ezeken az azonos komponenseken van értelmezve. Ez az eszköz további 4-5-szörös redukciót eredményez.

3./ A homogén strukturákon két szinten irhatjuk le a gépet: a strukturán magán és a megfelelő komponenseken. SL-ben ezek közül csak az egyikre van szükség, mivel minden állitás, amely valamely strukturára vonatkozik, egyuttal minden részstruktúrájára is vonatkozik és forditva. A két szint közötti átmenetelt a SL automatikusan biztositja, igy célszerü mindig a legáltalánosabb strukturát használni valamely mikroparancs leirásánál.

4./ A tipus egyeztetés a SL-nek sajátja, igy ezt a követelményt automatikusan teljesiti. A program generálása során a közbülső $S^{j-1} \longrightarrow S^{j}$ átmenetek megkeresésénél csak azokat veszi figyelembe, ahol a tipusokban nincs ellentmondás.

A generálandó program I/O relációit kell csak a felhasználónak megadnia. Ez /általában/ egyetlen

$$S_i \longrightarrow S_o$$

állitás leirása. A rendszer a tételbizonyitási technikával a gépet előzőleg leiró tényeket felhasználva bebizonyitja a felhasználó állitását és a bizonyitás egyuttal a kivánt programot adja, de csak mikroparancs-szekvencia formájában. Ez a program természetesen bizonyitottan helyes.

Ahhoz, hogy a kivánt mikroprogramot, azaz mikroutasitás-szekvenciát megkapjuk, vissza kell transzformálni a vertikális

formát horizontális és vertikális formába. Ezt a transzformációt szintén számitógéppel akarjuk végezni, igy mechanizálhatónak, algoritmizálhatónak kell leniie. Feltételezzük, hogy van egy listánk, amely minden mikroparancsra megmondja, hogy mely mezőkön szerepelhet. Egy generált mikroparancs-szekvenciához a rendszer generálta a megfelelő állapotu átmeneteket is:

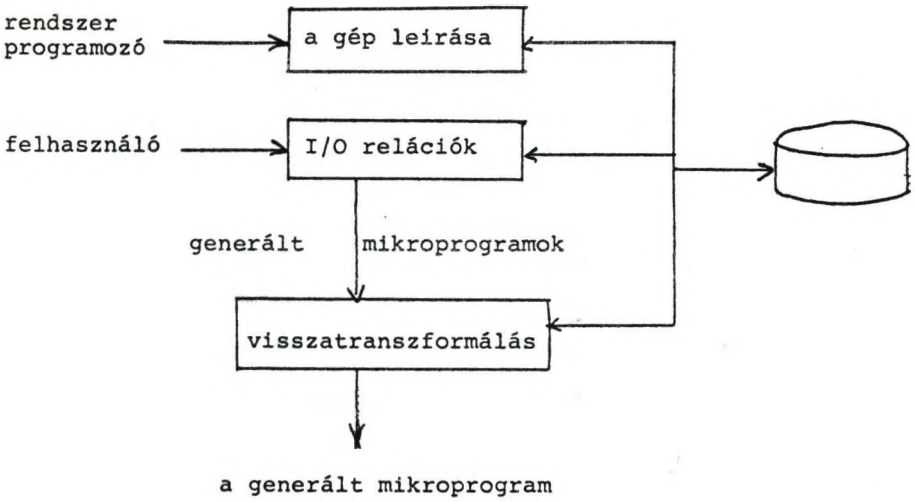
$$
\begin{array}{lcl}
 & s^0 = s_i & \\
p_1 : & \downarrow & m_{i1} \\
 & s^1 & \\
p_2 : & \downarrow & m_{i2} \\
 & s^2 & \\
 & \vdots & \\
 & s^{n-1} & \\
p_n & \downarrow & m_{in} \\
 & s^n = s_o &
\end{array}
$$

mikroparancsok átmenetek mezők kódjai

Valamely p_i és p_j párhuzamosan hajtható végre, ha egyrészt olyan mezőkön szerepelnek, amelyek párhuzamosan interpretálhatóak, másrészt a megfelelő s^{i-1} és s^{j-1}, valamint s^i és az s^j állapotokra a p_i és p_j együttesen mint egy utasitás:

$$(p_i, p_j) : s^{i-1} \wedge s^{j-1} \rightarrow s^i \wedge s^j$$

/ahol $\wedge$ a logikai és/ alakban leirhatóak. Végrehajtva az ilyen utasitások párhuzamositását, a megfelelő mezők kiosztásával kapjuk meg a kivánt mikroprogramot.



A rendszert az MTA CDC 3300-as gépén realizáljuk. A rendszer komponensei közötti kapcsolatot egy könyvtárkezelő rendszer tartja.

IRODALOM

1 Dávid G., Keresztély S., Sárközy A.: Microprogram Synthesis by Theorem Proving. II. Magyar Számitástudományi Konferencia, 1977. 1 kötet, pp. 291-310

EGY PORTÁBILIS FORDITÓPROGRAM EGYSZERŰ PRECEDENCIA NYELVEKHEZ

Dettrich Árpád
Számitógépalkalmazási Kutató Intézet

Több mint tiz esztendeje annak, hogy először megfogalmaztuk a portábilis forditóprogramok készitésének gondolatát, amit először a LAFO nevü makróprocessoron /De68/, majd az UTRA elnevezésü általános forditóprogramon mutattunk be /Dö68/. Hála a számitógép tipusok sokaságának a téma még ma is időszerü, ezért az azóta szerzett tapasztalatok alapján megmutatom a forditóprogram irás automatizálásának egy lehetséges utját, továbbá egy elemző algoritmust, amelyet több éve sikeresen alkalmazunk /először:De71/.

Tapasztalatok az UTRA-val

A megvalósitott rendszer alapgondolata az volt, hogy a számitógépes feldolgozás lényegében mindig jelsorozatok transzformációja. Ezt a transzformációs tevékenységet bontottuk két szintre, az egyik szint kimondottan gépfüggő, mig a másik géptől függetlenül irható le. A transzformációban résztvevő jeleknek három tipusát különböztettük meg. A C-tipusú jelek a perifériális készülékeken ábrázolt irásjelek belső kódjai. A V-tipusú jelek a gépfüggő tevékenységeket reprezentálják - hardware tevékenység - ami egy-egy gépi nyelven megirt szubrutin. A P-tipusú jelek pedig a géptől független tevékenységeket irják le - software

tevékenység - amelyek egy-egy jelsorozatot reprezentálnak /a jelsorozatban C,P,V tipusú jelek vannak/.
A rendszer lelke egy jelgenerátor /JG/, amely az - általában - 8 bites jeleket az első két bit alapján különbözteti meg. Ha az első bit 0 akkor a jel C-tipusú, amit a JG átad a "főkapcsoló"-n keresztül /FKP/ egy speciális gépfüggő rutinnak, pl. egy lexikális elemzőnek vagy egy szintaktikus elemzőnek. Ha a jel első bitje 1 akkor a második bit specifikálja, hogy P vagy V-tipusú-e. A többi /6/ biten a megfelelő tevékenység sorszáma adható meg /0-63/. Ha a jel V-tipusú, akkor a JG a sorszámmal egy "ugrótáblán" keresztül vezérel a megfelelő rutinra. Ha a jel P-tipusú, akkor a JG a sorszámmal egy jelsorozat pointerét veszi elő egy táblából, és elkezdi az új sorozat generálását.
Az éppen generált sorozat pointerét vermeli, hogy ha az új sorozat elfogyott visszatérhessen folytatni a korábbi jelsorozatot.
A rendszer előnye, hogy rendkivül gyorsan ereszti át a jeleket, és igen kicsiny a gépfüggő magja, ami nagyfokú portabilitást biztosit. A rendszer hátránya, hogy a jelsorozatok felirása P24,V13 stb. formában történik, ami csak a beavatottak számára kezelhető, és egyáltalán nem olvasmányos.
A következőkben bemutatok egy módszert, amely a formális leirást lehetővé teszi.

Az UTRA formális leirása makró-nyelven

A leirásban az Rlo makróassemblerére hivatkozom /Asz74/.

Az olvashatóság kedvéért a névhosszuságra vonatkozó korlátokat nem tartom be, és magyar irásjeleket is használok.

A P és V tipusú jelekhez tartozó táblákat az alábbi szöveggel lehet definiálni:

```
DEFINICIO1 TÁBLÁJA TIPUS=HARDWARE,KEZDET,OLVAS,
     LISTÁZ,VÉGE,KAPCSOLÓ,LERAK,....
DEFINICIO2 TÁBLÁJA TIPUS=SOFTWARE,EGYSOR,KÖVSOR,
     HIBA,...
```

Ebből a leirásból a következő makrókkal állitjuk össze a táblákat:

```
              MAC
              TÁBLÁJA      !SYSLST,!TIPUS=,
              !TIPUS
¢N            SETA    1
.ISM          AIF   ¢N GT ¢NPAR.FOLYT
              DATA   !SYSLST (¢N):
¢N            SETA   1+¢N
              AGO    .ISM
.FOLYT
¢NA           SETA   1
.IS MÉT       AIF  ¢N GT ¢NPAR.VÉGE
!SYSLST(¢N)   EQU  ¢N+¢TIP - 1
¢N            SETA  1+¢N
              AGO   .ISMÉT
.VÉGE         MEND
```

```
                MAC
                SOFTWARE
&TIP            SETA   128   * 10000000 A P-TIPUSU JEL
                MEND
                MAC
                HARDWARE
&TIP            SETA   192   * 11000000 A V-TIPUSU JEL
                MEND
```

A megoldás önmagáért beszél, csak arra hivom fel a figyelmet, hogy a leirásnak ez a módja minden olyan assembly nyelven lehetséges, amelyben van feltételes forditás és megengedett, hogy az aktuális paraméterlista változó hosszú legyen.

A kapott DEFINICIO1 és DEFINICIO2 nevü táblázatok cimketáblák:

```
DEFINICIO1      EQU  &
                DATA   KEZDET:
                DATA   OLVAS:
                  .
                  .
                  .
 DEFINICIO2     EQU &
                DATA   EGYSOR:
                DATA   KÖVSOR:
                  .
                  .
                  .
```

Ezenkivül az eredeti nevekhez egy-egy 8 bites információt rendeltünk hozzá az EQU direktiva segitségével, amely megfelel rendre a V0, V1,V2,... és P0,P1,P2,... jeleknek.

Ezek után a fordító egyes tevékenységeit az alábbi formában irhatjuk:

```
KÖVSOR:    SFWTEV KEZDET,OLVAS,EGYSOR,KAPCSOLO,SCANNER,VÉGE
```

amelyből a következő makró rakja le a megfelelő jelsorozatot:

```
                MAC
                SFWTEV   !SYSLST
&N              SETA  1
.ISM            AIF   &N GT &NPAR .VÉGE
                DATA,1   !SYSLST (&N)
&N              SETA  1+&N
                AGO   .ISM
.VÉGE           MEND
```

A jelsorozat kezdőcime a KÖVSOR: cimke, ami valóban benne van a DEFINICIO2 táblában és a P1 jelet a jól érthető KÖVSOR szó helyettesiti.

A hardware tevékenységek leirása assembly nyelven történik ilyen bevezetéssel:

```
KEZDET:   HWTEV
```

amelyből a következő üres makró segitségével

```
KEZDET:         EQU  $
```

sor lesz:

```
                MAC
                HWTEV
                MEND
```

és ezután lehet irni a megfelelő programot, amely a DEFINICIO1 táblán keresztül érhető el a VO-nak megfelelő KEZDET szó segitségével.

Elemző egyszerü precedencia nyelvekhez

Legyen G = {N,T,P,S} CF nyelvtan. A legtöbb alkalmazási területen elegendő, ennek alosztálya az egyszerü precedencia nyelvtanok ismerete. Az Aho-Ullmann könyv terminológiáját alkalmazva /AHO72/ az egyszerü precedencia nyelvtan olyan CF nyelvtan, amelyben a P elemei olyanok, hogy minden jobboldalhoz pontosan egy baloldal tartozik, továbbá E = N∪T bármely két eleme között fennáll a $\lessdot, \doteq, \gtrdot$ relációk valamelyike. Az ilyen nyelvtanokra definiált "shift--reduce" algoritmus egy olyan bottom up elemző, amely az input jeleket vermeli és csak a verem tetején lévő jel és az input következő jele között figyeli a relációkat. Az input jeleket addig "shifteli" amig $\lessdot, \doteq$ reláció áll fenn és az első $\gtrdot$ relációnál visszakeresi a veremből a mondatforma nyelét, amelyet az első $\lessdot$ reláció határol. Ezután következik a megfelelő baloldal helyettesitése, amit egy rendszerint szekvenciális keresési algoritmus előz meg. A Dömölki-algoritmus módszert ad a párhuzamos keresésre, amely a sokkal általánosabb determinisztikus nyelvtanok osztályára alkalmazható.
A precedencia nyelvtanok esetében azonban nincsen szükség a "nyélkeresés"-re, mert két szomszédos jel egyértelmüen definiálja az elemzés állapotát, és ha felismer egy nyelet, akkor a szükséges - szintaktikai és szemantikai - tevékenységet. Ezért a már jelzett /De71/ munka után is többször alkalmaz-

tam az alábbiakban leirt algoritmust. Az algoritmust egy automatával adjuk meg. Legyen a nyelvtan:

<értad> ::=<vált><egyenlő><kif><végjel>

<vált> ::=<betü>|<vált><betü>|<vált><jegy>

<betü> ::= A|B|C|...|Z

<jegy> ::= 0|1|2|3|...|9

<egyenlő>:: = =

<kif> ::=<tag>|<tag><addop><kif>

<tag> ::=<tény>|<tény><multop><tag>

<tény> ::=<vált>|<nyitó><kif><csukó>

<nyitó> ::= (

<csukó> ::=)

<végjel>::= .

Az automata müködését leiró átmeneti mátrixot /AM/ ugy adjuk meg, hogy a sorait a következő kategóriákhoz rendeljük: <betü>, <jegy>, <addop>, <multop>, <nyitó>, <csukó>, <szóköz>, <végjel>, <egyenlő>, <egyéb>.

A beolvasott kódjellel egy kategória-táblából kiválasztjuk, melyik kategóriába tartozik, pl. i-edik, akkor ha az automatánk a j-edik állapotban van akkor AM(i,j) mutatja meg a soros tevékenységet.

A szintaktikus kategóriának megfelelően az automata állapotait igy adhatjuk meg: baloldal kezdete, a baloldali változó /ez tulajdonképpen lexikális elemzés/, operandus, operátor, stb. A tapasztalat szerint minden ilyen állapothoz legtöbbször 2-3, néha 6-8 tevékenység tartozik, és legtöbb elemzést 15-2o tevékenységgel le lehet irni. /Különböző állapotok-

ban is előfordulnak ugyanazok a tevékenységek./
Ennek megfelelően az AM egy olyan bitmatrix, amelyben ha az i-edik állapotban a lehetséges tevékenységek száma $2^{k-1} \leq n < 2^k$, akkor a megfelelő oszlop k-bites. Minden állapothoz tartozik egy tábla, amelyben a lehetséges - software - tevékenységek pointerei vannak, amelyek az AM (i,j) értékkel kiválaszthatók.
A rendszer erőforrásai:
AM az átmeneti matrix
PT a pointer tábla
AT az állapot tábla, amelyben a következő információk vannak: SHIFT, léptető konstans, amellyel az AM$(i,*)$ vektor j-edik eleme leléptethető az akkumulátor aljára, MASK amellyel leválasztható az utolsó 2-3 bit, és a BASIS amely a PT megfelelő pontjára mutat.
KAT a kategória tábla, amellyel a jeltranszformációt végezzük.
Ezekre az erőforrásokra támaszkodva az elemző - géptől függően - 6-lo gépi-utasitással megvalósitható. Az Rlo-en a megfelelő FKP rutin kevesebb mint 2oo szó helyet foglal el.

Az elemző automatikus generálása

Az elemző elsősorban az UTRA rutinjaira támaszkodik, amelynek formális leirásáról korábban szóltunk. Problémát az új erőforrások automatikus generálása jelent, mert az AM előállitása a megfelelő fogalmak segitségével makróprocessor segitségével nem

lehetséges. Ezért definiáltunk egy egyszerü nyelvet, amelyen könnyen leirhatjuk az egyes állapotokhoz tartozó tevékenységeket és PL/I-ben irtunk egy egyszerü forditót, amelynek outputja a fenti erőforrások assembly nyelvü leirása /amely gépfüggően könnyen változtatható/.

Ezen a nyelven a következő módon kell a forditót megfogalmazni:

```
"KATEGORIÁK" BETÜ:ABCD ...;JEGY:0123...; ...
"ÁLLAPOTOK" BALOLD KEZDET, NÉV BALOLDALON,OPERANDUS,...
"ÁLLAPOT" BALOLD KEZDET;
BETÜ: POINTERT LERAK,UJÁLLAPOT,NÉV BALOLDALON,VÉGE
JEGY: HIBA,"NÉV SZÁMJEGGYEL KEZDŐDIK",VÉGE
:
```

A forditóprogram generálásakor felmerülő problémák

1. A bemutatott módszer a P és V-táblák méretét 64-re korlátozza. Ezen úgy segithetünk, hogy un. csapdás jeleket alkalmazunk, amelyek a következő jel beolvasásával választhatnak más táblából pointereket. A másik megoldás, hogy a P és V-tipusú jeleket 16 bitre képezzük le, ami szinte korlátlan táblaméretet biztosit.

2. Az AM akkor müködik optimálisan, ha egy sor hossza nem nagyobb mint az egy logikai müvelettel lekezelhető bitek száma. Ha ez nem teljesül, akkor az állapotok halmazát megfelelően particionálni kell. Ilyenkor az új állapot beállitásánál lehet, hogy teljesen új táblázatokat kell beállitani.

3. Mivel az elemző generálása PL/I fordítóval rendelkező nagygépen történik, a kisgépes megvalósításnál komoly probléma a forrásszöveg összeállitása. Az R1o gépnél erre a célra az IDOS rendszert használjuk.

REFERENCIA

/De68/ Dettrich Árpád: Maschienenunabhängiger Compiler für eine Listprocessing-Sprache, die zur Ausarbeitung von Compilern Geeignet ist /Wissenscheftliche Zeitschrift der Technischen Universität Dresden 5/1968/

/Dö68/ Dömölki Bálint: A Universal Compiler System Based on Production Rules /BIT 7/1968/

/De71/ Dettrich Árpád, Bakos Tamás: External Specification of Extended FASP and the Linkage Loader for FACOM R /INFELOR doku.1346/1971/

/Asz74/ Aszalós János: MASS külső specifikáció /INFELOR doku.1222/1974/

/Aho72/ A.V.Aho,J.D.Ullmann: The Theory of Parsing, Translation and Compiling /Prentice Hall 1972/

Budapest, 1978. május 13.

EGY MAGASSZINTŰ RENDSZERPROGRAMOZÓI NYELV KÓDGENERÁTORA AZ INTEL 8080-RA

Déri Gábor
MTA SZTAKI

1. Bevezetés

Egy magasszintű, rendszerprogramozás számára készült nyelv /1./ kódgenerátorát ismertetjük. A kódgenerátor, az intézetünkben fejlesztett GD8Ø grafikus display család processzorait alkotó INTEL 8Ø8Ø mikroprocesszorhoz készült. (A nyelv compilerét és TPA 7Ø kódgenerátorát Gerhardt Géza irta. Ezuton mondok köszönetet az INTEL kódgenerátor irásához nyujtott segitségéért.) A GD8Ø software egy része ezen a rendszerpgoramozói nyelven iródott, többek között egy grafikus BASIC interpreter /3/.

Előadásunkban először a nyelv és a kódgenerátor bemeneteként szolgáló "intermediate kód" jellemzőit foglaljuk össze.
Ezután a kódgenerátor felépitését, müködését, a generált kód jellemzőit és a felhasználás tapasztalatait ismertetjük.

2. Néhány szó a nyelvről

A GEZA rendszerprogramozói nyelven irt programok 16 bit szóhosszuságu gépet feltételeznek. Egy program egy vagy több szekcióból, a szekciók globális és eljárás /proc/ deklarációkból állnak.
A proc deklaráció proc fejjel kezdődik és END-el ér véget.
A proc teste végrehajtó utasitásokból áll lokális változói csak a végrehajtás idején élnek.
A procokon kivül globális változók deklarálhatók. Ezek a szekció végéig élnek.

A nyelv alaptipusain kivül /CHAR, INT, REAL, REF,.../, a felhasználó strukturákat, uniokat, flag tipusu változókat, stb. deklarálhat.
A nyelv a pointer tipusu változókat alaptipusként kezeli. Lehetőséget ad strukturákra mutató pointerek deklarálására is. /REFINT I; REF STRUCT ST;/

A nyelv compiler-e két részből áll. Az első rész végzi a compilálást, ennek kimenete a gépfüggetlen un. intermediate kód /2/. A második rész a kódgenerátor, amely az intermediate kód alapján az adott gépre assembly vagy bináris kódot állit elő.

A nyelv a szokásos értelemben "pórtábilis". A compiler "önmagában" is meg van irva; egy uj gépre elég például a kódgenerátort megirni és az intermediate kódu compiler átvitelét biztositani valamilyen adathordozón.

3. Az intermediate kód

Az intermediate kód a compiler gépfüggetlen kimenete, egy absztrakt stack-es gép utasitás készletének tekinthető. Az intermediate gép jellemzői:

- aritmetikai stack
- átmeneti /temporary/ tároló stack
- memória terület bázisregiszterekkel.

Valamennyi aritmetikai művelet az aritmetikai stack legfelső elemei között hajtódik végre. A temporary stack-be mentődik sorozatos összehasonlitások esetén az összehasonlitandó érték. /Az aritmetikai stack legfelső eleme/ Az intermediate gép memóriája memória osztályokra van osztva /global, external, static, local, parameter, result/. Valamennyi memória osztályhoz egy bázisregiszter tartozik, amely az adott terület kezdőcimét tartalmazza. A memóriára a memória osztály megadásával és egy 16 bit-es displacement segitségével lehet hivatkozni.

Az intermediate kód három fajta rekordból épül fel:

- a lexikális analizis
- a deklarációkat dekódoló rész
- az utasitásokat dekódoló rész

által előállitott rekordokból.

A lexikális rekordok a szimbólumok, forrásnyelvi sorok stb. ASCII kódját tartalmazzák. Általában csak információs célokat szolgálnak.

A deklarációs rekordok adják meg

- a változó hivatkozásokat
- konstansokat, kezdőértékeket
- proc start adminisztrációkat.

Az utasitás rekordok az utasitások argumentumait és operátorokat tartalmaznak.
A compiler az intermediate kódot két külön file-on állitja elő.
Az egyiken a deklarációs rekordok, a másikon az utasitás rekordok vannak.
A lexikális rekordok mindkét file-on megtalálhatók az adott file rekordjaival keverve.

4. A kódgenerátor

A kódgenerátor az intermediate kódot alakitja át egy valóságos gép assembly vagy gépi kódjává.

Az intermediate kód két külön file-jának megfelelően a kódgenerálás folyamata időben két részre osztható. Először a deklarációs rekordok kerülnek feldolgozásra, majd az utasitás rekordok alapján kigenerálódik a végrehajtható kód.

Deklarációk

A deklarációs passz az adattipusoknak megfelelően kigenerálja a helyfoglaló és adatlerakó utasitásokat. A CHAR tipusu változóknak egy byte, az INT és REF tipusoknak kettő, a REAL-eknek négy byte hosszu hely foglalódik. A LONGINT és LONGREAL tipusok a kódgenerátorban jelenleg nincsenek implementálva.
Adatokként a kezdőértékek, konstansok, stringek kerülnek lerakásra.

Az intermediate kód ismertetésénél, már emlitettük az absztrakt gép memória osztályait. A kódgenerátor a GLOBAL, EXTERNAL, STATIC memória osztályokat egy memória osztályként kezeli azonos bázisregiszterrel.
A LOCAL memória osztályhoz tartozó változóknak a procok elején, a gép hardware stack-jén foglalódik hely. Ez biztositja az eljárások rekurziv hivhatóságát.

A helyfoglalás és adatlerakás után EQ lista segitségével rendeljük hozzá a program szimbolumaihoz a megfelelő memória cimeket.

Mivel a kódgenerátor assembly kódot állit elő, a memória területek helye assemblálaskor dől el véglegesen.

Müveletek végrehajtása

Mint tudjuk az INTEL 8Ø8Ø mikroprocesszornak egy akkumulátora és hat regisztere van, és byte-os müveleteket tud végezni /4/.
Ezért az integer és real müveleteket run-time rutinokkal végeztetjük el.

A regisztereket a következő módon osztottuk fel:

A	- karakter regiszter
B-C	- integer regiszter /source/
D-E	- integer regiszter /destination/
H-L /M/	- cimregiszter
B-C-D-E	- real regiszter

A karakteres müveletek elvégzése "A" és "M" között történik a megfelelő INTEL assembly utasitással. Kivétel a shift-elés, összehasonlitás és maszkolás, ezeket run-time rutinok végzik. Ilyenkor a source operandus a B regiszterbe kerül, a müvelet pedig az akkumulátoron hajtódik végre. Szorzás és osztás karakter tipusu adatokra nincs értelmezve.

Az integer müveletek run-time rutinhivásokra fordulnak. Hivás előtt

D-E
B-C tartalmazza a két operandust

Hivás után

D-E -ben van az eredmény

B-C tartalma közömbös

A real műveletek esetén a run-time rutinok a paramétereket a gép hardware stack-jéről veszik, az eredményt a B-C-D-E real regiszterbe teszik.
Az elemi függvényekre external procként lehet hivatkozni.

Kontroll strukturák

Proc hivás esetén a paraméter átadás a hardware stacken történik.
Amennyiben a proc eredményt is ad vissza, visszatéréskor az eredmény a tipusának megfelelő regiszterben van /A,D-E, B-C-D-E/.

Az eljárás forditásánál a végrehajtás elején át kell állitani a stack pointert, mert a paraméterek /és a local változók/ hivatkozásait a compiler már ahhoz relativen adja meg.

Az IF-THEN-ELSE, a CASE és a ciklus utasitásokból az intermediate kódban már csak az ugrások és feltételes ugrások maradnak.

5. Kiegészitések

A mikroprocesszorokból felépitett mikrogépekben több processzor és több fajta memória /RAM,ROM/ használható.

A GÉZA nyelv nem tartalmaz párhuzamos processzorok programozására szolgáló különleges eszközöket. Az egyes processzorok programjai azonban külön-külön lefordithatók.

A memóriák egy része "read-only" ezért a generált kódban külön kell választani azokat a részeket amelyek majd ROM-ba ill. RAM-ba kerülnek.

Ez a szétválasztás elsősorban a deklarációs részre vonatkozik. A local változókkal nincsen probléma, mivel azoknak a hardware stack-en foglalódik hely a proc végrehajtása idején. Az irható változóknak értelemszerüen RAM-ba kell kerülni, azonban a konstansok és statikus táblázatok ROM-ba égethetők a végrehajtható kóddal együtt. Jelenleg csak a deklarációs rész és a végrehajtható kód van szétválasztva, azonban az intermediate kód lehetőséget ad több memória osztály és bázisregiszter kezelésére. Ennek megvalósitása a közeljövő feladata.

A programok kezdőértékeinek megadása minden futtatáskor a programhoz tartozó RAM terület inicializálását teszi szükségessé. A generált kód még eléggé terjengős. Javitási lehetőségek azonban vannak. Az intermediate kód magában hordoz néhány még ki nem használt lehetőséget. Ilyen pl. a tömb, ill. struktura elemek elérése, ami a jelenlegi kódgenerátornál elég nehézkesen történik.

Esetleg célszerü lenne, ha a kódgenerátor opcionálisan két különböző jellegü kódot állitana elő.

Az egyik egy rövidebb de lassabb kód lenne, ahol bonyolult címzések, load-olások run-time rutin hivásokra fordulnának, a másik egy hosszabb, de gyors kód lenne, ahol csak az emlitett integer és real müveleteket hajtanák végre run-time rutinok.

6. Összefoglalás

A GÉZA nyelv célkitüzése: hatékony kód. Kezdetben kérdéses volt, mennyire hatékony kódot lehet az INTEL-re generálni. A kód ma még eléggé terjengős, de már több javitási lehetőség látszik. Külön feladatot jelent a több, fizikailag különböző tároló és processzor kezelésének megoldása is.

Nagyon hasznos volt, hogy a már meglévő TPA 70 kódgenerátor segitségével, az INTEL kódgenerátor irásával párhuzamosan már lehetett próbálni a majdani programokat. A compiler jelenlegi változatánál opcióként meg lehet adni, hogy milyen nyelvü kód generálódjék.

Megjegyezzük még, hogy a GÉZA nyelv nem csak rendszerprogramok irására használható. Egy erősen számolásigényes program
/3D felületek ábrázolása/ FORTRAN-ból átirva rövidebb és mintegy husszor gyorsabb kódot eredményezett a TPA 70-en.
/Gaál Balázs és Várady Tamás szóbeli közlése./

Irodalomjegyzék

/1/ GERHARDT, G.: A programining language for system development
MTA SZTAKI 1976.

/2/ GERHARDT, G.: Intermediate code structure and format.
MTA SZTAKI 1978.

/3/ RATTINGER, M.: BASIC interpreter a GD80 GC-n
Előadás ugyanezen a konferencián.

/4/ INTEL 8Ø8Ø Assembly Language Programining Manual
INTEL Corp. 1976.

MÁGNESES ADATHORDOZÓKON SÜRITETT ADATTÁROLÁST ÉS KITERJESZTETT VISSZANYERÉST BIZTOSITÓ ÁLTALÁNOSITOTT ELJÁRÁS

Eifert Gyuláné - Szécsi Károly
FÜTI Programfejlesztési O.

A Fővárosi Épitőipari Üzemgazdasági és Ügyviteltechnikai Iroda - a FÜTI - Programfejlesztési Osztályán egy adatfeldolgozó rendszer tervezése során merült fel egy probléma, amelynek megoldása vezetett az ismertetendő általánositott sürített tárolást és kiterjesztett visszanyerést biztositó alprogram logikák kifejlesztéséhez.

A probléma a következő volt: egy-egy dolgozóra vonatkozóan sokféle egyedi adatot és változó számban ismétlődő "mezőcsoport"-ot kellett kezelni. A mezőcsoporton több egyedi mező együttese értendő; az egyedi mezők együttes megjelenése - a mezőcsoport - jellemez egy magasabb fogalmat. A dolgozó fizetéséből levonandó letiltás magasabb fogalmát például a következő egyedi mezők jellemzik: a letiltás jogcime, a bruttó tartozási összeg, a levonás havi mértéke, a fennálló aktuális tartozás, a kedvezményezettre vonatkozó adatok, stb. A felsorolt egyedi mezők alkotják a letiltási mezőcsoportot. Egy dolgozóra több letiltás vonatkozhat; a mezőcsoport tehát változó számban fordul elő. A mezőcsoporthoz tartozó egyedi mezők egy logikai adatstruktúrát alkotnak. A dolgozóra vonatkozó egyedi mezők és mezőcsoportok pedig, egy magasabb szintü, második logikai adatstruktúra egységet jelentenek.

A tervezendő rendszerben tehát egy bonyolult logikai adatszerkezetet kellett kezelni, leképezni. Az adatábrázolásra három módszer kinálkozott: a többszintes struktúráltság figyelembevételével "adatláncok" létrehozása; a mezőcsoportok változó számú előfordulását megengedve egy változó hosszú logikai rekord tárolása; az egyes mezőcsoportok ismétlődésének maximális számát meghatározva - a maximális lehetőséget olyan dolgozóra vonatkozóan is biztositva, ahol a mezőcsoportban esetleg egyetlen aktivitás sincsen - egyetlen, igen hosszú és nagyon gazdaságtalanul kihasznált logikai rekord létrehozása.

Az "adatlánc" azt jelenti, hogy "egy logikai adatot" több részletben, a részek közötti összefüggés feltüntetésével tárolunk. Az adatlánc egyes elemei igy éppen a dolgozóra vonatkozó egyedi mezők, továbbá az aktiv mezőcsoportok, mindegyik egy-egy egységként. Az adatképzésnek ez a módszere adatbáziskezelő programcsomag felhasználásával, vagy saját egyedi adatkezelő rendszer kifejlesztésével valósitható meg.

A változó hosszú logikai rekord kialakitása, s a mezőcsoportok ismétlődési számának megoldásával járható út, a mezőcsoportok azonositására állandó logikai müveleteket igényel.

A harmadik módszer látszólag teljesen elméleti, mivel rendkivül gazdaságtalan. Viszont, ha a fizikai tárolásnál el lehet kerülni az üres poziciók és nulla értékek tárolását, akkor ez a módszer a legkényelmesebb és legegyszerübb. Igy jutottunk el a feladat megoldása közben a süritett tárolás, a komprimálás problémájához.

A süritett tárolás elég sokféle megoldása lehetséges. A sürités lényege az, hogy "ál-információt": a mező alapértelmezése szerinti információt - az üres karaktereket, nullákat, stb. - és az ismétlődéseket nem tárolják, illetve nem eredeti formájukban tárolják, hanem egy logikai információ - átalakitás, sürités után. Minden komprimálással szemben elsődleges követelmény az egyértelmü visszaalakithatóság.

Alapvetően két irányzatot lehet megkülönböztetni: az egyik a mezőorientált, a másik a byteorientált komprimálás.

A mezőorientált süritési eljárás lényege, hogy a mezőket tekinti a komprimálás alapegységének. Minden mező kap egy azonositót és a sürités, tárolás, illetve a visszaalakitás a mezők, mezőcsoportok és a mezők aktuális értéke körében bonyolódik. Ennek az eljárásnak az a hátránya, hogy a mezőazonositók és mezőértékek valamilyenfajta tárolásával nagyon erősen kötött a konkrét mezőszerkezethez. Gazdaságosságát - a sürités hatékonyságát- nagy mértékben befolyásolja a tényleges mezőaktivitás /rossz esetben a "süritett" információ hosszabb lehet, mint az eredeti !/.

A másik süritési módszer a byte-orientált. Lényege, hogy a logikai adatban a byte-ok ismétlődését figyeli, és azonosság esetén tárolja az ismétlődő byte-ot, és az ismétlődés számát. Ennek a megoldási módszernek nagyon komoly problémája, hogy a visszaalakitásnál /dekomprimálásnál/ meg kell tudni különböztetni az ismétlődési számot a tárolt byte-tól - ami, ha egy tárolandó byte-on 256 féle információt engedünk meg, elég körülményes.

A legelterjedtebb komprimálási és dekomprimálási eljárás a DOS operációs rendszer forráskönyvtárával kapcsolatban használatos.

A probléma lényegének feltárása után támadt egy megoldási ötletünk, amely alkalmas minden olyan byte-sorozat tárolására, amely szöveges információkat, pakolt és zónázott számokat és bizonyos hexadecimális értékeket tartalmaz, viszont nem tartalmaz bináris ábrázolású számokat.

Az ötlet alapján kifejlesztett eljárás minden esetben rövidebb - de legfeljebb azonos hosszúságú - komprimált byte-sorozatot állit elő, mint az eredeti komprimálatlan byte-sorozat, és egyértelmü dekomprimálást biztosit. Érdekességként megjegyezzük, hogy egy byte hosszú komprimálandó információ esetén is igaz a rövidebb-egyenlőség állitása.

Szakmai körben ismert, hogy néhány felhasználó alkalmaz komprimálási eljárásokat, de ezekről igen kevés információ áll rendelkezésre. A FÜTI által kifejlesztett komprimálási és dekomprimálási eljárás az általunk eddig ismert összes eljárásnál hatékonyabb, a többi módszerrel elérhetőnél rövidebb komprimált byte-sorozatot állit elő, és eljárását tekintve hasonlót nem ismerünk.

A komprimálási eljárás olyan hatékonynak és használhatónak bizonyult, hogy érdemesnek tartottuk a tervezett adatfeldolgozó rendszertől független általános - bármely DOS felhasználó által hozzáférhető - kifejlesztésre.

Az eljárást változó rekordhosszra nem fejlesztettük ki, mivel a változó hosszúságú rekordok mindig

konvertálhatók fix hosszúakká, vagy több fajta fix hosszú struktúra valamelyikébe.

Sűritési eljárásunk elsősorban szekvenciális szervezésü állományokra alkalmazható, de ezen ismertető végén a továbbfejlesztési elképzelésekben erre a kérdésre még visszatérünk.

A komprimálási-dekomprimálási eljárásokat megvalósító logika modul formában áll a felhasználó rendelkezésére. A feladatot megoldó logikáknak négyféle tipusa van a későbbiekben meghatározott osztályozás szerint. A komprimálást és dekomprimálást külön modulok hajtják végre, igy a csomag összesen nyolc modulból áll.

A modulok magasszintü programozási nyelven és assembler nyelven irt programokból az alprogramhivás szabályai szerint hivhatók.
Tekintettel arra, hogy a felhasználási programok összes komprimálási, illetve dekomprimálási igényét egy komprimáló, illetve egy dekomprimáló modul bonyolitja, a komprimálás-dekomprimálás használatánál két osztályozó ismérv határozható meg: egy vagy több file-ról van-e szó, illetve az egy-egy file-ba kerülő logikai rekordok azonos struktúrájúak-e - mezőfelépitésüek-e -vagy több fajta struktúrájúak.

A két osztályozó ismérv két-két konkrét értékének megfelelően négy esetet különböztethetünk meg:

1./ Egy file - egy fajta logikai struktúra,
2./ Egy file - több fajta logikai struktúra,
3./ Több file - egy-egy fajta logikai struktúra, és
4./ Több file - több fajta logikai strultúra.

A kifejlesztett eljárás csomag használata esetén az egyes esetben a következő alprogram neveket - entry neveket - és operandusokat kell használni:
AZ ÁLLOMÁNY IRÁSÁHOZ - KOMPRIMÁLÁSÁHOZ:

a./ az első, inicializáló hívásnál egy-egy programban csak egyszer:

CALL INCPRSS([{2400 / 2311 / 2314}], SYSnnn, file-név, str.név {, hossz});

b./ minden egyes logikai rekord komprimálásához és tárolásához:

CALL CPRSS{ (str.-név)} ;

c./ a komprimálás befejezéséhez - az utolsó rekord komprimálása után:

CALL CLOSESS;

AZ ÁLLOMÁNY OLVASÁSÁHOZ-DEKOMPRIMÁLÁSÁHOZ:

d./ az első - inicializáló hivásnál:

CALL INDPRSS([{2400 / 2311 / 2314}] , SYSnnn, file-név,
eof-cimke {,str.-név} {, hossz});

e./ minden egyes logikai rekord olvasásához:

CALL DPRSS { (str.-név)} ;

Az operandusok jelölésének és használatának magyarázata az alábbi:

[{2400 / 2311 / 2314}] - az inicializáló hivásnál kötelező megadni az input/output medium /fizikai egység/ tipusát; sorrenden: mágnesszalag, 7,25 MB mágneslemez és 29 MB mágneslemez,

= ha itt 2400 szerepel, akkor a "hossz" operandust kötelező megadni.

SYSnnn - az a programozói logikai egységszám adandó meg, amelyen a felhasználó az állományát kezeltetni kívánja / ASSGN, EXTENT/.

file-név - a felhasználó által meghatározandó szimbólikus név, amely a file programbeli azonositására fog szolgálni /DLBL/.

str.-név - a programozó által definiált rekord szimbólikus neve; az a főtároló terület, ahonnan a felhasználó kiiratni, illetve ahová beolvastatni kiván.
MEGJEGYZÉS: az "a" hivásnál a logikai rekord valamennyi mezőjét; a mező jellege szerinti üres, vagy nulla értékre kötelező inicializálni /de csak a komprimálást inicializáló hivásnál/.
Ahol a "str.-név" kapcsos zárójelben van, ott nem kötelező megadni, olyankor az inicializáló hivásban megadott területre történik az I/o müvelet. Ha mégis meg van adva, akkor lehet az inicializáló hivástól eltérő név is, ilyenkor a konkrétan megjelölt területre történik az I/o müvelet.

A "d" és "e" hivás közül az egyikben feltétlenül meg kell adni az "str. nev"-et, ha mindkettőben meg van adva az "e"-ben megadott él, a konkrét "e"-re.

hossz - csak szalag I/o esetén kötelező /egyébként felesleges/, a fizikai blokkhossz értékét kell itt decimális szám formájában megadni. /min.: 18b, max.: 32Kb/.

eof-cimke - az a szimbólikus cim, ahol a felhasználó a saját programjában az input file-vége kondiciót feldolgozza.

Az eljárás hivásának rendszere és parametrizálása nyilvánvalóan további kérdéseket vet fel.

Az állomány kezelését - deklarálás, pufferfoglalás, OPEN, fizikai rekordok I/o-ja, logikai rekord kezelés és CLOSE - az eljárás végzi. A felhasználónak csak a hivásnál operandusként a file-ra vonatkozó információkat és az ASSGN, DLBL és EXTENT Job Control kártyákat kell megadnia.

A blokkméretek meghatározását az eljárás lemez esetén automatikusan elvégzi. Szalag esetén azonban a blokkméret nem dönthető el az eljárásból; a pufferek méretének megtervezése a particióban rendelkezésre álló főtárral való gazdálkodást jelent, és igy programtervezési kérdés. Mágnesszalagos állományok felépitésénél ezért a felhasználónak kell az optimális blokkméretet megterveznie, majd a felépitett állományokat a meghatározott blokkmérettel használnia.

Mint látható, rewrite - újrairatási - funkció nem létezik, az update-elt logikai rekord komprimált hossza ugyanis igen nagy valószinüséggel különbözik az

előző hossztól, igy nem kerülhet az előző fizikai helyre. Az uptade-elés mindig új file-területre történik.

A második esetben - egy file, több fajta logikai struktúra - a következő formákat kell használni:
AZ ÁLLOMÁNY IRÁSÁHOZ -KOMPRIMÁLÁSÁHOZ:

a./ CALL INCPRSM([{2400 / 2311 / 2314}] , SYSnnn, file-név,

str.1.-név, str.2.-név{, ... str.9.-név} ,{hossz});

b./ CALL CPRSM (str.[{1. / 2. / . / . / 9.}] -név);

c./ CALL CLOSESM;

d./ CALL INDPRSM ([{2400 / 2311 / 2314}] , SYSnnn, file-név,

eof-cimke{,strá.-név {, P}} ,{hossz}) ;

e./ CALL DPRSM {([strá.-név / strá.-név, P])} ;

Az operandusok jelölése és használata - az alábbi kiegészitésekkel - lényegében azonos az egyes esetnél leirtakkal.

Az "a" hivással kapcsolatban az egyes struktúrákat megkülönböztető jelzéseket az inicializáló hivás előtt az adott struktúrába be kell irni - és nem lehet üres pozicióval és nulla értékkel struktúrát azonositani. Az inicializáló hivásnál maximum kilenc fajta logikai rekordtipust lehet megadni, és az összes később ebben a file-ban használatos tipust meg kell adni.

strá.-név - a leghosszabb logikai rekord részére lefoglalt terület szimbólikus neve, vagy annak a pointernek a szimbólikus neve, amelyre a felhasználó az összes struktúráját akasztani akarja /based/, há pointer használatos, akkor ezt, a "P" kulosszavas operandusnak követnie kell.

A harmadik és negyedik esetben - több file, egy-egy fajta logikai struktúra és több file, több fajta logikai struktúra - az eljárás használata hasonló, mint az egyes vagy kettes esetben, csak a hivott alprogram névben egy egytől kilencig használható sorszámmal ki kell fejezni, hogy éppen melyik file-ról van szó. A file-ok és az azonositó sorszámok egymáshoz rendelése az inicializáló híváskor történik.
Az eljárás maximum kilenc file-ig müködik.

Az alábbiak egy példát mutatnak be:

```
CALL INCPR1M ( 24ØØ; SYSØ17, DATA, FIZIKAI, ALKAL-
               MAZOTT, EGYÉB, 16ØØØ);
     .
     .
     .
CALL CPR1M (EGYEB);
     .
     .
     .
CALL INCPR2M (2311, SYSOO5, LEMEZ, HAVI, NEGYED-
              EVES, EVES);
     .
     .
     .
```

```
CALL CPR2M NEGYEDEVES ;
CALL CLOSE1M ;
CALL CLOSE2M ;
.
.
.
```

Kifejlesztett eljárásunk tényleges hatékonyságát az "Épitőipari Cikklista Törzsállomány"-t sürítésével vizsgáltuk. Ez az állomány 180 ezer darab 105 byte-os logikai rekordot tartalmaz. A mezők aktivitása és kihasználtsága az adatfeldolgozó rendszerekben megszokott mértékü volt - sokkal intenzivebb, mint abban a logikai rekordban, amelyre eljárásunkat terveztük.

Az eredmény számunkra is meglepő volt, a mintegy 6000 sáv terjedelmü állományt az eredeti terület 51 %-ára sikerült összesüritenünk.

Eljárásunk továbbfejlesztését két irányban tervezzük.

A visszakeresési problémakörben nagy mértékben növelné a visszakeresés gyorsaságát, ha kulcsra rendezett szekvenciális komprimált input állományoknál a felhasználó program kulcsszerinti dekomprimálási kérését az eljárás úgy hajtaná végre, hogy a komprimált állományban megkeresné a kért logikai rekordot és csak azt dekomprimálná és adná vissza a felhasználó programnak.

Továbbá foglalkozunk egy olyan modul sorozat kifejlesztésével, ahol a felhasználói program és a komprimáló /dekomprimáló alprogram kommunikációja a logikai rekord szintjére tevődne át; ebben a rendszerben a felhasználói program az alprogramtól független állománykezelést valósithatna meg, és direkt és indexelt

szekvenciális szervezésü állományok, illetve változó rekordhossz közvetlen használatára is lehetőség nyilna.

Végül arra szeretnék rámutatni, hogy a komprimálás/dekomprimálás használata leveszi a rendszerszervezők válláról azt a néha nyomasztó felelősséget, hogy a mezők és a mezőkön belüli poziciók számának megtervezésével nagy mértékben meghatározzák egyrészt az adatfeldolgozás gyorsaságát, másrészt az adatfeldolgozó rendszer rugalmasságát. A komprimálás/dekomprimálás használata ugyanis szinte kizárólag az aktiv byte-ok átvitelezésére veszi igénybe az input/output rendszert, az üresen hagyott mezők szinte semmilyen terhet nem jelentenek.

AZ XPL RENDSZER ÉS ALKALMAZÁSAI A TÁVKÖZLÉSI KUTATÓ INTÉZETBEN

Endrődi Tibor - Fehér Iván - Vanczák József
Távközlési Kutató Intézet

Bevezetés

A Távközlési Kutató Intézetben folytatott alapsoftware fejlesztéssel kapcsolatos munkák meggyorsítása érdekében a hagyományosan alkalmazott assembly nyelvü realizáció helyett magasszintü nyelvet kívántunk alkalmazni. A megfelelő nyelv keresése közben irányult a figyelmünk az 1970-ben publikált XPL rendszerre.[1]. Ez a rendszer nagyon jó segédeszköznek látszott magasszintü programozási nyelvek fejlesztésére. Komponensei között szerepelt egy táblavezérelt szintax elemző program /SKELETON/, a szükséges táblákat előállító elemző generátor /ANALYZER/, és egy XPL nyelvről IBM SYSTEM/360 gépi nyelvre fordító fordítóprogram /XCOM/. Ezeknek az eszközöknek a birtokában került sor a TKI-ban az XXPL nyelv kifejlesztésére [3, 4].

A dolgozat első részében ismertetjük azt az utánhúzó eljárást, amely alkalmazásával az XPL rendszert müködőképessé tettük, a második részben összefoglaljuk az új nyelv fejlesztésével kapcsolatos tapasztalatainkat, végül bemutatjuk a nyelvet implementáló két fordítóprogramot.

1. Az XPL rendszer implementációja

A publikált XPL rendszer az egyes komponensek XPL nyelvü forrásprogramjait tartalmazza. Ezért, hogy a rendszerhez

hozzájussunk, az XCOM fordítóprogramot kellett működőképes állapotba hozni [2]. Olyan módszert kerestünk, ami minimális emberi munka mellett garantálja számunkra az eredeti XCOM IBM SYSTEM/360 gépi nyelvü változatát.

Az XCOM és az XPL nyelv sajátosságainak tanulmányozása során végiggondoltunk egy olyan közbülső /IML/ nyelvet, amire XPL nyelven írt programok leképezése viszonylag egyszerüen elvégezhető. Ennek a nyelvnek a rögzítésével egyben definiáltunk egy virtuális számítógépet is, amelynek utasításai az említett közbülső nyelv utasításaiból állnak. Ezek után a rendelkezésrünkre álló ICL SYSTEM/4-50 számítógép gépi nyelvén megírtunk egy fordítóprogramot /XCOMV/, és realizáltuk a virtuális számítógépet /VM-SIMULATOR/. Az XCOMB XPL forrásprogramokat képes lefordítani a közbenső nyelvre, a VM-SIMULATOR IML nyelven írt programokat hajt végre.

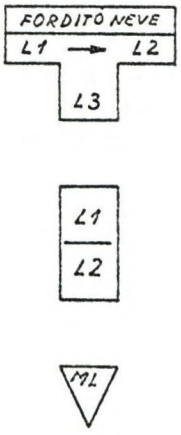
Az XCOM célnyelvi változatához a következő lépésekben jutottunk:

- az XCOMV program futtatásával lefordítottuk az XCOM-ot IML nyelvre,
- a VM-SIMULATOR-ral lefuttattuk az XCOM-nak ezt a közbenső változatát úgy, hogy bemenetként az XCOM eredeti, forrásnyelvi változatát adtuk meg. Mivel a közbenső változat az eredeti algoritmusok szerint futott, a kimeneten az XCOM IBM SYSTEM/360 gépi nyelvü változata jelent meg.

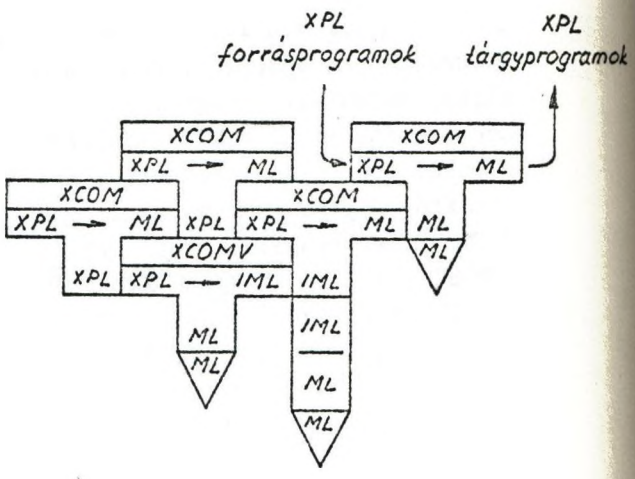
Az ismertetett módszer látszólag komplikált, de számos előnnyel rendelkezik a kézi fordítással szemben. A rendszer elkészítésével kapcsolatos munka így mintegy 4000 gépi utasítás megírása volt, míg a lefordított XCOM 13500 gépi utasítást tartalmaz. Az alkalmazott segéd-

programamok megírásakor nem kellett ügyelni a jó hatásfokra, hiszen csupán egy alkalommal használtuk ezeket.

A fordítóprogramok ilyen típusú előállítását utánhúzó /bootstrap/ módszernek nevezzük.



1.ábra



2.ábra

Az utánhúzó eljárások szemléltetésére jól használható jelölési rendszer látható az 1. ábrán, az alkalmazott utánhúzó eljárás pedig a 2. ábrán található. Az első ábrához útmutatásul a következő megjegyzéseket fűzzük:

- A T elem egy olyan fordítóprogramot jelöl, amely L1 nyelvről L2 nyelvre fordít, és L3 nyelven van megírva.
- Az I elem egy olyan interpretert azonosít, ami L1 nyelven írt programot tud végrehajtani, és L2 nyelven van írva.
- A V elem pedig az ML nyelvü program végrehajtását jelöli.

A bemutatott eljárás végrehajtása után már egyszerüen lehetett generálni az XPL rendszer többi elemét. A továbbiakban ezeket fogjuk ismertetni.

1.1. SKELETON program

A SKELETON program egy táblavezérelt, redukáló típusú, szintax elemző automata. Egy adott nyelv szintaxisát a vezérlő táblák rögzítik. A táblák cseréjével - ami igen egyszerü a program felépítésénél fogva - a SKELETONT különböző nyelvek elemzésére lehet felhasználni. A SKELETON az elemzéssel párhuzamosan - az egyes produkciók felismerésekor - szemantikus akciók indítását is vezérelheti. Igy alapul szolgálhat fordítóprogramok vezérlésére - ilyenkor kódok generálása a szemantikus akció, de minden olyan alkalmazói program részét is képezheti, ahol bonyolult felépítésü nyelv biztosítja az ember és a gép közötti kapcsolatot: ilyenkor a szemantikus akció jelentheti az üzenetek feldolgozását.

1.2. ANALYZER program

A SKELETON vezérlőtábláit a szintaxis szabályaiból az ANALYZER program generálja. Egyben azt is ellenőrzi, hogy a szabályokat megadó nyelvtanra alkalmazható-e a SKELETON-ban használt elemző algoritmus. Hiba esetén részletes diagnózist ad, a javítható esetekben elvégzi a nyelvtan megfelelő módosítását is.

2. Az XXPL fejlesztésével kapcsolatos tapasztalatok

Az XPL rendszert 1973. közepétől használjuk. Az elmúlt évek során számos feladatot oldottunk meg segítségével.

Az XPL nyelv előnyös tulajdonságainak tartjuk, hogy
- nyelvi konstrukcióival lehetőséget ad a struktúrált programozásra,
- előnyösen használható nem numerikus algoritmusok realizálására,
- könnyen olvasható, önmagukat dokumentáló programok írhatók vele,
- és könnyen elsajátítható nyelv.

Ugyanakkor hiányossága, hogy:
- kicsi az adattípusok választéka. Nincs lehetőség összetett struktúrájú adatok használatára. Nincs referencia típusú változó,
- a nyelv nem tartalmaz olyan konstrukciókat, amelyek használatával a felhasználó optimalizáláshoz szükséges információkat közölhet a fordítóprogrammal. Nehéz összefüggő adatterületek átmásolása, ami egy utasítással elvégezhető lenne. A növekvő /UP_TO/ és a csökkenő /DOWN/TO/ ciklusutasítás, a +=, -= utasítások bevezetése optimálisabb kód generálását biztosítja.

Az XPL-nek - mint alapsoftware írására használt eszköznek - további hiányosságai az implementációval kapcsolatosak:
- az XPL tárgyprogramok végrehajtásához egy interface program - a szubmonitor - szükséges. Ez biztosítja a kapcsolatot a program és a gép operációs rendszere között. A szubmonitor, kötött felépítéséből adódóan, korlátozza az elvégezhető programfeladatok körét. /Vitathatatlan előnye viszont, hogy a portabilitás lehetőségét növeli a különböző IBM kompatibilis számítógépek operációs rendszerei között./
- az eredeti XPL rendszer nem támogatja a moduláris software fejlesztést.

Az XPL nyelvvel és rendszerével kapcsolatos tapasztalataink adtak használható alapot a célul kitüzött új nyelv, és rendszerének specifikálásához. Olyan eszközhöz akartunk jutni, amely ESZR nagy- és kisgépeken egyaránt használható rendszerprogram jellegü feladatok realizálásához.

Az új nyelv, az XXPL
- felülről kompatibilis az XPL-lel, így annak minden előnyös tulajdonságával rendelkezik,
- tartalmazza az általunk hiányolt nyelvi elemeket.

A nyelvet realizáló fordítóprogramot - a fenti kívánalmaknak megfelelően - két változatban készítettük el, az ESZR nagygépekre és az R 10 kisszámítógépre. A moduláris programfejlesztés támogatásához pedig elkészítettünk ehy szerkesztő programot. A programok fejlesztése teljes egészében az XPL rendszerre támaszkodott.

Az XXPL nagygépes változatának fejlesztése 1975. közepén kezdődött, és 1977. közepére készült el. Azóta folyamatosan használatban van intézetünkben.

Komoly nehézségeket okozott az XPL nyelv modularitásának hiánya. A belövések ideje alatt egy-egy kisebb javítás kedvéért az egész, kb. 5000 XPL utasítássort tartalmazó programot le kellett fordítani, ami tetemes gépidőt jelentett. A gyorsabb gépeken /ICL SYSTEM/4-70, IBM 370/145/ 10 perc, R 30-on 45 perc volt egy fordítás ideje.

A modularitás hiányában - mivel nem volt nyelvi kényszer a kapcsolódó programrészek határfelületeinek alapos végiggondolására - a programstruktúra is deformálódott. Ez óhatatlanul a belövés elhúzódását vonta maga után.

Az XXPL fordító R 10-es változatának fejlesztése 1976.

végén kezdődött. A fordító ez év elejétől próbaüzemben müködik.

Készítését ugyanazok a problémák hátráltatták, mint a nagygépes fordítóét. A párhuzamos fejlesztés miatt itt sem lehetett még kihasználni a modularitás adta lehetőségeket.

3. Az XXPL magasszintü programozási nyelv fordítóprogramjainak felépítése

A fordítási folyamat három fázisra bontható. Az elsőben ellenőrizzük a forrásszöveget, a másodikban szerkesztésre alkalmas tárgykódot generálunk, majd a harmadikban elkészítjük a müködtető rendszerhez illeszkedő tárgykódot.

A többmenetes szerkezet választását a következő meggondolások indokolják:

/i/ A gépfüggetlen funkciók különválasztásával tetemes programozási munka takarítható meg több fordítóprogram változat készítése esetén.

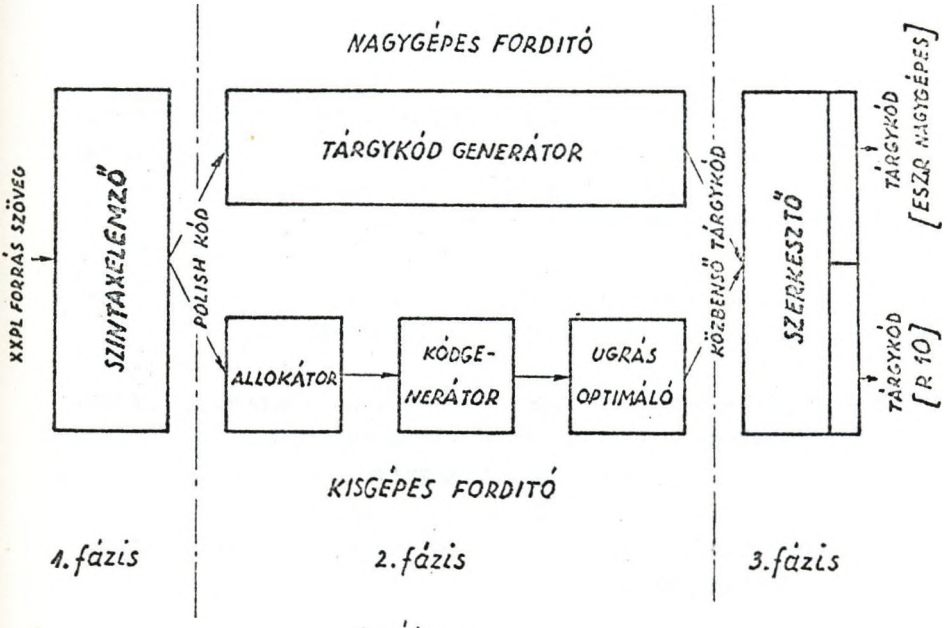
/ii/ A funkciók tagolásával csökkenteni lehet a modularitás hiányából adódó problémákat.

/iii/ A többmenetes inkrementális processzálás csökkenti az egyes menetek operatív társzükségletét.

/iv/ Az ilyen szerkezetü fordítóprogramok bővítése új menetek közbeiktatásával könnyen elvégezhető.

A fordítóprogramok struktúráját a 3. ábra mutatja be.

A fordítás első fázisát egy menetben a SZINTAXELEMZŐ program realizálja. A SZINTAXELEMZŐ a forrásszöveg formai helyességét ellenőrzi, majd a forrásprogramot egy közbülső nyelvre fordítja.



3. ábra

Ezek a feladatok az egész fordítási folyamat gépfüggetlen részét jelentik, ezért ez a program része mindkét fordítóprogramnak.

A SZINTAXELEMZŐ magja a SKELETON, amit a közbülső nyelvre fordító eljárások egészítenek ki. A közbülső nyelv kiválasztásánál a következő szempontokat vettük figyelembe:

- legyen alkalmas a forrásprogram olyan reprezentánsának a megfogalmazására, ami elegendő információt hordoz lokálisan jó kód generálásához,
- legyen könnyen kezelhető /illeszkedjen jól a következő fázis algoritmusaihoz/.

Megítélésünk szerint ezeknek a szempontoknak legjobban egy POLISH formátumú nyelv felel meg.

A következő fázis az erősen gépfüggő feladatokat tömöríti. Megtörténik az adatok allokációja, elkészülnek a magasszintü utasítások célnyelvü változatai, majd ezekből a közbenső tárgykód.

A feladatok gépfüggő jellege miatt ebben a fázisban különválik a nagy és kisgépes feldolgozás.

A nagygépes változat egy programból a TÁRGYKÓDGENERÁTORból /T_GENER/ áll. A T_GENER magja egy PUSH-DOWN automata, amit az ALLOKÁTOR, a KÓDGENERÁTOR és a TÁRGYSZERKESZTŐ egészít ki. Az ALLOKÁTOR az optimális tárkihasználásnak megfelelően allokálja az adatokat, a KÓDGENERÁTOR elkészíti az utasításoknak megfelelő gépi kódú szekvenciákat, a TÁRGYSZERKESZTŐ pedig a közbülső tárgykódot.

A KÓDGENERÁTOR adatbázisa a PUSH-DOWN automata verme. A kódgenerálás pillanatában a verem tetején az aktuális leképezésre váró operáció, és az operandusai vannak. A verem többi eleme a müveletet kezdeményező operációkat tartalmazza. A megfelelő kódszekvencia kiválasztásánál így figyelembe lehet venni az összes kezdeményező operáció igényeit is, szemben egy esetleges általános megoldással. Jól illusztrálja ezt a következő példa:

IF A=B THEN...

Az A=B utasításra általános esetben logikai értéket előállító kód generálódik. Annak ismeretében viszont, hogy az A=B egy IF utasítás operandusa, elegendő egy feltételkódot előállítani.

A kisgépes változat ezeket a feladatokat három menetben oldja meg. A további tagolásra azért volt szükség, mert így egyrészt jobban ki lehetett használni az akkor rendelkezésre álló emberi kapacitást /ii/, másrészt ezzel biztosítani lehetett, hogy a fordítóprogram R 10-en is futtatható legyen /iii/.

A három menet az ALLOKÁTOR, a KÓDGENERÁTOR és az UGRÁSOPTIMÁLÓ.

Az ALLOKÁLÓ az adatok allokálásán kívül "kiszüri" a POLISH-ból a további feldolgozás szempontjából érdektelen elemeket, Az operandus hivatkozásokat az operandusok típus és allokációs leírásával helyettesíti, és elvégzi a fordításkor kiértékelhető számításokat is.

A KÓDGENERÁTOR felépítése megfelel a nagygépes változatnak. Az ugrások kivételével itt is megtörténik az utasítások leképezése, majd a programgráfnak egy olyan leírása készült, ahol az élek a lehetséges vezérlésátadási útvonalak, a csomópontok pedig a generált kódszekvenciák. Az UGRÁSOPTIMÁLÓ ennek a leírásnak ismeretében generálja a környezetnek legjobban megfelelő ugrási utasításokat, majd elkészíti a forrásprogram közbülső tárgykódját.

A feldolgozás harmadik fázisában a közbenső tárgykód operációs rendszer függő tárgykódú megfelelőjét készítjük el. Azért készül közbenső tárgykód, mert így
- a második fázis kimenete függetleníthető egy adott operációs rendszer megkötéseitől, és
- lehetőség nyílik olyan szolgáltatásokra is, amelyeket általában a szerkesztő programok nem tudnak nyújtani.

A közbülső tárgykód megegyezik mindkét fordítóprogramnál, így a szerkesztő nagy része közösen használható mindkét esetben. Ez a program már XXPL nyelven íródott, így kihasználja a modularitás adta lehetőségeket is. A következő feladatokat látja el:

- elvégzi modulok paramétereinek típusellenőrzését /Akinek volt már alkalma PL/1-ben programozni, az fel tudja mérni ennek a jelentőségét!/,
- kitölti az externál változóra hivatkozó gépi utasítás címrészét. Ezzel biztosítja, hogy az externál változók elérésének hatásfoka megegyezzen a többi változó elérésének hatásfokával,
- generálja az adott operációs rendszernek megfelelő tárgykódot.

Végezetül felsorolunk néhány software terméket, amelyek eddig az XPL-XXPL rendszerrel alapozva elkészültek:

- elektronikus telefonközpontok operációs rendszerének generálását végző programok az LM ERICSSON cég számára /3 program, kb. 20000 sor/,
- a TKI-ben fejlesztett, elektromos áramkörök gépi tervezését végző rendszer vezérlő nyelvét értelmező modul,
- nyelvfejlesztési feladatok:
  Az R 10 TIME-SHARING rendszeréhez fejlesztett BASIC alrendszer.
  R 10-en működő szintax elemző program a CAI nyelvhez.
  Az XXPL fordító programok.
- távadatfeldolgozó rendszerekhez berendezés tesztelését végző program,
- szöveg szerkesztő-javító program,
- LR1 elemző generátor program.

A szerzők ezúton is köszönetet mondanak Német Józsefnek és Nagy Antalnak szakmai irányításukért.

Irodalomjegyzék

1 McKeeman, Hornig, Wortman: A COMPILER GENERATOR
Prentice-Hall, INC., 1970

2 Nagy Antal: XPL: ALAFSOFTWARE IRÁSÁRA ALKALMAS PROGRAMNYELV
Információ Elektronika, 1974/3.

3 Endrődi Tibor, Molnár Gábor, Vanczák József: XXPL PROGRAMOZÁSI NYELV, FEJLESZTŐI KÉZIKÖNYV /R 30/
Intézeti tanulmány, 1976.

4 Fehér Iván, Kocsis Ferenc, Szendrényi Tibor: XXPL PROGRAMOZÁSI NYELV, FEJLESZTŐI KÉZIKÖNYV /R 10/
Intézeti tanulmány, 1978.

IJS
/INTERACTIVE JOB-MANAGEMENT SYSTEM/

Farkas Anikó - Kerekes Iván
Számitógépalkalmazási Kutató Intézet

Tartalomjegyzék

0. A projekt elinditásának célja

Intézetünkben már évek óta folynak kutatások a nagyüzemi software gyártás eszközbázisának fejlesztése témakörben. Ezen kutatások egyik részeredménye az intézet gyakorlati igényeit is kielégitő párbeszédes üzemmódu programozási rendszer kifejlesztése. A fejlesztési terv kialakitása során alapvető feltételként szerepelt a rendelkezésre álló hardware és software eszközök maximális hasznositása. Kihasználva az R22 gép jelentős feldolgozó kapacitását és az R10 előnyös interaktiv képességeit, e két rendszer hardware és software szintü összekapcsolásával alakitottuk ki az IJS-t, mely lehetővé teszi az R10-hez csatlakozó képernyős terminálokon keresztül az interaktiv programfejlesztést és job bevitelt az R22-be.

1. IJS Átfogó ismertetése

1.1. Feladata, célja

Az R10-hez csatlakozó terminál/ok/-ról vezérelve a felhasználó számára minden olyan feladat ellátásának biztositása a nagygépen /R22 ill. más ESZR nagygép/, amely helyi batch környezetben is megadható és végrehajtható. Ezen alapvető cél mellett az IJS, a lokál-batch környezetben nem teljesithető interaktivitást is nyujtja oly módon, hogy közben nem korlátozza az egyidejüleg folyó batch-feldolgozás számára az operációs rendszer által biztositott particiószámu multiprogramozást.

1.2. Az IJS eszköz és rendszerfeltétele

Az IJS-t ugy fejlesztettük ki, hogy mellette mindkét gép megtartotta autonom jellegét és konfigurációját.

- ESZR nagygép /itt: R22/ oldal:

  DOS+Power /DOSPW/ operációs rendszer /IBM DOS 26.2 + Power II/4.1 release/

  Az IJS futtatható ESZR/DOS mellett is, minthogy a DOS-tól csak a multitasking supervisor lehetőséget várja el;

  Az IJS rendszer kb. 50 K-t igényel az F1 partició-ban és kb. 5 cilindernyi lemezterületet használ a Power igényein kivül.

  Az IJS müködése a konfiguráció lemeztipusától független.

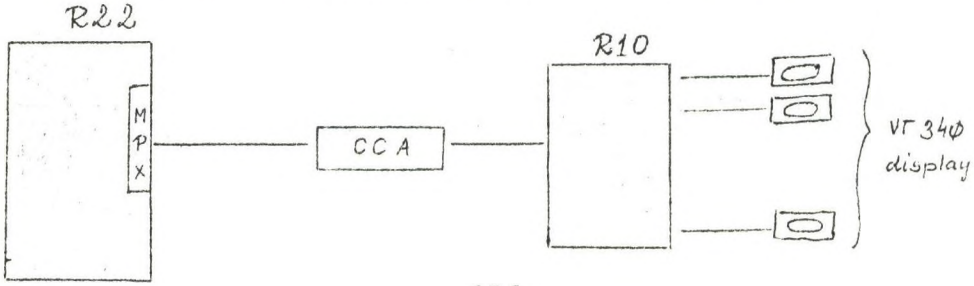
- R10 kisszámitógép oldal:

  IDOS operációs rendszer time-sharing változata;
  R10 központi egység 32 Kszó memóriával; legalább egy lemezegység /hatékony müködéshez legalább 2/
  min. 1 db multiplexer /4 display-enként 1 multiplexer/; sornyomtató;
  min. 1 mágnesszalag-egység;

- A hardware interface-t megvalósitó: CCA /VT 55040/, amely az ESZR nagygép multiplex csatornájára kapcsolódik és azon 16 alcsatornát foglal le.

- VT 340 terminálok az R10-re illesztve.
  Max. 16 darab

A hardware kapcsolat sematikus képe az alábbi:

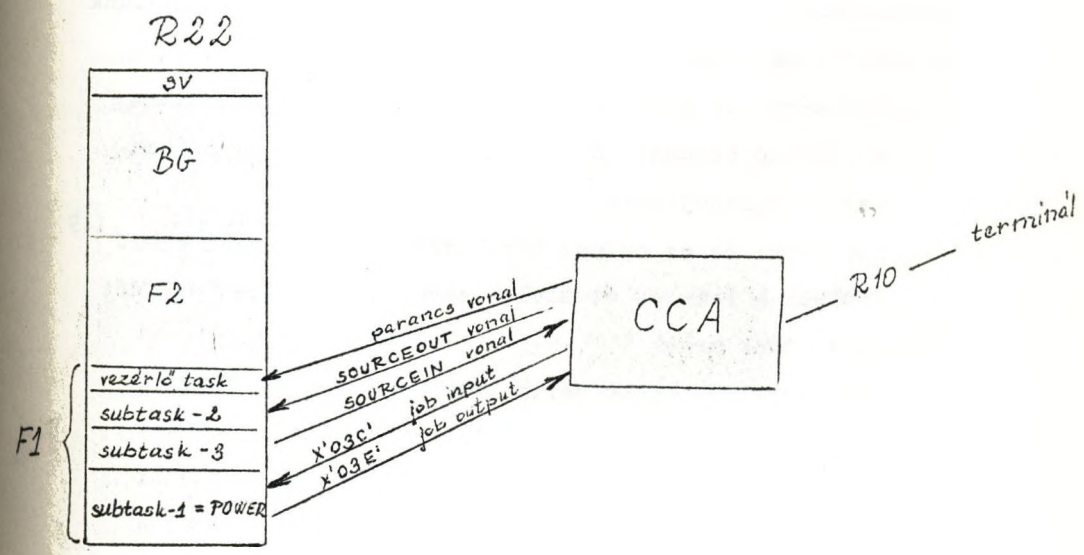


1.3. Az IJS nagygépes software oldala

A teljes IJS a felhasználónak az alábbi lehetőségeket nyujtja:

- job létrehozása és ellenőrzése a terminál mellett, interaktiv szerkesztési lehetőségekkel;
- terminálnál létrehozott job átküldése a nagygépbe futtatásra;
- job-output irányitásának megválasztása;
- forrás-book létrehozása a terminál mellett, interaktiv szerkesztési lehetőségekkel;
- forrás-book felvétetése a DOSPW rendszer forráskönyvtárába;
- meglévő forrás-book lekérése a DOSPW rendszer forráskönyvtárából;
- eljárás könyvtár kialakitás nagygépen és választott eljárás inditása felhasználói terminálról;

Ezen szolgáltatásokra a következőképpen készitettük fel a rendszert:



Az IJS-t a Power mellett az F1 particióban helyeztük el, felhasználva a DOS, partición belüli multitasking adottságát. A job input/output vonalakat, ha már előzőleg a supervisor táblájába felvétettük, egyszerü Power parancsokkal lehet aktivizálni. A power mint legmagasabb prioritásu subtask müködik, amely tehát a lokál perifériákról érkező job-ok mellett a terminálóktól jövő job-okat is fogadni és kezelni tudja. Ez utóbbi job-ok az R22-be kerülve már nem lesznek megkülönböztetve a "lokál"-job-októl. Outputjuknak irányitását a PRT-POWER kártyán tesszük lehetővé a felhasználónak /külön output-osztályt rendelve a terminálokhoz/. Az F1 particióban müködő subtaskok /számuk bővithető ld. Fejlesztések/ egy-egy funkcionális kapcsolatot bonyolitanak le, a vezérlő task irányitása alatt. A subtask-2 a terminálon létrehozott forrás-booknak a DOSPW rendszer forráskönyvtárába való felvétetését oldja meg, mig a subtask-3 a forráskönyvtárban már meglévő book-ot olvas ki és irányit a terminál felé. Ezek a funkciók lebonyolithatóak lennének olyan jobokkal is, melyek a forráskönyvtárat az operációs rendszer alatt érik el, ezzel szemben az általunk választott megoldás előnyei:

- nem kerül be a Power input várakozási sorába, hanem a funkció azonnal, a helyi feldolgozással egyidőben kerül végrehajtásra;
- nem kerül be az output várakozási sorba, ezzel csökkentjük a lemezes átvitelek számát, területet és időt takaritunk meg;
- nem foglal le külön particiót;

- a forrásnyelvi book-ok mozgatása a vonalakon tömörített formában történik /tömörítést-lazitást az R10 végzi/;
- a felhasználónak nem kell job-ot összeállitani, a szolgáltatást egyetlen paranccsal kérheti;

Az eljáráskönyvtárat a DOSPW SLI funkciójával párhuzamosan, annak mintegy kiterjesztéseként hoztuk létre, uj Power diszpozició bevezetésével. Ez teszi lehetővé a job eljárások katalogizálását és egyszerü Power paranccsal való aktivizálását. Ilyen eljáráskönyvtár használat DOSPW operációs rendszer ismeretét nem feltételezi.

2. A megvalósitás részleteiről

Ebben a részben az IJS felhasználónak nyujtott lehetőségeit megvalósitó funkciókról kivánunk részletesebben beszámolni. Külön kiemeljük azokat a részeket, amelyeket önálló fejlesztésként valósitottunk meg és amelyeket a Power-ba való beiktatással értünk el.
Az IJS müködését a vezérlő task irányitja. Ő olvassa be külön vonalon a funkciókat azonositó parancsokat és értelmezi azokat és ütemezi a végrehajtó subtaskokat.
Kezeli a subtaskok müködéséhez szükséges belső táblázatokat, vezérli az operátor - rendszer kommunikációt.

2.1. Power parancsok az R10 felől

A vezérlő task felismeri, hogy a beolvasott parancs Power-operátori parancs. Elhelyezi azt a Power parancspufferében, felébreszti a Power operátor-communication-task-ot ugy, mintha a parancs a nagygép operátori konzoljáról érkezett volna. Jelzi továbbá, hogy a parancs a CCA felől érkezett és az esetleges válaszokat csap-

dázza, hogy visszaküldhesse azokat az RIO megfelelő terminájára.

Ugyanezt a csapdázó funkciót látja el akkor is, ha egy olyan jobnak szóló konzol üzenetet észlel, amely egy terminálról érkezett.

2.2. Terminálon létrehozott forrásnyelvü-book felvétetése a DOSPW felügyelete alatt álló SL-könyvtárba

Ezt a funkciót a SOUT paranccsal lehet aktivizálni, melynek paramétere csak a katalogizálandó book neve és verziószáma.

A subtask-2, miután őt a vezérlő task aktivizálta, a CCA SOURCEOUT vonalán tömöritett formába olvassa be a forrásnyelvü book-ot és egy parkoló file-ba helyezi. Ez a file egy DOS standard formátumu egyszemélyes forráskönyvtár, amely csak ezt az egy könyvet tartalmazza. Miután a teljes könyvet a subtask elhelyezte, egy könyvtári eljárás segitségével hozzá "merge-li" a DOSPW forráskönyvtárához. Az eljárás hivása az ujonnan megvalósitott Power-diszpozició felhasználásával történik.

Az uj diszpozició:

Leave - megfelel a Hold diszpoziciónak, de a job a végrehajtás után nem törlődik a job-queue-ból.

Az eljárás inditása egy szimulált release Power-paranccsal történik.

Az adatok védelmét ugy oldottuk meg, hogy nem engedélyezzük egy uj book beolvasását mindaddig, mig az előző katalogizálása be nem fejeződött.

2.3. A DOSPW felügyelete alatt álló SL könyvtárban már meglévő book lekérése terminálra

Ezt a prancsot a SIN paranccsal lehet aktivizálni, melynek paramétere a hivott book neve és a könyvtár azonositója /privát vagy rendszer/.

A subtask-3, miután őt a vezérlő task aktivizálta, az adott könyvtárból kikeresi a megfelelő book-ot és 160 karakteres blokkokban tömöritett formában a SOURCEIN vonalon továbbitja. Az adatok külön védelméről itt nem kell gondoskodni, mert a könyvtárakból csak olvasás történik. A task az egyidejüleg befutott kérelmeket sorbaállitja.

2.4. Interface problémák, megoldásuk

Az eredetileg kifejlesztett CCA berendezés csak TCU jellegü vezérlőegységek programozott emulálését tette volna lehetővé.

Az IJS rendszerhez a CCA-t ugy módositották, hogy tetszőleges I/O parancsokat és interrupt-okat átengedjen. Igy lehetővé vált, hogy tetszőleges periféria utasitáskészletét tudjuk emulálni, sőt akár IBM ill. ESZR prototipussal nem rendelkező berendezést definiáljunk és valósitsunk meg.

A parancsok felismerése és a megfelelő válaszok generálása az RIO-handler feladata /ld. IJS-RIO előadás/.

A periféria-emulálások megvalósitása során a legfőbb nehézséget az okozta, hogy a CCA nem tud olyan interruptokat generálni, amelyet nem egy előzőleg kiadott csatornaparancs végrehajtása eredményez.

Pl. a kártyaolvasó START-gombjának megnyomása egy "device-end" megszakitást okoz. A Power figyeli ezeket és innen tudja, hogy be kell olvasnia a következő job-ot a kártyaolvasóról.

Minthogy az IJS-ben is meg kellett valósitanunk azt, hogy az RIO-operátor jelezze kommunikálási szándékát, software uton kellett szimulálnunk hasonló jelenséget. Ezt ugy oldottuk meg, hogy külön választottuk azokat a vonalakat, amelyeken irás illetve olvasás történik és az olvasó vonalakra egy állandó read parancsot "lógattunk" ki. Az interrupt-ok állandó figyelésével pedig az IJS különválasztja a CCA-ról érkező megszakitásokat. Az interrupt-kezelő rutinba épitettük be a belövést segitő I/O-trace rutinunkat. Az interrupt kezelő rutinok supervisor statuszban müködnek.

3. <u>További fejlesztések</u>

A rendszer fejlesztése több fázisban történt. Egy nulladik fázisban a két gép CCA-n keresztüli összekapcsolását speciálisan erre a célra kifejlesztett önálló programokkal tesztelték.

Következő fázisként a job-beadás és output visszaküldés funkciót fejlesztettük ki. A job input/output vonalak ekkor kártyaolvasó és printer-berendezéseket emulálnak.

A harmadik fázisként került sor a vezérlő-task és a funkcionális subtaskok elkészitésére. Ebben a fázisban oldottuk meg a POWER-ban szükséges módositásokat.

További fejlesztésként tervezzük a terminálról történő disc-es file kreálás lehetővé tételét. Ezzel kompletté tennénk a felhasználó számára nyujtandó szolgáltatások körét.

Tervezzük még a SOUT funkció tisztán fizikai szintü kezeléssel való lebonyolitását /job-eljárás kiiktatását/ a rendszer hatékonyságának növelése céljából.

SZÁMITÓGÉPES FORGALOMIRÁNYITÓ RENDSZER

Flórencz András - Vető István
Számitógépalkalmazási Kutató Intézet

Bevezetés

A SzÁMKI-ban egy olyan számitógépes információs rendszer kifejlesztésével foglalkozunk, mely képes viszonylag sok, eltérő tipusú terminálból álló hálózaton keresztül nagy mennyiségü, folyamatosan beáramló információ feldolgozására. Az ilyen tipusú rendszer alkalmas lehet országos szállitási vállalatok szállitóeszközeinek és forgalmának folyamatos nyilvántartására és operativ irányitására. A rendszer R22 vagy R4o központi és Rlo hálózatvezérlő számitógépből, valamint olyan terminálhálózatból áll, amely az egész országra kiterjedhet.

Több ilyen, egyébként egymástól függetlenül müködő hálózatot egymáshoz lehet kapcsolni. Az igy létrejövő kapcsolatban az egyik hálózat központi gépe hozzáférhet a másik hálózat adatbázisához, valamint lehetséges a különböző hálózatokhoz tartozó terminálok közötti kommunikáció. Az ilyen tipusú rendszereknek két fő funkciót kell ellátniuk:

- Egyrészt nagy mennyiségü adat gyüjtését /például adatbázis feltöltését/

- Másrészt interaktiv kapcsolat megvalósitását /például az adatbázisban való lekérdezést/ a terminálok és a központi gép között.

A továbbiakban az általános célú rendszer egy adott megvalósitásával foglalkozunk. A rendszer felépitésében az Rlo hálózatvezérlő gép teljes software-ének, valamint a központi R4o gépnek az Rlo-zel való kapcsolatát biztositó software-nek a kidolgozása a feladatunk.

A terminálhálózat

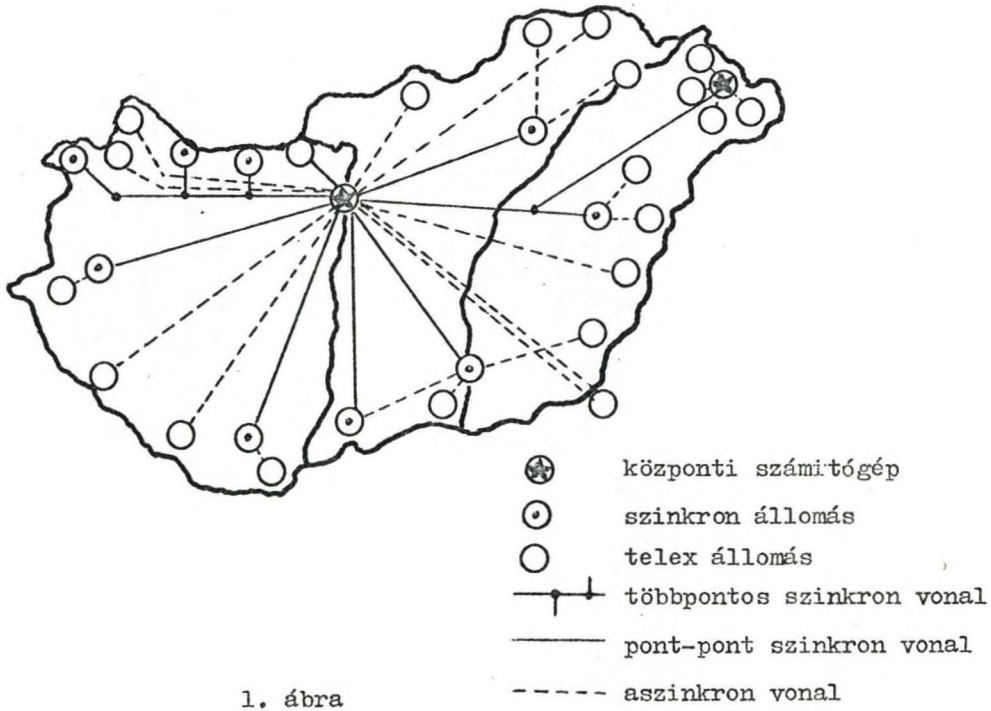
Az adatátviteli hálózat telexgépekből és speciális - a saját terminálfunkciói mellett három telexgépet is koncentráló - VTS 56loo szinkron terminálból áll. Ezenkivül szinkron vonalon csatlakozik a rendszerhez néhány távoli számitógép is. Az Rlo a közvetlen /pont-pont/ csatlakozó terminálokon kivül /egy privát/ kapcsolt telex hálózatra is rákapcsolódik. Az Rlo a szinkron vonalakon a BSC multipoint algoritmus vezérlő /master/ funkcióit valósitja meg, de szinkron vonalon csatlakoztatott távoli számitógép felé képes az alárendelt /tributary/ állomás szerepét is játszani.

A hálózat bármelyik terminálja - a terminál operátorának döntésétől függően - kétféle üzemmódban dolgozhat:

- adatgyüjtő üzemmódban a terminál egy külső adathordozón /általában lyukszalag, de szinkron terminál esetén lehet mágneskazetta is/ rögzitett üzenetet /a továbbiakban táviratot/ küld a számitógépnek, vagy a már korábban elküldött táviratra vonatkozó hibalistát, ill. nyugtázó táviratot fogad a számitógép felöl. Erre az üzemmódra a viszonylag hosszú üzenetek cseréje a jellemző.

- interaktiv üzemmódban a terminál az R4o központi gép interaktiv szolgáltatásait veszi igénybe /pl: lekérdezés egy adatbázisból, stb./. Erre az üzemmódra az Rlo és a terminál közötti rövid üzenetek gyors cseréje a jellemző.

A terminál operátora - rövid operátori üzenetváltás során - bármelyik üzemmódot kezdeményezheti. Az egyes terminálok egyidőben különböző üzemmódban müködhetnek, egymástól függetlenül.


központi számítógép
szinkron állomás
telex állomás
többpontos szinkron vonal
pont-pont szinkron vonal
aszinkron vonal

1. ábra

A hálózatvezérlő számitógép funkciói

A hálózatvezérlő számitógép egy maximális tárkiépitésü Rlo, amely az adatátviteli vonalakkal COS adatátviteli multiplexeren és TTH automatikus hivóegységen kersztül, az R4o központi géppel pedig CCA csatorna-csatorna adapteren keresztül tartja a kapcsolatot.

Az Rlo feladata - a terminálok üzemmódjainak megfelelően kettős:

- az adatgyüjtő üzemmódban dolgozó terminálokról küldött táviratokat az Rlo diszkes file-ba teszi, konvertálja és formai szempontok szerint ellenőrzi. A hibás táviratokról hibalistát készit, a hibátlanokra nyugtát generál és ezeket választáviratként a diszkre teszi, majd elküldi a terminálra. A távirat fejében érkezett információktól függően az ellenőrzött táviratot az R4o-felé továbbitja, vagy egy másik terminálnak cimzett üzenetként diszkre helyezi. Ily módon az Rlo üzenetkapcsoló funkciókat is ellát.

  A hálózatvezérlő a központi gép felől képes táviratokat fogadni, amelyeket vagy egy meghatározott terminálra kell eljuttatni /válasz adattávirat, lista, stb./, vagy valamennyi kijelölt terminálra el kell küldeni /körözvény/. Ezek a táviratok is diszkre kerülnek. A diszkterület dinamikus kihasználásáról és a diszken tartózkodó táviratok adminisztrálásáról egy speciális file-kezelő gondoskodik.

- az interaktiv üzemmódban dolgozó terminálok felé az Rlo legfontosabb feladata a gyors üzenetközvetités a terminál és az R4o között, ezért az interaktiv üzenetek nem kerülnek diszkre. Az adott memória és az Rlo-R4o közötti logikai kapcsolatok száma miatt a teljes hálózatban egyidőben max. 14 terminál müködhet interaktiv üzemmódban.

A fentieken kivül az Rlo-nek a hálózat vezérlésével kapcsolatban sok segédfunkciót kell ellátnia:

- a terminálok adminisztrálása. Egy diszkes file-ban helyezkedik el a hálózat-könyvtár. Ez valamennyi olyan terminál, eszköz és számitógép adatait tartalmazza, amelyek a hálózaton keresztül kapcsolódhatnak a rendszerhez. A hálózathoz max. looo terminál tartozhat, természetesen egyidőben ezeknek csak kis része - kiépitéstől függően 4o-6o terminál - lehet fizikailag aktiv.

- statisztikák készitése. A rendszer terminálonként gyüjt statisztikai adatokat a vonali hibákról és a forgalmi viszonyokról.

- a rendszer biztonságos üzemelésével kapcsolatos feladatok. Az adatrendszer mentése, ill. újrainditása a CCA-n keresztül az R4o segitségével.

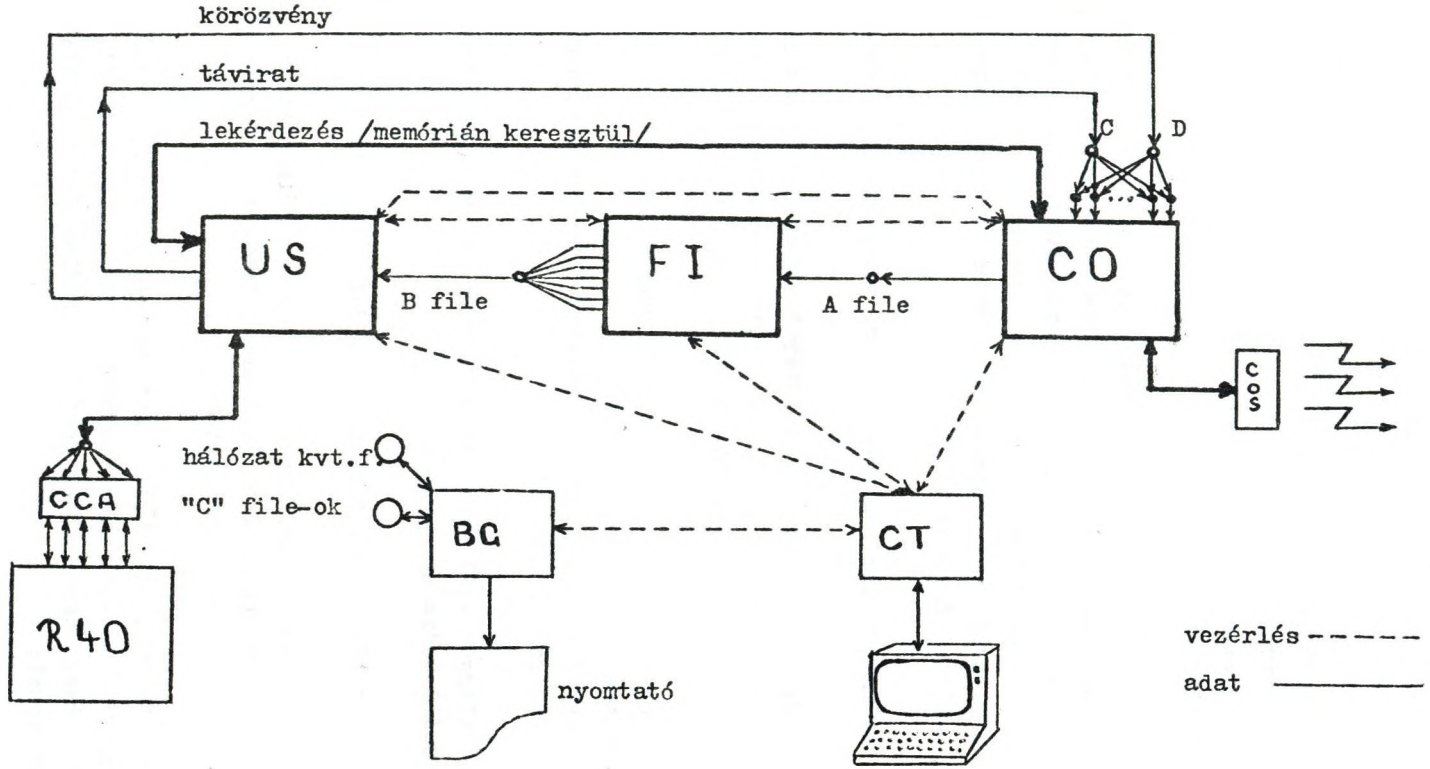
A központi gép funkciói

Az R4o központi gép feladata a hálózatvezérlő számitógéppel való kapcsolattartáson túlmenően, az üzemmódoknak megfelelően szintén kettős:

- fogadja az Rlo által összegyüjtött, ellenőrzött táviratokat, mágnesszalagra naplózza és feldolgozza azokat, - például egy adatbázis felújitását végzi a táviratok tartalma alapján - adattáviratokat, listákat, körözvényeket generál és továbbit az Rlo-be, ahonnan azok a terminálokra kerülnek.
- fogadja a terminálról beérkező interaktiv igényeket és a megfelelő taskokat lefuttatva kielégiti azokat /pl. lekérdezés egy adatbázisban/.

A hálózatvezérlő gép software rendszere

A hálózatvezérlő Rlo-ben egy speciálisan átalakitott real-time diszkes monitor felügyelete alatt a rendszer öt szinten /o-4-ig./ dolgozik. Az 1-4. megszakitási szinteken rezidens alrendszerek, a 0. szinten az alrendszerek igényeit kiszolgáló, dinamikusan betöltött segédprogramok futnak. /2. ábra/


körözvény
távirat
lekérdezés /memórián keresztül/
C
D
US
FI
CO
B file
A file
COS
hálózat kvt.f.
"C" file-ok
CCA
R40
BC
CT
nyomtató
vezérlés
adat

2. ábra

- A legmagasabb software szinten /4./ dolgozik az ún. koncentrátor /CO/ alrendszer. Feladata:
  - A vonali forgalom vezérlése a terminálok üzemmódjának, munkafázisának irányitása, üzenetváltás a terminál operátorával /"operátori algoritmus"/. A memória jobb kihasználása érdekében dinamikusan gazdálkodik a vonali pufferekkel.

  - Az adatgyüjtő fázisban müködő terminálokról vett táviratok diszkre irása. Az adatvesztés elkerülése és az áteresztőképesség növelése miatt valamennyit egyetlen közös, direkt elérésü file-ba /A-file/. A terminálokra kiküldendő táviaratokat a C-tipusú, a körözvényeket a D-tipusú file-okból olvassa.

  - Az interaktiv fázisban dolgozó terminálokról érkező

  - rövid - üzeneteket a pufferrel együtt az US alrendszernek adja át. Az Rlo válaszait ugyanezekben a pufferekben fogadja az US-től és továbbitja a terminálra.

- A 3. megszakitási szinten fut az üzeneteket válogató FI alrendszer. Feladata:
  - az egyidőben érkezett táviratokat keverten tartalmazó A-file szétválogatása;

  - a telexgépekről érkezett forditott telex kódú szöveg EBCDIC kódra konvertálása;

  - minden táviratból önálló file /B-file/ készitése.

- A 2. szinten dolgozik az US alrendszer. Feladata:
  - a B-tipusú file-okból kiolvasott táviratot elemzi, ellenőrzi és az R4o-re továbbitja;

  - nyugta, ill. hibaüzeneteket generál és ezeket, valamint az R4o-től érkezett táviratokat, listákat, ill. a cimzett terminálnak megfelelő kódban C-tipusú fileba teszi;

- a hosszú ideje - pl. vonalhiba,terminálfoglaltság miatt - diszken tartozkodó táviratokat kerülő útra irányitja;

- közvetiti az interaktiv üzeneteket a CO alrendszer és az R4o között;

- megvalósitja az üzenetkapcsoló funkciókat.

A táviratforgalomra a CCA 16 logikailag független alcsatornája közül kettőt vesz igénybe /egy input és egy output/. A fennmaradó 14 alcsatorna az interaktiv üzemmódban dolgozó terminálok rendelkezésére áll.

- Az 1. megszakitási szinten dolgozik a CT vezérlő alrendszer. Feladata:
  - fogadja konzolról beadott operátori parancsokat és az egyes alrendszerek igényeit;

  - az igényeknek megfelelően betölti és ütemezi a background-ban futó segédprogramokat;

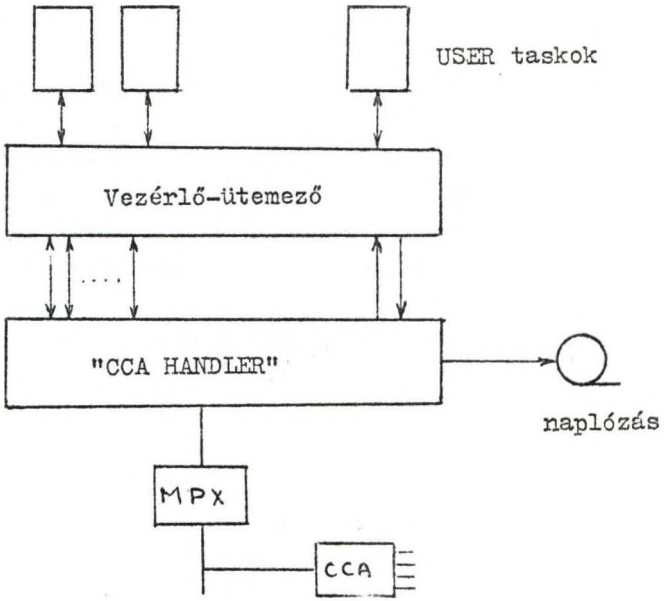
  - megteremti és vezérli a kapcsolatot a segédprogramok és az alrendszerek között.

- A 0. szinten a background zónában futnak az egyes alrendszereknek szóló parancsokat feldolgozó, az alrendszerek konzolüzeneteit kiiró, a hálózatkönyvtárat kezelő, a rendszert inicializáló és a mentést/újrainditást végző segédprogramok.

A központi gép software rendszere

Az R4o tipusú, 1 Mbyte tárkapacitású központi gép OS vezérlete alatt müködik. A multiplexer csatornára csatlakozó CCA-val egy "CCA handler" tartja a kapcsolatot. /3. ábra/. Az interaktiv alcsatornákon érkezett üzeneteket egy vezérlő-ütemező task-nak adja át, amely az üzenetek értelmezése után inditja a megfelelő felhasználói task-okat. A felhasználói task válaszát /pl. egy lekérdezési fázis eredményét/

a "CCA handleren" keresztül továbbítja az Rlo-be. A felhasználói task-ok és a vezérlő között a kommunikáció logikai szinten /READ, WRITE, stb. jellegü makrók formájában/ történik.



3.ábra

A file-input alcsatornán érkező táviratokat a "CCA-handler" mágnesszalagra naplózza, majd a vezérlőn keresztül a táviratokat feldolgozó felhasználói taskoknak adja át. Ezek és bármelyik felhasználói task generálhat válasz-, lista-, körözvény-, stb. táviratokat, amelyek a vezérlőnek átadva, a "CCA-handleren" keresztül jutnak el az Rlo-be.

Forgalmi viszonyok, adatbiztonság

A fentiekben ismertetett rendszer dinamikusan alkalmazkodik a forgalmi viszonyokhoz. Ha nagy a vonali forgalom, feldolgozó funkcióit későbbre halasztja, és csak adatgyüjtést végez. Az Rlo rövid ideig /25 perc/ tartó csúcsterhelés esetén másodpercenként 6o rekordot /kb. 3ooo karakter/ képes fogadni és

diszkre tenni. A rendszer átlagos áteresztőképessége folyamatos terhelés esetén kb. 1500-2000 karakter/sec.

Ha a rendszernek napi 24 órán keresztül, folyamatosan kell üzemelnie, két azonos R10-R40 konfigurációt célszerü üzembehelyezni. A megfelelő hardware eszközök hiányában azonban a melegtartalékolás és az adatok duplikálása nem oldható meg. A kézi átkapcsolás és az adatrendszer feltöltése, újrainditása várhatóan 20-30 perces kiesést okozhat.

Az előadás a rendszer néhány érdekesebb részletét /pl. dinamikus diszk kezelés/ és a fejlesztés tanulságait fogja tárgyalni.

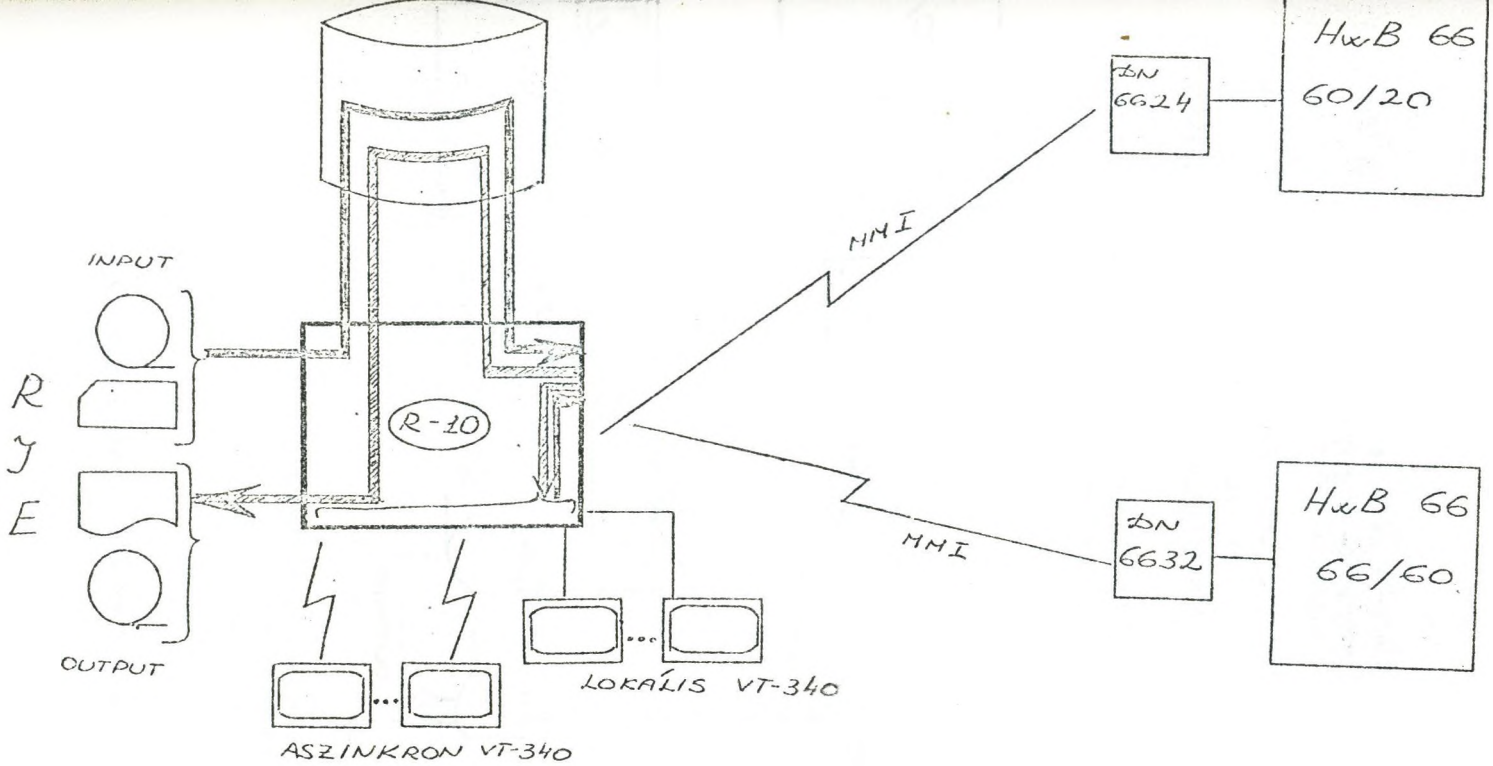
INTELLIGENS TERMINÁLOK ÉS KONCENTRÁTOROK

Földvári I. - Laborczi Z. - Mink Jné - Rajki P.
Számitógépalkalmazási Kutató Intézet

A SzÁMKI-ban 1974 óta foglalkozunk az Rlo számitógép adatátviteli alkalmazásaival. A fejlesztéseinket az Rlo számitógép operációs rendszerében lévő real-time diszkes monitorokra és az általunk létrehozott adatátviteli monitorkiegészitésre és dinamikus diszk-kezelést biztositó modulokra épitettük.
Jelen előadásban ezek a fejlesztések közül a legújabbakat kivánjuk ismertetni.

HONEYWELL intelligens koncentrátor

Az Rlo környezetében telepitett terminálok felhasználói számára biztositja a Honeywell time-sharing és más interaktiv rendszereinek elérését, valamint lehetőséget ad remote-jobok forgalmazásához. Ezzel egyidejüleg korlátozott méretü helyi batch feldolgozást is folytathat. Ez a fejlesztés révén az ÁSzSz nagy kapacitású számitóközpontja távoli felhasználói kényelmesen és gazdaságosan tudják kihasználni a központi gép által nyújtott szolgáltatásokat.



TIME-SHARING

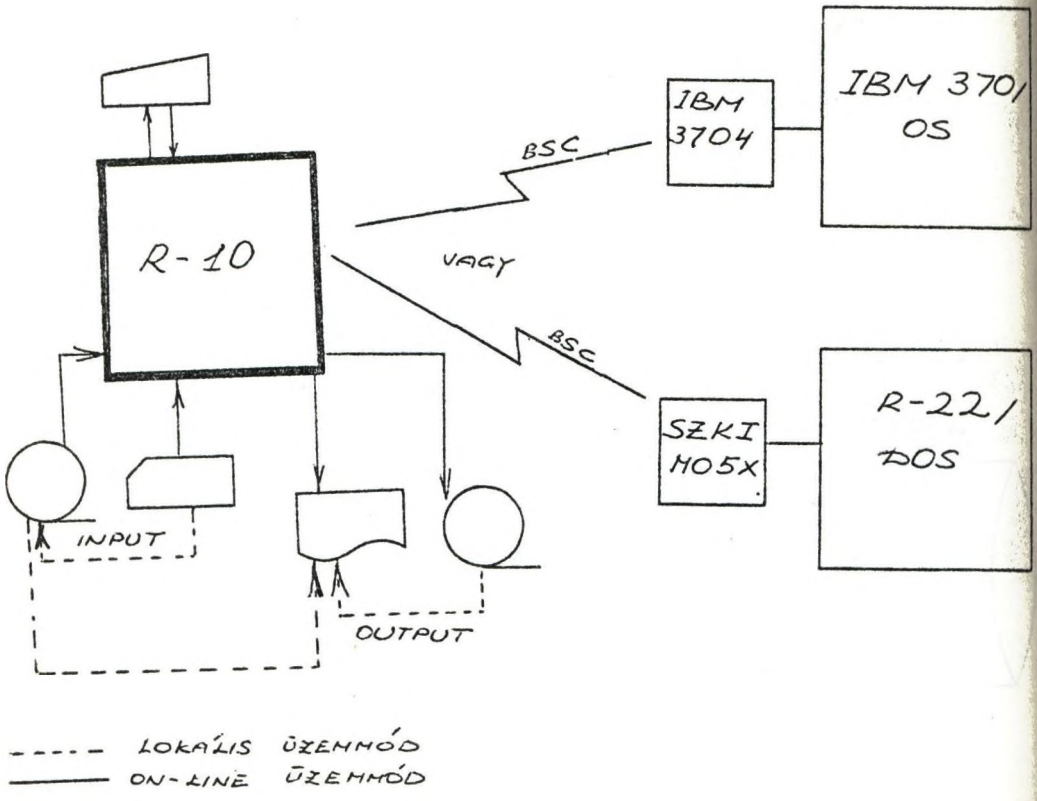
HONEYWELL INTELLIGENS KONCENTRÁTOR

SzÜV-RJE-állomás

Ez a fejlesztésünk a SzÜV részére készült RJE /Remote Job Entry/ állomás az IBM/OS és ESzR/DOS rendszerekhez.

1. Vonali üzemmódban vagy mint operátori konzol támogatással kiegészitett IBM 277o programozói terminál, vagy mint mágnesszalag-orientált adatforgalmazó terminál müködtethető.

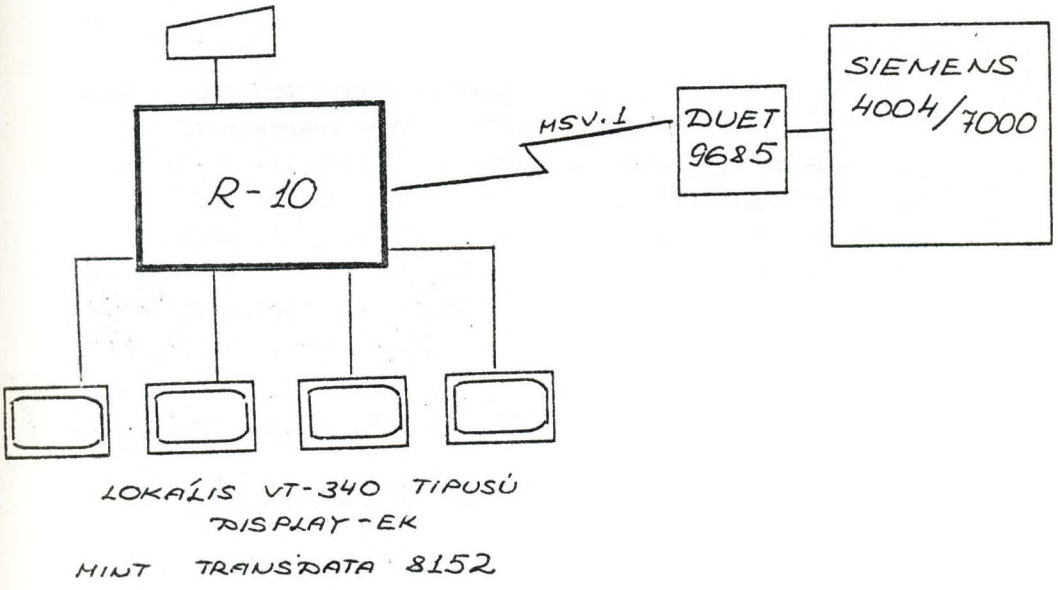
2. Az Rlo lokális üzemében használható géptermi programcsomag az adathordozók konvertálását és IBM/ESzR követelményeknek megfelelő kezelését biztositja.



SZÜV - RJE - ÁLLOMÁS

SIEMENS terminál-koncentrátor

Az Rlo konfigurációba tartozó minden egyes VT34o tipusú display-t önálló SIEMENS TRASDATA 8152 terminálként képes müködtetni a Siemens 4oo4/7ooo gépcsalád tagjai felé, ezáltal biztositva az interaktiv hozzáférést a központi gép által nyújtott lehetőségekhez.



SIEMENS TERMINÁL-KONCENTRÁTOR

Uj módszer alkalmazása

Az utóbbi program tervezése és kódolása során új módszer került kipróbálásra. A SIMULA 67 processz /folyamat/ fogalmát a lényegi tulajdonságok megtartása mellett módositottuk, és egyszerüsitettük, s ezzel kettős célt értünk el: kitünő eszközt kaptunk: real-time programok irására, s az assembly nyelven történő megvalósitás sem ütközött különösebb nehézségekbe. Az eredmény lényege, hogy a programot szekvenciális szervezésü, együttmüködő kvázipárhuzamos folyamatokra lehetett felbontani.
Ez a felbontás a program adatszerkezeteinek strukturálását és elérési algoritmusok mögé rejtését is nagyban elősegitette. Tág lehetőségek nyiltak a folyamatok közötti információcsere megvalósitásánál.
A további kutatás arra fog irányulni, hogy a sikerrel kipróbált eszközt alkalmas magasszintü nyelvbe beágyazzuk.

AZ IJS INTERAKTIV JOBKEZELŐ RENDSZER R10 OLDALI FEJLESZTÉSI EREDMÉNYEI

Földvári Iván - Mandler György
Számitógépalkalmazási Kutató Intézet

Az előadás a SzÁMKI-ban kifejlesztett IJS rendszernek az R1o software fejlesztési szempontjaival foglalkozik elsősorban.

ESzR-gépek interaktiv elérése

A rendszer tervezésénél fontos cél volt, hogy az alapvetően batch orientált R-22 /ill. R-2o-tól fölfelé/ gépeket interaktiv elérési lehetőséggel lássuk el. Másik tervezési szempont az volt, hogy a számitógépek összekapcsolásuk ideje alatt is maradjanak önállóak. Ezt a követelményt az R-22 esetén teljesen, az R1o esetén részben meg tudtuk tartani.

Az R1o-es software rendszer változatai

A terminálok száma szerinti felosztás

A rendszernek elkészült az egydisplay-es változata, a több /maximum 16/ displayt kezelő változat fejlesztés alatt van. Mindkét változatban alapvetően kétfajta szolgáltatást nyújt a rendszer a felhasználónak.

A szolgáltatások szerinti felosztás

A. Általános kártyaolvasó-sornyomtató szimuláció

Az egyik /a/ szolgáltatás segitségével a felhasználó az R1o diszkjén tárolt job-okat küldhet át az R-22-nek, az R-22 végrehajtja azokat és a keletkezett listákat a felhasználó visszakérheti az R1o-re. Megjegyezzük, hogy a listák nem közvetlenül sornyomtatóra kerülnek, hanem diszkre, úgynevezett "szöveg" file-ba. A listát a felhasználó display-én "megnézheti", részben vagy egészében ki-

printelheti. A lista alapján kijavithatja /szintén display-ről/ a diszken tárolt job-ot és ismét elküldheti az R-22-nek végrehajtásra.

Az /a/ szolgáltatás az általunk kifejlesztett JOBIN, LISTOUT és LISTPRINT; valamint az alkalmazott Interaktiv Diszk Operációs Rendszer /IDOS/ EDITOR programja segitségével vehető igénybe.

B. Speciális szolgáltatások az R22-ben lévő software segitségével.

A másik /b/ szolgáltatás az R-22 diszkjén tárolt forráskönyvtári file-ok Rlo segitségével történő javitását biztositja. A javitás menete a következő: A SOURCEOUT program segitségével az R-22 forráskönyvtárából át kell kérni a javitandó file-t az Rlo könyvtárába. EDITOR-ral történő kijavitása után pedig a SOURCEIN programmal vissza kell juttatni a kijavitott forrásfile-t az R-22 könyvtárába.

A rendszert általában az R-22 felhasználók használják, számukra az Rlo csupán az interaktiv hozzáférés eszköze.

Az egy display-es változat lehetővé teszi azonban, hogy Rlo felhasználók anyagjaikat /programok, adatok/ az R-22 diszkjein tárolják, mintegy mágnesszalag helyett használják az R-22 háttértárolóit.

Igen érdekes és értékes szolgáltatás lenne /amihez a két gépet összekötő hardware lehetőséget is biztosit/ az R-22 filejaihoz Rlo-ről történő hozzáférés /input, output, update/ megvalósitása, mely segitségével különféle adatbázis kezelő rendszereket lehetne kialakitani.

Megemlitjük továbbá, hogy az A. tipusú szolgáltatás nem igényel semmilyen speciális software-t az ESzR-nagygépes oldalról és azt bármilyen hagyományos batch rendszerben alkalmazni lehet. Az ESzR gépen futó munkák előfeldolgozása - szintaktikus analizise, adatfile-jainak előállitása, stb. - programozott módon folyhatna.

A rendszer működéséhez szükséges eszközök

Hardware:

Rlo központi egység 32K szó tárral. /1 display esetén 16K is elég/

diszk minimum 8ooK byte, /lehet DRI vagy IZOT diszk is/

VT-34o-es display/ek/ multiplexeren

sornyomtató

Chanel-chanel adapter /CCA/

Software:

A SzTAKI-ban kifejlesztett IDOS /Interactiv Disk Operating System/.
A SzTAKI-ban kifejlesztett /több display szimultál kezelését biztositó/ DELFIN rendszer.

A SzÁMKI által elkészitett CCA handler és az adatforgalmazást biztositó szolgáltató programok.

RENDSZER HANGOLÁS TUDOMÁNYOS KÖRNYEZETBEN

Gaál Tamás
MTA Központi Fizikai Kutató Intézet

1. Bevezetés

1.1 Rendszer-hangolás

"Rendszer" néven a továbbiakban nagy számítógépeket és operációs rendszerüket együtt tekintjük. A hangolás a meglévő hardware és software lehetőségek optimális kihasználása a környezetben. Ehhez célt kell kitüzni. Megkülönböztetendő a hangolás a tervezéstől, a hangolás jóval szükebb. Igy egy OS rendszer először generálása tervezés, a továbbiak inkább hangolások. /Az ESZR alapfogalmakat ismertnek tételezzük fel./

A KFKI-ban 1977 nyarán intézeti nagygépként R40 tipusu számitógépet állitottak üzembe. /1 Mbyte operativ tár, 1 multiplexor, 5 szelektor csatorna, 12 db 7,25 Mbyte-os mágneslemez egység, 3 sornyomtató, lyukkártya és lyukszalag olvasó/lyukasztó perifériák és 2 konzol irógép./ Kezdetben ESZR OS MVT 4.0 operációs rendszerrel működött, az induló rendszert a KFKI és a gyártó közösen tervezte. Általános célu számitógépként használják, programok fejlesztésére és futtatására. Több részleges /I/O/ generálás után IBM 21.8 OS MVT rendszert is generáltunk, ezt ennek bizonyos előnyei indokolják.

A cél, hogy a felhasználók igényeit az operátorok véleményét is figyelembe véve, kielégitsük /az operátori munka itt viszonylag bonyolult, és nagy a befolyása a hatékonyságra/ ugy, hogy az eredmény a különböző mértékek /job-mix-ek, bench mark-ok, job/nap stb/ szerint a rendszerprogramozó számára is megfelelő legyen [1], [2], [3], [4]. Kivánatos a gyors job-fordulás, a kényelmes használat, hibajavitási és figyelési lehetőség. A hangolás apró lépések sorozatából áll. A hatékonyságot egy-egy hangolás gyakran csak 1 %-kal növeli [1], ezért

/is/ sok "tuning" szükséges. Természetesen a folyamat visszacsatolt, érdemes a változás hatását tesztelni és a változtatási lehetőségeket meghagyni /ld később/.

1.2 Tudományos környezet

Jellemzői a sok FORTRAN program, sok importált program, ezért sok, esetleges ismeretlen szerkezetü, irássürüségü mágnesszalag, különleges adathordozók, kódok, szervezések, sok a bonyolult szerkezetü, esetleg más géphez illesztett nagy programrendszer, sok a magán-fejlesztés, saját eljárásokkal, könyvtárakkal, a sornyomtatónál jobb rajzoló berendezés kivánatos /CAD felhasználók/, CPU igényes job-ok is vannak, és végül, sok a felhasználó.

2. A megvalósitott hangolási lépések

Egy rendszer próbaüzeme során első cél a szük keresztmetszetek megállapitása. Leggyakrabban /mint nálunk is/ ez az I/O tevékenység. Kevés a mágneslemez kapacitás, és a mágneslemezek gyakran elromlanak - müködésük bizonytalan. A lehetséges hangolási lépéseket többféleképp csoportosithatjuk, az OS MVT rendszer igen sok lehetőséget nyujt, saját fejlesztések is lehetnek, hangolhatjuk a hardware-t a software-en keresztül, és a felhasználók irányitása fejlettebb programozási technológiák, nyelvek felé is hangolás.

2.1 A hardware hangolása software eszközökkel

A rendszer generálásánál van a legtöbb lehetőség, később is helyrehozhatók az esetleges hibák, részleges /I/O/ generálással. A perifériák a csatornákon keresztül érhetők el. A csatornák meghibásodái esélye is viszonylag nagy, ezért érdemes egy eszközt több csatornához csatolni. Ezt a megfelelő rendszergeneráló makroban jelezni is kell [9].

Növeltük az elérési utak számát, másik követendő elv a mágneslemezek között a tevékenység megosztása [5], [1], [3]. Az eljárás itt az [9] , hogy az azonos tipusu mágneslemezekből csoportokat képzünk, a felhasználók e csoportokra hivatkozhatnak, a lemezhasználat célja szerint /pl. "USER", "WORK", "SYSDA", "SYSSQ"/. Ez egyszerü lehetőség lenne a rendszer lemezek védelmére is /gépi hiba következtében való leállásnál egyes munka adatállományok nem törlődnek, igy feleslegesen csökkentik a továbbiakban a lemez kapacitását/. Sajnos az elv itt szembetalálkozik a gyakorlattal: kijelöltük ugyan a csoportneveket, de az egységek gyakori meghibásodása folyán előfordul, hogy egy-egy csoport teljesen kiesett, igy a legjobb a tipusnévvel /pl. 2311/ való hivatkozás.

A fejmozgások átlagtávolságának, igy az elérési időnek a csökkentésére érdemes megtervezni minden egyes köteten a kötet tartalomjegyzék /VTOC/ helyét. Egyszerü esetben /szimmetrikus felépitésü kötetnél/ ez középen /100. cilinder/ van.

Két konzol irógépünk van: lehetséges fő és másodlagos konzolként való használatuk /MCS/, ez javasolt is, az operátori munkát meggyorsitja. További lehetőség "összetett konzol" létrehozása: lyukkártya olvasóból és sornyomtatóból áll. Ez a kezdeti programbetöltésnél /NIP/ hasznos, utána áttérhetünk a valóságos konzolra, felszabaditva az összetett konzol alkotórészeit [9], [11].

2.2 A software hangolása

Itt a legfontosabb a rendszer lehetőségeinek megismerése és kihasználása. A programozói munka hatékonyságának javitására érdemes magaszintü, célorientált gépi nyelvek bevezetni. Szükség lehet még speciális programozói vagy operátori igények kielégitésére, saját fejlesztésü rendszerprogramokra; az OS rendszer általában még ezek illesztéséről is gondoskodik, vagy azt lehetővé teszi.

2.2.1 Az OS MVT rendszerbe beépitett lehetőségek

Mint jeleztük, ez a legfontosabb,[5], [6], [7], [9] irja le. Felsoroljuk az általunk megvalósitott lehetőségeket és hasznukat.

- A rendszer működésének megismeréséhez információt kell gyüjteni róla. Erre való az SMF funkció [7]. Ez a job-ok és a rendszer különböző attributumairól feljegyzéseket készit és beirja egy e célra szolgáló adatállományba. Innen kimenthető, másolható, feldolgozható. Segitségével eloszlásokat, hisztogramokat készithetünk, például a job-ok CPU-idő vagy perifériához fordulási igényéről, a felhasznált adatállományok számáról job-onként, az adatállományokra való hivatkozásokról, stb. A job-számlázó programok is innen nyerik /általában/ a statisztikákat, melyekből sulyozó tényezők segitségével /ezek szintén az SMF alapján jöttek létre/ a számla megállapitható. Érdemes szükiteni a gyüjtés körét: felesleges az ideiglenes adatállományokról tudni, ki, hol, mikor létesitette őket, általában az állandó adatállományok sem felügyelendők. Az adatgyüjtés időigényes, ez indokolja a szükitést.

Az SMF tehát mérőeszköz. Paraméterei a SYS1.PARMLIB nevü particionált adatállomány SMFDEFLT nevü tagjában találhatók és felülbirálhatók, NIP-nél is változtathatjuk [11], [12].

- SMF kilépési rutinok: különböző ki/belépési pontokon csatlakoznak a supervisorhoz, módosithatják a job-ok futását. Mi JCL /job vezérlő nyelv/ ellenőrző rutint irtunk, ez az osztályba sorolt job paramétereit /főtárigány, prioritás, időigény/ ellenőrzi, és szükség esetén megváltoztatja [7].

- MCS /konzol/ kilépési rutinok: érdemes egy job-ra vonatkozó lényeges operátori üzenetet, mely egyébként a konzolon jelenne meg, a programozóval is tudatni [6].

- Job szeparátor: a sornyomtatón elkülöniti a job-okat, kiirja nevüket, a kezdeti, befejezési időt, a felhasznált CPU időt, a napi dátumot stb.

- PRESRES lista: ez a SYSI.PARMLIB adatállomány egy tagja, a nevük szerint benne felsorolt mágneslemez kötetek definiciójuktól függően állandóan vagy időlegesen lefoglalják a mágneslemez egységet, és használhatja őket bárki, aki mágneslemezt kiván /pl. ideiglenes adatállománynak/ vagy csak aki névszerint hivatkozik rájuk [5], [6] . Mi valamennyi rendszerkötetet és magánkötetet felvettük erre a listára, haszna kettős: csökkenti az operátori munkát /a MOUNT parancs ritkábban kell/, a rendszeridőt , és védi a mágneslemezeket az illetéktelen ráírástól.

- Rezidens rutinok: sok rendszer-rutin mágneslemezen van, használatuk perifériához fordulást és a mágneslemezen keresést igényel.

- A BLDL-lista segitségével /SYSI.PARMLIB, ld. [6]/ felvehető egy táblázatba, mely a mágneslemezen lévő fizikai címeket tartalmazza, igy a rutin hivásakor nem kell a VTOC-hez fordulni. Több lista lehet, NIP-nél kiválasztható.

- Teljesen rezidenssé tehető a főtár LPA vagy SQA nevü területein. Nálunk a FORTRAN futtató rutinok mind rezidensek [6][10].

- A GFT /általános nyomkövető/ lehetőség szolgál a leggyakrabban hivott SVC rutinok figyelésére [8], [9], [10]. Ezeket kiválasztva, és rezidenssé téve, nő a hatékonyság [5].

- Az iniciátor /initiator/terminator/ és a beolvasó /reader/interpreter/ egyes moduljainak [6] rezidenssé tételével növeltük a főtárban rendelkezésre álló helyet: ezek ezentul közös részek, több iniciátor vagy beolvasó számára, igy pl minden beolvasó az eddiginél kb 30 Kbyte-tal kevesebbet igényel.

- Megfelelő iniciátor inditásával kézben tarthatjuk a job-ok versengését: határoló és kötelező prioritások definiálhatók, job-osztályok szerint. Az osztályok helyes megválasztásával növelhető a hetékonyság /ld. 2.3/.

A leggyakoribb prioritási szintekre időszeletelést /time-slice/ alkalmaztunk: az időszelet elteltével a sor következő, ugyanolyan prioritásu job-ja kap vezérlést. Erre a sok CPU igényes job miatt volt szükség [6], [9], [10], [11].

- Futás közbeni dinamikus tárigénylési lehetőség a "Roll in/Roll out" JCL nyelven kérheti a felhasználó a programja területének növelését, ha a főtárban a közvetlenül felette lévő job kigördíthető, akkor azt szükség esetén átmenetileg mágneslemezre teszi. Hátránya, hogy csak ugyanoda kerülhet vissza. Használatához a rendszer NIP-kor a rezidens diszken területet jelöl ki, ezért ott ehhez elegendő szabad helyről kellett gondoskodnunk [5], [9], [12].

- Betölthető formáju /load module/ programokat tartalmazó könyvtárak láncolhatók a SYS1.PARMLIB könyvtár LNKLIST nevü tagjában a SYS1.LINKLIB-hez. A programot hivásakor először az első, majd a második stb könyvtárból próbálja a programbetöltő a főtárba vinni. Mi a saját fejlesztésü rendszerprogramok /operátori és felhasználói segédprogramok/ könyvtárát láncoltuk [6].

- A beolvasó eljárás megfelelő paraméterezésén sok mulik, itt dönthető el az idő, terület stb alapértelmezése, saját rendszer vagy felhasználói eljáráskönyvtár láncolható a SYS1.PROCLIB nevü OS eljáráskönyvtárhoz [6].

- A beolvasás gyorsitására szolgál az ASB reader nevü beolvasó: ez nem kártyánként, hanem - CCW láncolással - 4o kártyánként olvas be, és csak 1o beolvasott job után kerülnek a SYS1.JOBQUE-ba a job-ok. Ezt is kipróbáltuk, azonban tul sok lemezterületet igényelne a job-ok ideiglenes tárolásához, igy nem használjuk. A multiplexor csatorna jobb kihasználását segitené elő.

- A legfontosabb,folyamatos ellenőrzést kivánó lépés a job--queue /SYS1.JOBQUE/ adatállomány méretének és jellemzőinek meghatározása. Ezen mulik, hány job futhat, egy job milyen bonyolultságu lehet, egyszerübben tehát a multiprogramozás hatékonysága. Rendszergeneráláskor határozandó meg, de NIP-nél felülirható. Nálunk a JOBQFMT=15, a JOBQLMT=105 /ez a szám arányos a job erőforrás szintü bonyolultságával/. A job queue tulajdonságai határozzák meg DD kártyák job-on belüli maximális számát is, különleges igényeknél tehát

uj NIP segitségével lefuttatható olyan job is, amelynél a szokásos job queue már betelne. Az értékek az SMF statisztikáktól függően változtatandók, a job queue igény várható értéke és szórása szerint [5], [6], [9], [10], [12], [7].

- A dinamikus eszközáthelyezés /DDR/ lehetőségét az IBM OS/MVT 21.8 rendszer hozta meg, érdemes kihasználni a mágneslemez egységek bizonytalansága miatt: az operátor és az I/O supervisor egyaránt kérheti egy lemezcsomag valóságos vagy névleges kicserélését, a rezidens kötet számára is kijelölhető felváltó egység [9], [5], [11].

- Nyomkövetési célt szolgál a "Trace" nyomkövető táblázat, ahol feljegyződnek a megszakitások. Generáláskor 50 mélységü táblát létesitettünk [5], [9].

- Az INITD eljárás hozzárendelt /dedicated/ adatállományok segitségével gyorsitja az ideiglenes adatállományok használatát. Érdemes a gyakoribb eljárások /nálunk FORTRAN fordítás, szerkesztés, futtatás, könyvtárkezelő segédprogramok stb/ számára megvalósitani [6], [5].

2.2.2 Saját fejlesztések

Ezekre a különleges igényeknél volt szükség.

- A mágnesszalag vizsgáló segédprogram tetszőleges adatszerkezetü mágnesszalagról statisztikát készit, ezzel ismeretlen adatszerkezetü szalagok vizsgálhatók. Több adatállományos, cimkézett, cimkézetlen és adathibás szalagot is feldolgoz [13].

- Operátori parancsként konzolirógépről indítható mágnesszalag cimke olvasó program, a kötet cimke elolvasására szolgál, és ezt /vagy a rendellenes körülményt, pl. tape mark vagy adathiba/ a konzolra kiirja. Ez is a rendszer lehetőségeire támaszkodik, a SYS1.PROCLIB nevü adatállományban lévő eljárás START paranccsal indítható [11].

- A papirszalagos adatgyüjtés sajnos /a mérőberendezések nem elég korszerüek/ még gyakori. Helyes lenne, ha a felhasználó kisgé-

pen vinné fel adatait papirszalagról mágnesszalagra, erre nincs mindig lehetőség. Változó hosszu, lyukszalagos rekordok fix hosszuságu mágnesszalagos rekordokba vitelére szolgáló segédprogram készült [13]

- Az operátori tevékenység során érdekes lehet egy kiválasztott job állapota. Futás közben a rendszer információ elég hézagos /... JOB SELECTED ... üzenetet kapunk a DISPLAY parancsra/, ezért /nagy prioritásu/ program szükséges a job prioritásának, hátralévő és eltelt idejének kiirására. Ennek segitségével az operátor eldöntheti, pl olyan várakozási állapot esetén, mikor két job a másik által épp lefoglalt erőforrásra vár, melyiket érdemes törölni [11].

2.3 A felhasználók irányitása

A hatékonysággal közvetlen összefüggésben van a job-osztályok megállapitása, mert a prioritás kijelölés osztályok szerint történik. CPU-idő, főtár-helyigény és lemezcsomag szám szerint sorolódnak a job-ok. Jelenleg 6 osztály van, a JCL ellenőrzéstől a 800 kbyte memóriát, 3 saját lemezegységet és 6 mágnesszalagos egységet "korlátlan" ideig használó job-okig terjed, a job-ok osztályuk alapján napi többszöri fordulástól heti egyszeri fordulásig kerülhetnek sorra [14].

A számlázás is osztályonként változtatja a sulyokat. Mind az osztályok kijelölése, mind a sulyok megválasztása az SMF statisztikák alapján történik.

3. Eredmények, lehetőségek, jövő

A hangolás eredményéről tájékoztat az SMF vizsgálata, job mix-ek futtatása /bench-mark job-ok/ - és a felhasználói vélemény. A leirt változásokat szükségesnek és hasznosnak tartjuk; legtöbbjük olyan, hogy hiba lenne nem megvalósitani.

Szándékunkban áll HASP rendszert bevezetni, ezt közvetett [1] és közvetlen tapasztalatok indokolják, várható az I/O tevékenység javulása, hatása valószínüleg sokszorosan meghaladja egy-egy hangolási lépését.

Javasoljuk a rendszer által nyujtott lehetőségek felmérését és kihasználását, a GTF, SMF opciók és a SERVICE AIDS segédprogramok [5], [7], [8] használatát a hatékonyság méréséhez.

Aki hasonló környezetben kiván számitógépet müködtetni, a konfiguráció tervezésénél feltétlenül gondoljon 7 és 9 csatornás, többféle irássürüségü mágnesszalag beszerzésére, rajzoló /plotter/ is szükséges. Ajánlható /gyakran szükséges/ a kezdeti után ujabb rendszergenerálás, az igények pontosabb ismeretében.

Rendszerünk hamarosan kibővül nagykapacitásu mágneslemezekkel, majd terminálos programjavitó és futtató rendszer is megvalósul. E változások igen nagy horderejüek, önmagukban is sokszorosára növelik, reményünk szerint, a hatékonyságot, és nyilván sok uj hangolási feladatot hoznak, megváltoznak az eddigi szük keresztmetszetek.

A leirt hangolási folyamat a KFKI Számitástechnikai Főosztály R40 software csoport munkájának eredménye, köszönetem fejezem ki Zimányi Magdának és Dobolyi Zsoltnak.

Irodalom

[1] INFOTECH, State of the Art vol II, 18
INFOTECH Limited Maidenhead Berkshire England

[2] Auerbach Computer Technology Reports Systems Software 613.
Auerbach Publ. 1974.

[3] Storey, T. - Todd, S.: Performance analysis of large systems.
Software Practice and Exp. 7. No. 3. /June-July 1977/

[4] Warshall, S.: On computational costs.
Annual Review in Automatic Programming 5., Pergamon, 1969.

[5] IBM 360/OS - MVT Guide
GC 28-6720-4

[6] IBM 360/OS - System Programmer's Guide
GC 28-6550-9

[7] IBM 360/OS - SMF
GC 28-6712-7

[8] IBM 360/OS - Service Aids
GC 28-6719-3

[9] IBM 360/OS - System Generation
GC 28-6554-13

[10] IBM 360/OS - Storage Estimates
GC 28-6551-16

[11] IBM 360/OS - Operator's Reference
GC 28-6691-3

[12] IBM 360/OS . Messages and Codes
GC 28-6631-13

[13] KFKI Programozási Tájékoztató
Budapest, 1978/1.

[14] Az R40 számitógép üzemeltetési rendje.
KFKI, 1978 május 15.

RENDSZERPROGRAMOZÁSI NYELV FELHASZNÁLÁSA OPERÁCIÓS RENDSZER BŐVITÉSÉHEZ

Gerhardt Géza
KSH Államigazgatási Számitógépes Szolgálat

A szerző által kidolgozott rendszerprogramozási nyelv /Design and Implementation of a High Level System Programming Language, Second Hungarian Computer Science Conference Preprints 1977./ több implementációja elkészült, ill. folyamatban van különböző számológépekre. Az eddigi implementációk közös jellemzője, hogy mini-, ill. mikroszámológépekre készültek. A nyelv felhasználásának eddigi tapasztalataiból úgy tünik, hogy érdemes azt nagygépek esetén is használni rendszerfejlesztési feladatok programozására.

A szóban forgó implementáció a Honeywell-Bull 66-os gépekre készül. Célja az eddigi kisgépes implementációkkal szemben nem önálló rendszerek létrehozása, hanem a gép meglévő operációs rendszerének bővitése. Ennek megfelelően a fordítónak nemcsak a gép utasitásrendszerére kell tekintettel lennie, hanem a gép operációs rendszerének megkötéseit, konvencióit is ki kell szolgálnia. Előzetes vizsgálatainkból úgy tünik, hogy a rendszerprogramozót sujtó számos kötöttség, szabály, ami az assembly programozást ne-

hézzé teszi, és sok hiba forrása is, nagyrészt automatikusan kielégithető a fordítás során. Meglepő módon ide tartoznak az operációs rendszer szinkronizációs primitivjei is.

Szeretném kiemelni a már meglévő software környezethez való alkalmazkodás feltételeit, nyelvi következményeit, kiemelve, hogy ezek figyelembe vétele ugyan olyan módon lehetséges, ahogy a hardware adottságoka a fordítók figyelembe veszik.

A fentiekben vázolt lehetőség - magasszintü nyelvek használata meglévő rendszerek bővitésére is - a későbbiek folyamán talán lehetővé teszi nagyobb szabásu fejlesztések létrehozását már meglévő rendszerek esetén is.

1. A GESAL nyelv rövid ismertetése

A nyelv definiálásakor két felhasználási terület lett figyelembevéve: rendszerprogramozás és bonyolult alkalmazói programok irása. A nyelv szintjének meghatározásánál ez a következőket jelenti: a nyelv tudja lehetőleg mindazt, amit a jelenlegi számitógépek "hardware"-ben tudnak /ami többé-kevésbé egyértelmüen és optimálisan lefordítható gépi utasitásokra/, de ne tudjon /lényegesen/ többet. Ez utóbbi, első hallásra meglepőnek tünő szempont már több esetben hasznosnak bizonyult.

A nyelv részletekbe menő ismertetésére itt nincs mód, csak néhány alapvető vonását soroljuk fel:

a/ Program struktúra

Egy program szekciókból áll, melyeknek egy vagy több entry-pontja van. A szekció egyben forditási egység. Egy szekció hivatkozhat más szekciók entry-pontjaira /általában proc-okra/, valamint használhatja az azokban deklarált típusokat. Irható "common" szekció, amely az összes szekcióból elérhető változókat tartalmazza.

b/ Típusosság

A nyelv típusos változókkal dolgozik. A típusok részben beépitettek, részben a felhasználó által deklaráltak. Az alaptípusok /char, int, longint, real, longreal/ a gépi alapmennyiségeknek felelnek meg. Deklarálható tipus az enumeráció /lehetséges értékeit a felhasználó felsorolja/, set tipus /bitenkénti megfeleltetés/, struktúra tipus /változómező felosztása különböző tipusú mezőkre/, union tipus /memóriaterület elérése különböző tipusokkal/ proc tipus /paraméterek és az eredmény tipusának hozzárendelésével/.
Bármely tipusból képezhető tömb /0 alsó, fix felső indexhatárral/, referencia az adott tipusra, valamint ezek tetszőleges kombinációja.

c/ Aritmetika

Az aritmetikai utasitás a megszokott formájú, prioritással ellátott operátorokkal, zárójelezhető. Eltérés az értékadás müvelete ./->/, amely balról jobbra mutat. Az operátorok készlete fel-

öleli a gépi müveleteket /and, or, xor, shiftek is/. A müveletek kiértékelése során tipusellenőrzés történik. Az összetett tipusok kezelésére speciális müveletek szolgálnak: struktúra elem elérésére a . /szelekció/, tömbelem elérésére a [ ]/indexelés/, dereferenciálásra a ↑/indirekt elérés/.

d/ Proc definició, memóriakezelés

A nyelv alapvetően nem blokkstruktúrált /pontosabban: egy mélységü blokkstruktúrával rendelkezik/. Minden szekció globális deklarációkból és proc-okból áll. A globálisan deklarált menynyiségek az összes procból elérhetők, proc-ok belsejében deklaráltak csak az adott proc-ban. Proc-ban proc-ot deklarálni nem lehet. A globális változók a szekcióra nézve statikusak, a lokálisak stacken allokálódnak. Rekurziv prochivás lehetséges. Lehetőség van egyéb memóriaosztályok bevezetésére. Proc paraméterei és az eredmény csak egyszerü változók és pointerek lehetnek. Paraméterátadás érték szerint történik.

e/ Vezérlési struktúrák

A nyelv az ALGOL 68, ill. a PASCAL vezérlési struktúráit alkalmazza /IF, WHILE, FOR, GOTO, CASE/. Utasitás-orientált, ilyenek a vezérlési struktúrák is.

f/ Gépfüggetlenség

A nyelv eredetileg a 16 bites szavakkal /8 bites byte-ok/ rendelkező kisgépek esetére lett kidolgozva, ezeknél a gépeknél teljes gépfüggetlenséget igér. Ennek lehetőségét két meglévő implementáció /TPA 70 és INTEL 8080/ igazolja.

2. A HwB 66 gép hardware és software sajátságai

a/ Hardware

A gép szószervezésü, kettes komplemens kódban dolgozó, 36 bites szóhosszúsággal rendelkezik. Egy cimes, gyakorlatilag egy akkumulátoros. Minden cim abszolút, 18 bites. Két kódrendszert alkalmaz: a 6 bites BCD és a 9 bites ASCII-t. Főbb cimzési módjai: normál, direkt /immediate adat az utasitásban/, indexelt, pre, ill. postindirekt. Speciális indirekt cimzési móddal 6, ill. 9 bites karakterek is elérhetők, inkrementális, ill. dekrementális cimzésre is lehetőséget nyújt.
Két üzemmódja van, a master és a slave mód. Master módban minden cim abszolút, az összes utasitás használható, slave módban minden cim relativ egy relokációs regiszterhez képest, csak a memória egy része érhető el.
Minden utasitás tartalmaz egy interrupt inhibit bitet.
Lehetőség van többprocesszoros kiépitésre, közös memóriával.

b/ Software

Az alábbiakban a GCOS operációs rendszer főbb vonásait ismertetjük, a hangsúlyt a programokkal szembeni konvenciókra helyezve.

A memóriában valamely pillanatban a rendszer rezidens része, valamint slave programok vannak, amelyekhez un. program number /1-63/ tartozik. Ez utóbbiak részben rendszerfunkciókat látnak el, de ilyenek a felhasználói programok is. Általában slave módban futnak, a rendszerprogramok áttérhetnek master módba is. A rezidens rész áll egy közös adatterületből /rendszer-változók és táblázatok/, valamint /többnyire reentráns/ modulokból. Rezidens az 1-es számú slave program is. A slave programok két részből állnak, a slave 0 fölötti /slave módban elérhető/ területből és az ez előtt levő un. slave service területből. Ez utóbbi a programhoz /job-hoz/ szükséges rendszerváltozókat tartalmazza, valamint buffert egy rendszermodul számára. Ujabb rendszermodul hívása esetén a hívó file-n stackelődik. Rezidens és a slave service területre overlay-eződő modulok hívására rendszerprimitívek szolgálnak /.CALL, .GOTO, .EXIT/.

Speciális slave program a time sharing rendszer /TSS/. Ennek saját common adatterülete, executiveja, valamint buffer-területe van az un. alrendszerek részére. Az alrendszerek általában slave módban futnak.

A rezidens rész common területének elérése a változók abszolút címével történik.
A rezidens modulok használhatnak abszolút címeket saját területükön is, de a relokációt maguknak kell elvégezni /egy un. INIT proc segitségével/.
A nemrezidens modulok szükségszerüen eltolhatóak /memóriakezelés csúsztatással és swappel történik/. Saját területük elérésére relativ cimzést használnak, a slave service terület elérése, valamint a slave programé indexelten történik. Master módban 3 index regiszter értéke kötött: az 5-ös tartalmazza a kurrens slave program 0-cimét, a 6-os a program number-t, a 7-es a processor number-t.

Az egész rendszer assembly-ben íródott, a modulok közti kapcsolatot makró és szimbólum könyvtáron keresztül tartják fenn.

3. A forditás szempontjai

a/ A hardware figyelembevétele

Bár a nyelv 16 bites gépekre lett kidolgozva, a 36 bites szóhosszúság nem okoz különösebb problémát. A 8 bites CHAR helyét a 6 bites CHAR és a 9 bites BYTE veszi át. Az INT 18 bites /félszó/, a LONGINT 36 bites, ugyanígy a REAL. Ezzel a megfeleltetéssel az INT és a pointer változók hossza megegyezik, éppúgy, mint 16 bites gépeknél, sok esetben a transzportabilitást is megőrizzük.

b/ Az operációs rendszerhez történő alkalmazkodás

A GCOS "modul" fogalmának nyelvi szinten a szekciót feleltethetjük meg. Szerencsés véletlen, hogy ez a megfeleltetés szinte egy az egyben történhet. A GCOS modul éppen úgy fordítási, ill. betöltési egység, mint amilyen a szekció. Minden szekció fordításához egyetlen paramétert kell hozzárendelni, amely megadja a rendszerben elfoglalt helyét /rezidens, slave service, TSS, közönséges slave/.
A fordítási szabályok ennek megfelelően a következők:

- rezidens rész fordításakor az INIT proc /relokáció elvégzésére/ automatikusan generálandó;
- overlay modul fordítása esetén ha lehet relativ címzést kell generálni, ha ez nem megy, az abszolút cim kiszámolása és felhasználása alatt interrupt inhibit-et kell alkalmazni;
- szekciók /modulok/ közti hivás, visszatérés esetén a rendszerprimitíveket kell használni. Ezek a szükséges stack-kezelést, az esetleges diszk kezelést elvégzik;
- a rendszer rezidens adatterületének leírását common szekció tartalmazza, a slave service terület leírását ugyanez tartalmazza speciális memóriaosztály bevezetésével.

Problémát a rendszer már megirt proc-jaihoz való fordulásnál csak a paraméterátadás jelent.

A sok különböző paraméterátadási szabály leírása a proc-ok definicióiban, nyelvbővitésként lehetséges.

4. Speciális kérdések

Van az operációs rendszernek néhány olyan sajátsága, amelyet célszerü nyelvi szintre emelni. Ilyenek a következők:

- Áttérés slave módból master módba és vissza, /Slave szekció esetén/. Célszerünek látszik erre a funkcióra egy utasitás zárójel-párt bevezetni /MASTER...ENDMAST/, így a fordító a megfelelő kódot ki tudja generálni, valamint más kódgenerálási stratégiára tér át a közbeeső kódnál.
- A GCOS alapvetően többprocesszoros, multiprogramozott rendszer. Ennek megfelelően szükség van kritikus változóinak védelmére konkurrens hiba ellen. Az alkalmazott módszer pontosan megegyezik a Brinch-Hansen által leírt eljárással /Principles of Operating Systems/. Az általa javasolt jelölés bevezetésével /shared v; ... with v do ... od/ a védelmet végző kód automatikusan generálható.

5. Tanulságok

A GESAL/HwB nyelv szükségszerüen limitált felhasználói területtel rendelkezik. Az itt követett módszer azonban példa lehet arra, hogyan lehet más, meglévő rendszert rendszerprogramozói nyelv fel-

használásával továbbfejleszteni, illetve ilyen nyelven alapuló új rendszert írni. Ebben az esetben a követendő munkamenet a következő:

- A rendszer alapkoncepciójának kidolgozása.
- A koncepció által megkivánt programstruktúra és kódolási szabályok kidolgozása.
- Az alkalmazni kívánt nyelv egyes elemeinek megfeleltetése a struktúrával.
- Megfelelő kódgenerátor irása, ill. módositása.
- A modulhivást végrehajtó rendszerkomponens bekódolása.

A továbbiakban a rendszer az alkalmazott nyelven fejleszthető tovább.

I N T E R A K T I V P R O L O N
R E N D S Z E R /IPR/
A SZÁMOK INTERAKTIV TANULÓ-FORDITÓPROGRAMJA AZ OKTATÁS SZOLGÁLATÁBAN

Gordon Erzsébet - Székely Zoltán
Nemzetközi Számitástechnikai Oktató és Tájékoztató Központ

1. Bevezetés

A világ számos oktatási intézményében foglalkoznak oktatási célu forditóprogramok tervezésével és megvalósitásával. A SZÁMOK-ban 1974 óta törekszünk egy hasonló software termék előállitására A megfelelő nyelv kiválasztása ugy történt, hogy az megfeleljen a SZÁMOK oktatási koncepciójának és ezáltal hatékonyan támogassa a hazai követelményeknek megfelelő programozó és szervező képzést. Az alapfoku programozói vagy szervezői tanfolyamra beiratkozó hallgató a "Programozás alapjai" cimü tantárgy keretein belül sajátitja el a gyakorlati programozás elemeit, melyhez az általunk definiált PROLON biztositja a nyelvi hátteret /PROLON - Programozási Logikára Orientált Nyelv/. Gyakorlati szempontból nem tartottuk célszerünek, hogy ez egy fiktiv nyelv legyen, ezért egy PL/I D subset mellett döntöttünk. Elsőként az 1976 óta müködő kötegelt PROLON változatot valósitottuk meg, amelyhez a SZÁMOK IBM 370/145-ös számitógépe

szolgált háttérként. Erről már beszámoltunk az előző konferencián 1975-ben. Az azóta eltelt idő alatt összegyüjtött oktatási és software készitési tapasztalatokat felhasználva, 1978 elejére elkészitettük a PROLON interaktiv változatát. Ennek implementálását a SZÁMOK PDP 11/7o-es 16 terminálos time-sharing gépén végeztük el.
A kötegelt PROLON lényege az, hogy a load-and-go compiler-interpreter rendszer egy job step-ben tetszőleges számu hallgatói programot képes.leforditani és végrehajtani. /Pusztán ennek a feladatnak az elvégzése a DOS alatt a PL/I fordító és a Linkage Editor segitségével, kb. lo-szer annyi gépidőt vesz igénybe!/ A PROLON másik nagy előnye, hogy az apró pontatlanságok és szintaktikus hibák nem feltétlenül terminálják a forditást, mivel a rendszer, feltételezésekkel élve, korrigálja a hibákat és kikényszeriti a végrehajtást. A hibaüzenetek oktató jellegüek és magyar nyelvüek. A rendszer nyomkövetési eszközökkel is rendelkezik.

A most bemutatásra kerülő Interaktiv PROLON Rendszer /IPR/ még fokozottabb mértékben képes teljesiteni oktatási célkitüzéseinket. Legfőbb előnye az, hogy lehetővé teszi az IPR egyéb lehetőségei mellett a felhasználó /hallgató/ közvetlen interaktiv kapcsolatba kerüljön magával a forditóval. A terminálról bevitt forrásprogram szintaktikus-szemantikus ellenőrzése soronként történik. Természetesen, ellentétben a kötegelt változattal, itt nem kerül sor hibajavitásra, hanem ehelyett a fordító kényszeriti a hallgatót a hibás sor korrigálására. Az IPR forrásnyelve ugyancsak PL/I subset és a kötegelt PROLON kompatibilis az IPR nyelvével, de nyelvi készlete jelentősen bővült.

2. Az IPR nyelve

Az alábbiakban röviden összefoglaljuk az IPR PROLON nyelvét:

Egy program egy vagy több külső eljárásból állhat. Információ átadás kizárólag paraméterlistán keresztül valósulhat meg.

Főeljárás:

```
eljárásnév: {PROCEDURE} OPTIONS(MAIN);
            {PROC     }
[adatleiró utasitás]...
[utasitás]...
END [eljárásnév];
```

Eljárás:

```
eljárásnév: {PROCEDURE} [(paraméterlista)];
            {PROC     }
[adatleiró utasitás]...
[utasitás]...
END [eljárásnév];
```

A paraméterek száma legfeljebb 10.

Adatleiró utasitás

```
{DECLARE} listaelem [,listaelem]... ;
{DCL    }
```

Minden változót deklarálni kell. Megadhatók elemi, tömb /max. 3 dimenziós/ és struktura /1 és 2 szinten/ változók. A lehetséges tipus attributumok: FIXED, FLOAT, CHARACTER /hossz/.

A rekordmódu adatátvitelhez szükséges file-ok deklarációja:

```
{DECLARE} filenév FILE {INPUT } RECORD;
{DCL    }          {OUTPUT}
```

Értékadó utasitás

```
{változónév    }
{SUBSTR (i,j,k)} = kifejezés;
```

Csal elemi értékadás van. A kifejezés lehet aritmetikai vagy karakterlánc tipusu.
Aritmetikai müveletek: + - * / **.
Karakterlánc müvelet:: !! konkatenáció .
A kifejezésben az alábbi függvényhivatkozások szerepelhetnek:

```
ABS(X)      ATAN(X)     COS(X)
EXP(X)      LOG(X)      LOG10(X)
SIN(X)      TAN(X)      SQRT(X)
SUBSTR (i,j,k).
```

Feltételes utasitás

```
IF logikai kifejezés THEN utasitás-1 ELSE utasitás-2
```

A logikai müveletek: ¬ /nem/, & /és/, | /vagy/.
Relációs müveletek: <, ¬<, <=, =,¬=, > , ¬>,>=.
Az IF utasitások egymásba ágyazhatók.

DO-END csoport

```
DO; [utasitás]... END;
```

Az IF utasitásban utasitás zárójelként használható.

Ciklus képzés

```
DO WHILE (logikai kifejezés);
  [utasitás].
END;
DO változónév=kifejezésl TO kifejezés2 [BY kifejezés3]
  [utasitás]...
END;
```

A DO utasitások egymásba ágyazhatók.

Vezérlésátadó utasitás

GOTO cimke;
Minden nem DECLARE utasitásnak lehetnek cimkéi, de nem lehet ugratni IF utasitás THEN ágára vagy ELSE ágára, valamint DO csoport vagy DO ciklus belsejébe.

Eljáráshivás

CALL eljárásnév [(argumentum-lista)] ;
A praméterátadás cim szerint történik. A paraméter nem lehet kifejezés.
Rekurziv hivás nincs.

Visszatérés eljárásból

RETURN;

Adatátviteli utasitások

A rekord módu adatátvitel utasitásai:

OPEN FILE(filenév) [,FILE (filenév)] ...;
CLOSE FILE(filenév) [,FILE (filenév)] ...;
READ FILE (filenév) INTO (változónév);
WRITE FILE (filenév) FROM (változónév);

A stream módu adatátvitel utasitásai:

GET LIST (adatlista);
GET EDIT (adatlista) (formátum lista);
PUT [{PAGE / SKIP}] LIST (adatlista);
PUT [{PAGE / SKIP}] EDIT (adatlista) (formátum lista);

A formátum lista lehetséges elemei:

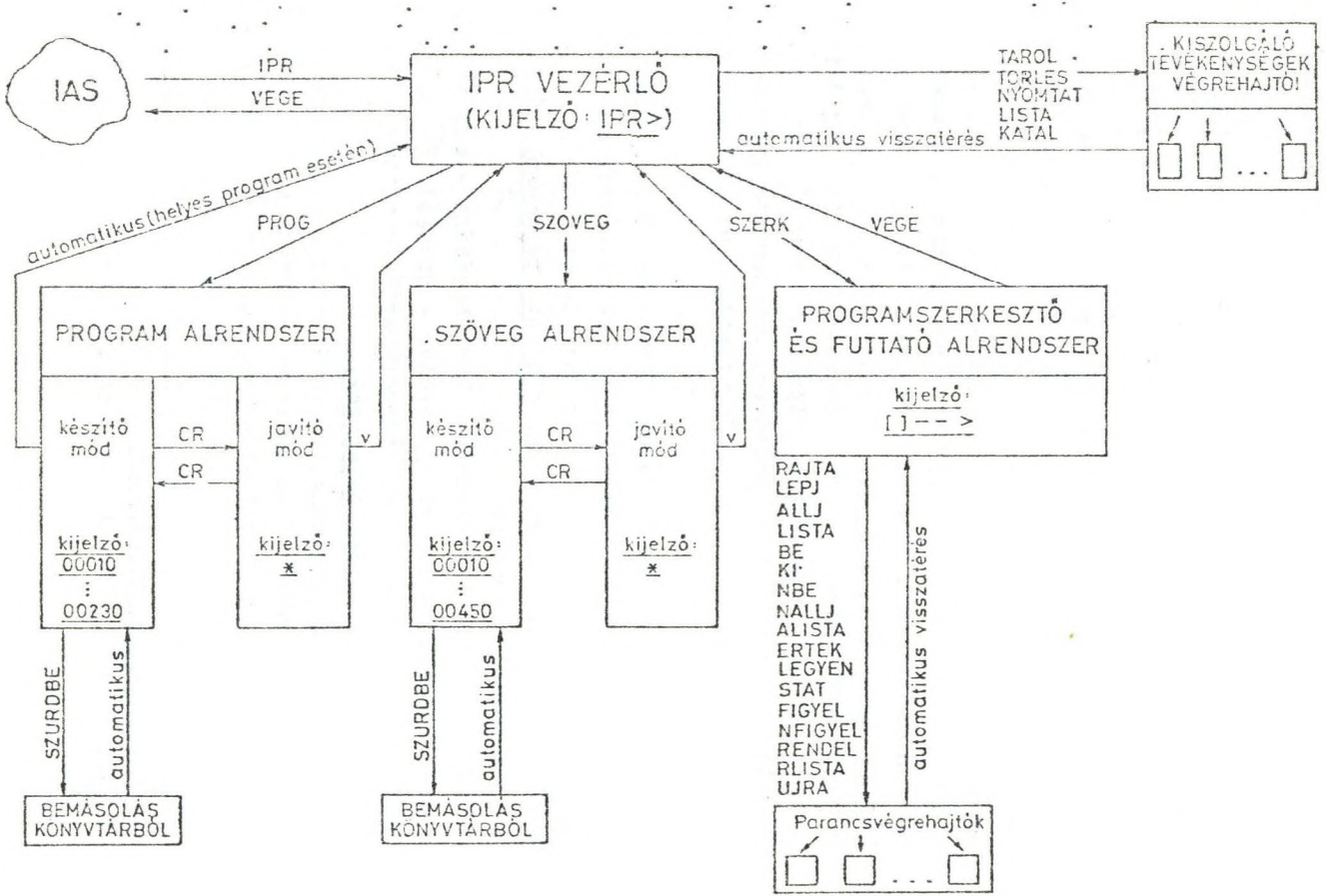
F (W [,d])
A (W)
X (W)

3. Az IPR file-rendszere és parancsnyelve

Az IPR a PDP 11/7o IAS time-sharing operációs rendszere alatt müködő parancs.

Az IPR file-rendszere egy ideiglenes file-ból, a munkaterületből és állandó file-ok rendszeréből, a könyvtárakból áll. A munkaterület a terminálról vagy egy állandó file-ból bevitt tetszőleges szöveg vagy program /eljárás/ tárolására szolgál. A könyvtárak tetszőleges számu file-ból állhatnak. Minden IPR felhasználónak van egy saját könyvtára. Minden tankör rendelkezik egy közös tankör-könyvtárral, és az összes IPR felhasználóhoz tartozik egy IPR rendszer-könyvtár. Az igy definiált három könyvtárszint hierarchikus szerkezetet alkot. Amig a felhasználók saját könyvtáraikat tetszőlegesen módosithatják, addig a felettük álló, magasabb szintü könyvtárak file-jait csak olvasási funkcióval érhetik el. A tárolt file-ok a következő tipusokkal rendelkezhetnek: tetszőleges szöveg, programszöveg, sornyomtató output-ra készitett file, a program WRITE utasitása által létrehozott file.

Az IPR parancsnyelvét és egyben a rendszer teljes szerkezetét a mellékelt ábra szemlélteti. /A parancsokban megadható paramétereket és azok funkcióit nem tüntettük fel/.
Az IPR főszintjén kiadható un. főszintü parancsok látják el az alapvető tevékenységeket.
A PROG parancs PROLON eljárások létrehozására - szintaktikus-szemantikus ellenőrzéssel egybekötve - szolgál. Ebben az alrendszerben egy saját szövegszerkesztő is aktivizálható. Itt adhatókki az editor parancsok.
A SZÖVEG parancs csak annyiban különbözik a PROG parancstól hogy nincs szintaktikus-szemantikus ellenőrzés. A SZERK parancs a több eljárásból álló végrehajtható program létrehozására és futtatására szolgál. Hatására a program a futtató alrendszerbe kerül, ahol a program futására és nyomkövetésére vonatkozó parancsok /RAJTA, LEPJ, ALLJ, stb/ adható ki. A többi főszintü parancs kiszolgáló tevékenységet lát el.


IAS
IPR
VEGE
IPR VEZÉRLŐ
(KIJELZŐ: IPR>)
TAROL
TORLES
NYOMTAT
LISTA
KATAL
automatikus visszatérés
. KISZOLGÁLÓ
TEVÉKENYSÉGEK
VÉGREHAJTÓ!
automatikus (helyes program esetén)
PROG
SZOVEG
SZERK
VEGE
PROGRAM ALRENDSZER
készítő
mód
CR
CR
javító
mód
v
kijelző:
00010
:
00230
kijelző:
*
SZÖVEG ALRENDSZER
készítő
mód
CR
CR
javító
mód
v
kijelző:
00010
:
00450
kijelző:
*
PROGRAMSZERKESZTŐ
ÉS FUTTATÓ ALRENDSZER
kijelző:
[ ] - - >
RAJTA
LEPJ
ALLJ
LISTA
BE
KI
NBE
NALLJ
ALISTA
ERTEK
LEGYEN
STAT
FIGYEL
NFIGYEL
RENDEL
RLISTA
UJRA
automatikus visszatérés
SZURDBE
automatikus
BEMASOLAS
KONYVTARBOL
SZURDBE
automatikus
BEMASOLAS
KONYVTARBOL
Parancsvégrehajtók

4. Az IPR implementálása

Az IPR megvalósitására a következő számitógépes környezetben történik:

PDP 11/7o-es time-sharing gép /256 kb-os központi memóriával/

1 88 mb-os lemezegység
1 mágnesszalag egység /800 B/inch/
1 sornyomtató
1 kártyaolvasó
8 alfanumerikus display
8 teletype

A rendszer a DEC által szállitott IAS /Interactive Application System/ time-sharing operációs rendszer alatt, a MACRO-11 assembly nyelven készült.
A rendszerterv hüen tükrözi a mellékelt ábrán bemutatott szerkezetet.

Néhány fontosabb részletet szeretnénk kiemelni az általunk megvalósitott rendszerből.

- A szükséges file-müveletek elvégzésére egységes file-kezelő makrókat irtunk és az IPR csak ezeken keresztül érintkezik az IAS I/O müveleteivel.

- A rendszer legfontosabb tevékenységét, a programforditást egy olyan felismerő látja el, amely a PROG parancs alatt csak szintaktikus-szemantikus elemzést végez és,a szimbólum tábla előállitásán kivül, semmiféle kódot nem generál. Tároláskor, egységesen, csak a forrásszöveg őrződik meg. A SZERK parancs alatt ugyanez a felismerő müködik. Segitségével felépül az összeszerkesztett programot alkotó eljárások szimbólum táblája és a program futás-ra kész állapotba kerül. Ezuttal sincs kódgenerálás. A program továbbra is forrásállapotban marad, csupán annyiban különbözik eredeti formájától, hogy az egyes forrásnyelvi utasitások elé, a nyomkövető utasitások számára szükséges byte-ok kerülnek beszurásra. Az igy keletkezett ideiglenes forrásfile-t a futtató alrendszerből történő kilépés után töröljük.

A program végrehajtását is ugyanaz a felsimerő végzi, amely PROG állapotban elemzőként müködött - ezuttal ez interpretálja a forrásprogramot. Ez a módszer nagymértékben egyszerüsitette a felhasználói file-ok kezelését, ugyanis egységesen, csak forrásnyelvi programokat tárolunk. Az a tény, hogy futás közben rendelkezésre áll a forrásszöveg, lehetővé teszi, hogy a futás közben kiadott hibaüzenetek beszédesebbek legyenek, valamint azt is, hogy a program várakozási állapotban - bármikor - listázható legyen a terminálra.

A rendszer 1976-77-ben készült a SZÁMOK Software Fejlesztési Osztályán, mintegy 10 eév munkaerő ráforditással. 1978 elejétől kezdve - kisérleti jelleggel - sikerrel alkalmazzuk az IPR-t az oktatásban.

Irodalom:

David Gries: Compiler Construction for Digital Computers.

dr.Mérey-Tóthné Gordon E.: A PROLON nyelv és forditórendszer /Információ Elektronika 1976.3.sz./

Budinszky-Buxton-dr.Mérey-Tóthné Gordon E.: Az Interactive Educational System /PROLON/ /Informatica '76 - Bled, oktober 1976./

KISSZÁMITÓGÉPES FUNKCIÓK INTEGRÁLÁSÁNAK TAPASZTALATAI

Gilicze László
Központi Statisztikai Hivatal Államigazgatási
Számitógépes Szolgálat

A nagy kapacitású számitógépek mellett világszerte növekvő szerep hárul az egyre olcsóbb üzemü kisszámitógépekre. Az ASZSZ eszközbázisának kialakitása során az egyik célkitüzés az volt, hogy az eddig gazdaságtalanul manuálisan, vagy nagygéppel megoldott feladatokat kisszámitógéppel automatizáljunk. Ebből a célból több kisszámitógépes fejlesztést inditottunk el, mivel a beszerezhető rendszerek nem feleltek meg a nagy számitóközpont speciális követelményeinek. Ezek közül már három rendszeren /Interscan 2100, R10, DN700/ már alkalmazási és üzemeltetési tapasztalatokkal rendelkezünk.

Az ÁSZSZ Honeywell-Bull 66/60 nagygép köré épülő számitóközpontja komplex alkalmazói igények kielégitésére készült, jellemzői:

- multi-access /helyi és távolsági kötegelt, időosztásos feldolgozások, adatbázis alkalmazása, stb/
- sok felhasználó kiszolgálása /7 minisztérium és főhatóság/
- kiterjedt távadatfeldolgozási hálózatra épül.

Az egyes fejlesztések irányait az ebből fakadó feladatkörök ellátásának igénye és a rendelkezésre álló software erőforrások szabták meg.

FELADATKORÖK

Egész sor olyan feladatkör van egy nagyszámitógép környezetében, amelyek ellátására a kisgépes megoldás olcsóbb, alkalmasabb:

- adatrögzités és ellenőrzés
- kódkonverzió
- adathordozó átalakitások
- távoli terminálfunkciók

hálózatvezérlés

- helyi adatbázisok alkalmazása, stb

A hazai, vagy nyugati piacon kapható kisgépes berendezések ezek közül általában egy vagy kettő kielégitésére alkalmasak. A hazai hardware-software árviszonyok és az addigi alkalmazói tapasztalat szerint célszerü volt ezeknek a berendezéseknek a sokoldaluságát fokozni a software bővitésével. Ezt a berendezések feltételei általában lehetővé tették.

A RENDELKEZESRE ÁLLÓ BERENDEZESEK FELTETELEI

A fejlesztésekre különböző okokból figyelembe vett berendezések a következők:

- R10
- Honeywell-Bull System 700
- Interscan 2100

A Honeywellhez kapcsolódó hálózatban szereplő egyéb kisgépes berendezések /RC3500, Redifon, TPA/ főleg illesztési feladatokban voltak érintve.

A berendezések feltételeit a következő jellemzők határozták meg:

- periféria-választék
- megszakitási rendszer
- igényelt üzemeltetési környezet
- az operációs vagy célrendszer hatékonysága
- rendelkezésre álló software eszközök
- az eredeti tervezési koncepció rugalmassága.

A fejlesztési célkitüzések megállapitásához választani kellett az alábbi alternativák közül:

- milyen funkcióval melyik berendezést bővitsük
- a funkciók integrálásának melyik szintjét válasszuk
  - . teljes egyidejüséget valósitsunk meg
  - . automatikus rendszerváltást tegyünk lehetővé
  - . az egyes feladatokat ellátó célrendszereket manuális rendszercserék útján aktivizáljuk.

A JELENLEGI RENDSZEREK

A berendezések jelenlegi funkcióinak kialakitásához szükséges fejlesztések módjai:

- kooperáció a szállitóval
- önálló fejlesztés
- belföldi megrendelés.

Az egyes berendezéseken ellátott funkciók egyidejüségét az alábbi ábra szemlélteti:

FUNKCIÓ / GÉP	INTERSCAN	R10	DN707
ADATRÖGZITÉS		nem egyidejü /VIDEOPLEX/	
KONVERZIÓ			
REMOTE BATCH			
KONCENTRÁLÁS			
HELYI FELD.	manuális rendszercsere		manuális rendszercsere

ALKALMAZÁSI TAPASZTALATOK

Az INTERSCAN 2100 eredetileg egy csoportos adatrögzitést végző célrendszer. Kibővitve egy sokoldaluan használha-

tó háttérgéppé vált. A software bővitésével járó memória-problémát overlay-technikával hidaltuk át, ami az egyidejüség hatékonyságának valamelyes csökkenésével járt. Az üzemeltetési gyakorlat szerint ésszerü szervezés mellett a funkciók - egyidejü - használata legalábbis kielégitő.

Az R10-en az RTDM monitor minden szolgáltatása hozzáférhető a remote batch funkcióval párhuzamosan. Az R10 igy egy intelligens terminál lett, amely egyidejüleg négy Honeywell tipusu nagyszámitógéppel /vagy vonallal/ képes kapcsolatot tartani. Az egyidejüséget a remote batch memóriaigénye korlátozza, de a megoldás dinamikus.

A DATANET 707 a Honeywell 700-as rendszerén müködő koncentrátori és remote batch célrendszer. Nagy a szerepe a távadatfeldolgozási vonalhiány feloldásában, kényelmes kezelésü és hatékony hálózati kisgép. Az operációs rendszer szolgáltatásai jelenleg még nem egyidejüleg használhatók.

EGY PÁRHUZAMOS PROGRAMOZÁSI RENDSZER
/A CONCURRENT PASCAL ADAPTÁCIÓJA PDP-RŐL IBM-RE/

Groszmann Gusztáv
Számítástechnikai Koordinációs Intézet

1/ Egy absztrakt magasszintü eszköz koncepciója párhuzamos programozásra

A párhuzamos programozás a szekvenciálisnál általánosabb és természetesebb, amennyiben minden minden valóságos esetben párhuzamos tevékenységek együttmüködése ill. versengése történik megosztott erőforrásokon; viszont minőségileg bonyolultabb, amennyiben az események időbeli lefolyása egyedi, reprodukálhatatlan. Ennek legélesebb következménye az, hogy a hagyományos verifikálási módszerek párhuzamos programozási körülmények között használhatatlanok. A teszt nem feltétlenül mutatja ki a hibát /mivel nem állitja elő a hibás időzitési viszonyokat/, ha pedig kimutatja, a hiba lokalizálása az időfüggés miatt reménytelen /mivel reprodukálhatatlan/. A nyomozás erősen befolyásolhatja az időzitési viszonyokat, a hibát elkerülheti, ami a nyomozás elhagyásával megjelenik.

Ebben a szituációban a programozási nyelvek tervezőire hárul az a feladat, hogy olyan eszközöket épitsenek be a strukturába, amelyek az időfüggő hibákat elkerülhetővé teszik /"védelem programozási időben"/ ill. azokat még a forditóprogram által felismerhetővé teszik /"védelem forditási időben"/. Az első ilyen kisérlet a Concurrent Pascal /CP/ nyelv volt.[6.1.] A CP releváns eszközei:

1.1. Absztrakt moduláris felépités: process, monitor és class absztrakt adattipusok képezik a nyelv moduljait, amelyek privát adatok és rajtuk végezhető megengedett müveletek összessége /valamilyen egyszerü invariáns tulajdonsággal/. A forditóprogram felismeri a privát adatok illetéktelen ill. helytelen használatát.

1.2. Hierarchikus felépités: az absztrakt adattipusok definiciói ciklusmentesen egymásra épülnek, amit a forditóprogram ellenőriz.

1.3. Elérési jogok /irányitott/ gráfja: az absztrakt adattipusok példái iniciáláskor paraméterként átadott hivási jogokkal rendelkeznek más modulok felé /ciklusmentes irányitott gráf, szemben a blokkstruktura elérési fájával/; a hivási jogok megkerülését a forditóprogram felismeri.

Ez az architektura meghatározza a párhuzamos programozás egy metodologiáját is:[6.3.]

1.4. Tervezd meg /még könnyen áttekinthető funkcióju absztrakt egységekből/ a hierarchikus szinteket /"virtuális gépek"/, specifikáld,hogy azok mit realizálnak /bottom-up vagy top-down, azaz az absztrakció vagy finomitás irányába haladva; valószinüleg iterative/; rajzold fel az elérési jogok gráfját.

1.5. Tervezd meg, hogy az egyes absztrakt modulok hogyan realizálják privát adatok és rajtuk végezhető megengedett müveletek segitségével funkciójukat; ezután az egyes modulok leprogramozása /CP vagy bármely más nyelven/ már triviális feladat.

1.6. Verifikáld a programot bottom-up, szintről-szintre haladva, minden lépésben csak egyetlen uj modult állitva a már verifikált virtuális gép "tetejére".

További nyelvi követelmények:

1.7. A processek között megosztott absztrakt modulok privát adatainak integritásához /invariáns tulajdonságai fenntartásához/ szükséges, hogy az azokon végzett müveletek időben kölcsönösen kizárják egymást. Ennek kierőszakolása részben futási időben történik / a monitorok esetében/, részben forditási időben /a processek és class-ok esetében/.

1.8. Rendszerprogramozási célokból szükséges, hogy a processek felhasználói programokat indithassanak meghatározott elérési jogokkal /azaz megengedett operációs rendszer hivásokkal, amelyek megkerülését a forditóprogram felismeri/, továbbá a felhasználói programoknak hivási paraméterek legyenek átadhatók /az őt hivő bármely felhasználói program vagy process által/ a job control funkciók realizálása érdekében. Ezt tükrözi CP-ben a /szekvenciális/ program és a felhasználói programozást szolgáló Sequential Pascal-ban /SP/[6.2.] az interface /prefix/ rutin fogalma.

1.9. A párhuzamos programozási nyelvek absztraktnak kell lenni abban az értelemben, hogy gépfüggő részletektől /cim, regiszter, megszakitás, perifériakezelés/ lehetőleg eltekintsen. Ez a programot jelentősen leegyszerüsiti és bármely architekturán még hatékonyan végrehajtható.

Megjegyzések:

- az 1.1., 1.2., 1.3., 1.7., megszoritások nem zárják ki az időfüggő viselkedést, csak explicitté teszik azt és kizárják az időfüggő hibák leggyakoribb forrásait;

- bár a koncepció a megbizhatóságot állitja előtérbe, a hatékonyságot nem veszitjük el /a programkészités jelentős hatékonyságnövekedése mellett a futási hatékonyság tekintetében kedvező körülmény a "védelem futasi időben" minimalizálhatósága/.

2/ A Concurrent Pascal realizációja

A felvázolt koncepció software-technologiai értékeinek a gyakorlatba való átültetésére az első kisérlet az 1975-ben P.B. Hansen /Caltech, USA/ és munkatársai által eredetileg a PDP/11/45 kisgépre létrehozott /azóta több kisgépre áttett/ SOLO operációs rendszer volt. A realizálás a lehető legáttekinthetőbb megoldásokat követte és inkább engedményekre hajlott az általánosság /pl. statikus tárallokáció a CP nyelv rekurzivitása ill. a processek dinamikus törölhetőségének feláldozása árán/ ill. részben a hatékonyság /pl. a forditóprogramok nem PDP/11 gépi kódra forditanak/ tekintetében, szemben az elsődleges kezelhetőségi /egyszerüség, megbizhatóság, adaptivitás, átvihetőség/ kérdésekkel. A realizáció átvihetőségi szempontból releváns vonásai:

2.1. Absztrakt tárgykód: 6.3., 6.4. a forditóprogramok a Concurrent Pascal Machine /CPM/ virtuális gép gépi nyelvére forditanak, amely adekvát a CP/SP nyelvek implementálása számára és egy kompromisszumot képvisel a forditás és az interpretálás között a gépfüggetlenség ésszerü szintjének elérésére. A /home-gépen software-ben vagy firmware-ben realizált/ CPM felfogható olyan futási környezetnek, amely közös mag minden párhuzamos programozásu rendszerben, megvalósitva a következő absztrakt funkciókat:

2.1.1 monitorhoz rendelt short-term várakozási sor kreálása;
2.1.2 belépés monitorba ill. kilépés monitorból /azaz a monitorhoz rendelt várakozási sor kezelése/;
2.1.3 process kreálása /azaz állapotvektor és adatterület allokálása/ és befejezése /azaz törlése a short-term ready várakozási sorból/;
2.1.4 process felfüggeszti magát egy monitorban és belép annak egy kijelölt medium-term várakozási sorába;
2.1.5 process felébreszti egy monitorban azt a process-t, amely az illető monitor egy kijelölt medium-term várakozási sorában található, önmaga pedig kilép az illető monitorból;
2.1.6 process felfüggeszti magát meghatározott időre ill. operátori beavatkozásig /azaz két további virtuális medium-term várakozási sor kezelése/;

2.1.7 process felfüggeszti magát I/O-kérés miatt a müvelet befejeződéséig;

2.1.8 fair CPU-allokáció a futásképes processek között /azaz a short-term ready várakozási sor kezelése/;

2.1.9 megszakitások elrejtése;

2.1.10 összesen 112 egyszerü stack-, vezérlési- és állapotvektor-müvelet interpretálása.

A PDP/11-n a CPM-t egy 8Kbyte PDP/11 gépi kódu kernel realizálja.

2.2. Kiváló kezelhetőségü CP/SP forditóprogramok: SP /tehát magasszintü/ nyelven iródtak; jól szeparált 7 szintaktikai ill. szemantikai lépcsőben /menetben/ 6 közbülső nyelv segitségével végzik a forditást; beépitett tesztelhetőséggel rendelkeznek; az alkalmazott forditási módszer könnyen átvihető más magasszintü nyelvekre. Egy átlagos menet 33 Kcharacter ill. 10 Kbyte CPM gépi kód. [6.4.]

2.3. CP/SP felhasználókat jól kiszolgáló /modul/ operációs rendszerek: [6.3.] CP/SP nyelven iródtak; a SOLO job control parancsokat hajt végre és kiszolgál egy diszkfile-rendszert /5 kommunikáló process segitségével buffering módszerrel/; a JOBSTREAM kis jobok rövid válaszideju kiszolgálását végzi /5 kommunikáló process segitségével spooling módszerrel/; ezek az első kiváló kezelhetőségü /jóldokumentált, egyszerü, megbizható, könnyen módositható, bővithető és átvihető/ operációs rendszerek. A SOLO méretei:

- CP-ben: 25 Kcharacter ill. 8,5 Kbyte CPM gépi kód;
- SP-ben: 20500 sor ill. 212 Kbyte CPM gépi kód.

A realizáció igazolta a koncepció sikerességét: a rendszerprogramok kifejlesztésének hatékonysága egy nagyságrenddel javitható az assembler szinthez képest anélkül, hogy a futási hatékonyság érdemben csökkenne. [6.6.]

3. A Concurrent Pascal adaptációja

A DOS/VS Concurrent Pascal Programozási Rendszer /CPPS/ az előzőekben leirt kisérlet folytatásának tekinthető nagyobb teljesitményü számitógépes rendszer körülményei között. Az adaptációnak első fázisban az elsődleges célja a forrásrendszer koncepcióinak minél teljesebb, kompatibilis realizálása volt /sok tekintetben a tárgyrendszer hatékonyságának rovására/ annak IBM-n való mielőbbi használatbavétele, megismertetése és elterjesztése érdekében. A CPPS a CPM realizálásával és a CP/SP nyelvek implementálásával:

3.1. megteremti az IBM korszerü elveken alapuló párhuzamos programozásának magasszintü, strukturált lehetőségét /szemben az assembler szintü multitasking-gal/;

3.2. lehetővé teszi DOS/VS alatt ujszerü, saját fejlesztésü, magasszintü nyelven irható, gépfüggetlen operációs rendszerek kifejlesztését oly módon, hogy /pl. kompatibilitási okokból/ megengedi a hagyományos és az uj rendszerek egyidejü használatát;

3.3. a CPM szintjén teljes kompatibilitást nyujt a PDP 11 és az IBM /+DOS/VS/ között;

3.4. lehetővé teszi a párhuzamos programozás ezen korszerü irányzatának gyors követését /a forditóprogramok és a model operációs rendszerek változatlanul átvehetők voltak/;

3.5. kiinduló alapja további alkalmazási /pl. IBM-re szabás, többterminálos hozzáférhetőség/ és innovációs /pl. manager koncepció realizálása dinamikus monitor-allokáció érdekében/ továbbfejlesztésekhez /lásd 5.pont/.

A 2. pont alapján az adaptációra két módszer kinálkozik: 6.5.

3.6. IBM-re szabott tárgykód generálása kihasználva a CP/SP forditóprogramok jólkezelhetőségét egy várhatóan effektiv, de a forrásrendszerrel /CP/SP nyelvek szintjén/ bizonytalanul kompatibilis rendszert eredményez viszonylag nagy befektetéssel /azon kivül, hogy két cross- és két direkt forditóprogram megirása szükséges, a két-két nyelv szemantikai ekvivalenciája nehezen garantálható ill. verifikálható/;

3.7. kihasználva a CPM absztrakt, gépfüggetlen jellegét a CPM realizálása egy IBM-re irt kernel segitségével viszonylag kis befektetéssel egy várhatóan kevésbé effektiv, de a forrásrendszerrel CPM szinten is garantáltan kompatibilis rendszert eredményez /sem a forditóprogramok, sem a model operációs rendszerek nem változnak, sőt tárgykód formájában átvehetők/. Mivel a CPM-re P.B.Hansen formális definiciót nem adott, a legbiztonságosabb módszernek a forrásrendszer kernel-jének szolgai szimulációja igérkezett. Tekintettel a 3.2. szempont előnyeire és arra, hogy VM operációs rendszer nem állt rendelkezésre, egy DOS/VS-szel koegzisztáló változat egyszerübbnek és hasznosabbnak igérkezett egy stand-alone változatnál.

Utólagosan értékelve a 3.7. választás első fázisként helyesnek bizonyult: az adaptáció 6 ember-hónap kapacitást és 22 óra IBM-gépidőt vett igénybe /egy 18 Kbyte méretü kernel megirásához assemblerben és veri-

fikáláshoz/; a fordítási ill. interpretálási sebesség /IBM/370/145-ön/ közelitőleg megegyezik a forrásrendszerével /PDP/11/45-ön/, bár ez nyilvánvalóan erőssen függ a DOS/VS terheltségétől és a prioritási viszonyoktól. A CPPS 1978 április óta müködőképes.

4. Az adaptáció tapasztalatai

4.1. A 3.7. szerinti adaptációs módszer konzekvens végigvitelét a következő főbb tényezők korlátozták:

- erős architekturális inkompatibilitás;
- a forrásrendszer stand-alone, tehát a kernel szabadon rendelkezik a PDP architektura minden vonása fölött, viszont az IBM-en a kernel a DOS/VS alatt egy közönséges felhasználói program;
- a tárgyrendszernek elfogadhatóan hatékonyan kell müködnie.

Ezen korlátok következtében a 2.1.1. - 2.1.10. funkciók realizációja sok esetben tul kellett hogy lépjen a különféle konverziós módszereken /reprezentációs izomorfia/ és a kernel-komponensek logikájára ill. strukturájára mélyebben kiható változtatásokat is szükségessé tett. Azonban ezen változtatások invariáns jellege a CPM szemantikájára nézve többé-kevésbé triviális /a forrásrendszer alaposabb ismeretében/.

Az alkalmazott konverziók: CPM fixpontos szám↔IBM fixpontos félszavas szám; CPM lebegőpontos szám↔IBM lebegőpontos szám; CPM felhasználói 2 byte-os virtuális cim↔IBM 3 byte-os virtuális cim; CPM kernel módu 2 byte-os virtuális cim↔IBM 3 byte-os virtuális cim; ASCII↔EBCDIC; RK11 disk cim→2319 diskcim.

Strukturát ill. logikát érintő változások:

- a 2.1.1.-2.1.10. funkciók szükségessé teszik a kernel- és felhasználói módok megkülönböztetett realizációját, amely az ATTACH /"init process"/, WAITM /"delay"/, POST /"continue"/ supervisor makrók felhasználásával történt. A kernel egy altask, az interpreter a főtask. Az előbbi tetszés szerint átállithatja az utóbbi save-drea-ját és igy rákényszeritheti a szükséges ütemezési algoritmusokat /pl. a monitorbelépések kölcsönös kizárását/. Az aktuális process azonositható a főtask save-area-jának aktuális tartalmával. Az interpreter task 2.1.1.-2.1.7. funkciónál POST-tal hivja a kernel taskot, amelyik WAITM-mel "veszi át" és "adja vissza" a vezérlést /a supervisoron keresztül/ az interpreter tasknak. Az I/O-kérés /2.1.7./ lekezelése fizikai szinten történik /EXCP/. Az óra-funkciónak a SETIME, az operátori megszakitásnak az STXIT OC supervisor makro lett megfeleltetve. A kernel task kernelhivás nélkül kapja meg a vezérlést a supervisortól óra és I/O müvelet befejezésekor, amire szintén WAITM-mel vár. A logikailag legnehezebb

problémát a 2.1.9. funkció okozta /itt a megoldás felvázolására sem lehet vállalkozni/, t.i. a prioritás-orientált megszakitási rendszer az időfüggő hibák legfőbb forrásának bizonyult.

- az interpreter bizonyos visszatérő funkciói saját makrókba lettek tömöritve és bizonyos in-line funkciókból pedig külön rutin lett /ez egyszerüsitette a struktúrát/;
- a kernelben a szubrutinhivások és visszatérések nem stack segitségével történnek /a PDP/11 stack-szervezésü/;
- a CPM virtuális cim<->PCP/11-real cim konverzió hardware forditási map-ek segitségével történik; a tárgyrendszerben ezek a map-ek megszüntek és helyette szükség esetén /software-ben/ egy egyszerü lineáris cimtranszformáció történik;
- a konzol karakteres-echo mód helyett sorperifériaként müködik.

4.2. Verifikálási szisztéma

A CPM-et nagyméretü, bonyolult müködésü párhuzamos assembler program realizálja. A program helyességének elérésében a következő eszközök játszottak döntő szerepet:

- a kernel hierarchikus és moduláris felépitése /lásd 1.4.-1.6./ Az interpreternek, mint szekvenciális programnak, az önálló tesztelése külön erre a célra irt CPM gépi kódu IBM fázisok betöltésével és interpretálásával történt; ezután az iniciáló rész önálló belövése végül a kernel-class-ok bekapcsolásával az egész rendszer tesztje következett /a SOLO betöltése és interpretálása utján/;
- kváziszekvenciális üzemmód. A tesztelés a szekvenciális müködéstől haladt az egyre bonyolultabb párhuzamos müködés felé /egy közbülső szint pl. amikor az aszinkronitást csak bizonyos I/O müveletek képviselik, az óra az interpretált CPM-utasitások száma/.
- szelektiv nyomozási technika.

4.3. Az adaptáció legfontosabb rendszerprogramozási tanulsága egy nagativ megállapitás: a párhuzamos programozás legalacsonyabb nyelvi szinten a jelenlegi technológiák mellett csaknem járhatatlan ut a szinte kiszürhetetlen /főleg a megszakitások miatt fellépő/ időfüggő hibák miatt; a biztonságosabb magasabb szintü párhuzamos programozási vonások realizálásához pedig inadekvátak az IBM-szerü supervisor hivások

5. Továbblépések irányai 6.3.,6.7.

5.1. Nagyobb teljesitményü ill. méretü operációs rendszerek realizálására való alkalmazhatóság kipróbálása /ezideig CP-ben csak kisméretü, igen speciális célu rendszerprogramok iródtak/.

5.2. Programhelyesség-bizonyitási eszközökkel való megtámogatás.

5.3. Nyelvi továbbfejlesztések: speciális célu /pl. real-time/ alkalmazásokra; általánositások /pl. elérési jogok finomitása esetleg szigoritása, dinamikus erőforrás-allokáció/; hatékonyságjavitás /IBM-re szabás, tárgykód-optimalizálás/.

5.4. Számitógép-struktura tervezési eszközként való alkalmazhatóság vizsgálata /pl. a mikrogép a számitógéptervezésben az absztrakt adattipus szerepét játszhatná/.

6. Hivatkozott irodalom

6.1. Concurrent Pascal Report /P.B.Hansen, Caltech, 1975/

6.2. Sequential Pascal Reprot /P.B.Hansen, Caltech, 1975/

6.3. The Architecture of Concurrent Programs /P.B.Hansen, Prentice Hall, 1977/

6.4. A Concurrent Pascal Compiler for Minicomputers /A.C.Hartmann, Lecture Notes in Computer Science, 50; 1977/

6.5. Concurrent Pascal Implementation Notes /P.B.Hansen, Caltech,1975/

6.6. Experience with Modulor Concurrent Programming /P.B.Hansen, IEEE Transactions on Soft.Eng. 1977, 2/

6.7. Extending Concurrent Pascal to Allow Dynamic Resource Management /A.Silberschatz, R.B.Kieburtz, A.J.Bernstein, IEEE Transactions on Soft.Eng. 1977, 3/

CDL-NYELVŰ PROGRAMFEJLESZTŐ RENDSZER PDP-11 SZÁMITÓGÉPEN

Hanák Péter - Rácz Gábor
Budapesti Müszaki Egyetem
Müszer és Méréstechnika Tanszék

Bedő Árpád - Laborczi Zoltán
KSH Számitógépalkalmazási Kutató Intézet
Programozási Rendszerek Főosztálya

Budapesti Müszaki Egyetem
Müszer és Méréstechnika Tanszék

Bevezető

Cikkünkben ismertetjük azokat az okokat, amelyek a hatékonyabb programkészitési módszerek keresésére és megteremtésére késztettek. Ezután összefoglaljuk az "események krónikáját": elmondjuk, hogy milyen nehézségekkel találkoztunk, illetve milyen kedvező tapasztalatokat szereztünk a Honeywell-Bull 60/66 számitógépen futó és e gépcsalád makróassembly nyelvén megirt CDL-forditónak a PDP-11 számitógép RT-11 operációs rendszerébe való áthozatalakor. E munka sikere alapján a makrónyelvet a továbbiakban a processzorfüggetlen programirás egyik segédeszközének tekintjük. Végül azokról a továbbfejlesztési elképzeléseinkről számolunk be, melyek eszközeinket és módszereinket programfejlesztő rendszerré fogják tenni.

I. Igények és inditékok

A BME Müszer és Méréstechnika Tanszéke egy évtizede foglalkozik kisszámitógépek és mintegy öt éve mikroszámitógépek és mikroprocesszorok alkalmazásával.

Kutatási és fejlesztési feladatainkat elsősorban az jellemzi, hogy a processzorhoz általános és speciális célokat szolgáló készülékeket illesztünk. Az ezekből kialakitott rendszerek működtető programjait - megfelelő magasszintü nyelv hiján - általában valamilyen assembly nyelven kellett megirnunk, s nem ritkán a keresztforditókat is nekünk kellett elkészitenünk.

Programozási feladatainkat az alábbiak jellemzik:

1. Esetről-esetre változó, speciális hardware-t kell kiszolgálni.

2. A programok kifejlesztéséhez legkönnyebben a tanszéken található számitástechnikai eszközöket /TPAi, PDP11-45/ használhatjuk.

3. A kifejlesztett rendszerek egyediek, kevés kivétellel csupán egyetlen példányban készülnek.

4. Az alkalmazott processzor tipusa függ
   - az esetleges megrendelő igényeitől,
   - a beszerzési lehetőségektől,
   - a megvalósitandó feladat által támasztott technikai igényektől.

5. Általában nem egy meglévő operációs rendszer alatt futó programcsomagot, hanem egy teljes, önálló rendszert kell elkészitenünk.

Megvalósitott munkáinkból az alábbi tapasztalatokat szürhetjük le:

1. Az egymástól igen eltérő célu programokban igen sok az azonos részfeladatot megvalósitó programrész, pl. szám-

és kódkonverziós rutinok, szöveg-, táblázat- és pufferkezelő rutinok. A különböző processzorok alkalmazása és a programok eltérő belső környezete miatt azonban a szerzett tapasztalaton kivül szinte semmit sem lehetett átmenteni a régebbi programokból az ujabbakba, még kevésbé az egyik programozó munkájából egy másikéba.

2. A programok igen erősen hardware-függőek, ami az assembly nyelvek alkalmazását megköveteli. Ennek ellenére a programok szerkezete, a kidolgozott algoritmusok nagy része független a hardware-től.

3. Az alkalmazott programnyelvek sokfélesége lehetetlenné tette egységes dokumentációs előirások kidolgozását, igy programjaink általában rosszul dokumentáltak, és csak elkészitőjük számára érthetőek. Ez még tovább neheziti a programok és algoritmusok átvételét.

4. Mint emlitettük, más lehetőség hiján néhány processzor assembly nyelvét, illetve a megfelelő keresztforditót saját magunknak kellett specifikálnunk, illetve elkészitenünk. Mivel ezekre általában csak egyetlen, vagy kevésszámu esetben volt szükség, ezért kidolgozásukra nem volt érdemes tul sok energiát forditanunk. Igy a keresztforditók éppen csak a legszükségesebbeket tudták, és sem a programok megirásához, sem a hibák felderitéséhez, sem a dokumentáláshoz nem nyujtották a korszerü assemblerektől elvárható segitséget /pl. feltételes forditás, makrók, szövegkezelés, szótárkészités, fejléccel, dátummal, lapszámmal ellátott programlista/.

E tapasztalatok arra késztettek bennünket, hogy a haté-

konyabb programfejlesztést elősegitő nyelvek és módszerek után nézzünk. Igy esett a választásunk a SzÁMKI-ban és másutt már sikerrel alkalmazott CDL nyelvre. Döntésünket az alábbiak motiválták:

1. A CDL un. nyilt nyelv: a program szerkezete magas szintü nyelven, a végrehajtandó utasitások tetszőleges nyelven, pl. assembly szinten irhatók meg.

2. A CDL nyelven irt programok hordozhatóak: ugyanaz a /hardware-től független/ programrészlet - esetleg korlátozások betartásával - több gépen is futtatható.

3. A CDL nyelv alkalmazása elősegiti a felülről-lefelé való programtervezést és programirást.

4. A CDL nyelven irt programok mások által is jól olvashatóak, igy dokumentálásra is alkalmasak.

Bár Magyarországon CDL-forditó több számitógépen fut, tanszékünk nem rendelkezett ilyen géppel. Tapasztalataink azt mutatták, hogy a munkáinkhoz szükséges programok elkészitését olyan gépre célszerü alapoznunk, amelyhez könnyen férhetünk hozzá, és amely a közelünkben van. Ezért ugy döntöttünk, hogy elkészitjük a CDL-forditónak egy, a tanszéken üzemelő PDP11-45 számitógép RT-11 operációs rendszere alatt futó változatát. E döntésünket alátámasztó egyéb indokaink:

1. A nemzetközi MSzR-programot is figyelembe véve a PDP11 gépcsalád programkészletének fejlesztése indokolt.

2. A fejlesztést nem kell előről kezdeni, mivel a Honeywell-Bull /HwB/ 60/66 gépcsalád makróassembly nyelvén megirt és majdnem belőtt forditó legtöbb modulja szinte változtatás nélkül áthozható a PDP11-re. A HwB/CDL-forditó el-

készitői ugyanis abból a felismerésből indultak ki, hogy egy fejlett makróassembler a makrókifejtés idején képes magasszintü nyelv forditóprogramjához hasonlóan viselkedni. Ehhez csupán arra van szükség, hogy a megvalósitandó nyelv nyelvi fogalmait egy alkalmasan megválasztott makrókészlettel fejezzük ki. Ha már egyszer létrehoztak egy ilyen makrókészletet - azaz van ilyen módon megirt "forditóprogramunk" -, akkor a makrótörzsek átirásával a nyelv egy tetszőleges másik gépre átvihető. Ennek feltétele, hogy a célgép memóriája elegendően nagy legyen, és a szokásos szolgáltatásokat nyujtó, fejlett makróassemblerrel rendelkezzék /amely pl. tartalmaz a makrók aktuális paramétereinek tulajdonságait vizsgáló, feltételes direktivákat/. Ilymódon a CDL-fordító áthozatala egyuttal érdekes kisérlet is a programok makrónyelven való hordozhatóságára.

3. A SzÁMKI munkatársaival való közös munka kapcsán a tanszék programozóinak lehetősége nyilik mások tapasztalatának, módszereinek átvételére.

4. A CDL-fordító megvalósitása közben alaposan megismerjük a fordító belső felépitését és müködését, igy a továbbiakban képesek leszünk a CDL-fordító továbbfejlesztésére, illetve figyelembe tudjuk venni azt a követelményt, hogy a CDL nyelven megirt programokat többféle processzoron kell majd futtatnunk.

II. A fordító áthozatala

A CDL-fordító áthozatalára irányuló munka 1977. márciusában kezdődött el, nem tul feszes ütemben, s első fázisával 1978. márciusában lettünk készen: ekkor forditottuk le és futtattuk a PDP11-en az első CDL-programokat.

Az elvégzett munka az alábbi részfázisokra bontható:

1. 1977. március - május:

Tanfolyam a munkában résztvevők és az érdeklődők részére a CDL nyelvről és a CDL-fordító müködéséről.

2. 1977. junius - augusztus:

A CDL-fordító un. futtatórendszerének a megtervezése, a CDL-fordító által használt makrók PDP11/RT11 assembly nyelven való megirása és belövése.

3. 1977. szeptember - december:

A CDL-fordítót alkotó 12 modul átvitele HwB formátumu mágnesszalagról a PDP11 háttértárolójába, majd a programok szintaktikai hibáinak kijavitása. E "hibák" elsősorban a két rendszer és a két makróassembler különbözőségéből adódtak. Ez a részfázis volt a leginkább idő- és gépidőigényes. Érdemes figyelmet forditani a legtöbb gondot okozó eltérésekre /lásd az 1. táblázatot/.

Következő lépésként meg kellett terveznünk, majd megirnunk a fordító un. Basic Input Output /BIO/ modulját, hiszen a HwB gép operációs rendszere és a PDP11 RT11 operációs rendszere, mint programkörnyezet is jelentősen eltér egymástól. A legnagyobb gondot talán most is az okozta, hogy a CDL-fordító belső "kártyaformátumát" és az RT11 CR-LF-fel lezárt, változó hosszuságu sorát kellett egymással összhangba hozni.

Ezután a fordító szemantikai hibáit szürtük ki. A hibák nagy része a processzortól, illetve a kártyaformátumtól függő konstansok helytelenül beállitott értékére volt visszavezethető. Sok problémát okozott az is, hogy a PDP11 számi-

	HwB	PDP-11
1. kód	EBCDIC	ASCII
2. sorformátum	80 oszlopos kártya, formátumvezérlő karakterek /CR,LF,HT/ nélkül	0-80 karakter hosszu sorok, a sor végét CR-LF pár jelzi, HT megengedett
3. cimke	a kártya 1-6 oszlopában lévő alfanumerikus karakterek	1-6 alfanumerikus karakter, amit : követ
4. megjegyzésjel	* /csillag/	; /pontosvessző/
5. assembler-aritmetika	mindig decimális /a CDL-fordító tehát mindig decimális számokat használt/	oktális és decimális /a PDP-hagyományok miatt a makrókat - sajnos - az oktális számrendszerben irtuk meg/
6. számkonstansok	nincs speciális jelölésük	előzi meg, egyébként cimet jelent
7. . /pont/	közönséges szimbólumként használható /és használt!/	kitüntetett szimbólum, az elhelyezésszámlálót jelenti

1. táblázat A HwB és a PDP-11 makróassemblerének szintaktikai eltérései

tógépen az utasitások és a szó-operandusok cime csak páros szám lehet. Külön gondot jelentett az, hogy az eredeti HwB/CDL-forditó az áthozatal előtt nem volt teljesen belőve, igy saját programjaink hibái mellett más programozók programjainak hibáit is ki kellett javitanunk. A kiindulásul szolgáló CDL-forditót dicséri viszont az, hogy a két gép szóhosszuságának eltérése /HwB: 36 bit/szó, PDP11: 16 bit/szó/ nem okozott különösebb nehézséget.

1977. végén már rendelkezésünkre állt egy 13 modulból álló olyan CDL-forditó, mely az RT11 operációs rendszer alatt futott, de még a HwB makróassemblere számára szóló kódot állitott elő. Ez a kód sem volt hibátlan, de ugy döntöttünk, hogy a szintaktikai és a szemantikai hibák kiszürését későbbre halasztjuk.

4. 1978. január - március:

A CDL-forditó elindulása után a munkát gyorsabb ütemben folytattuk. Első lépésként az un. gépfüggő, illetve célnyelvfüggő modulokat terveztük és irtuk át. Ilyen volt a már korábban megirt BIO modul, s ilyen a "Semantics" /SEM/, azaz a kódgeneráló, a "Macro Definitions" /MDF/, azaz a mekrókat definiáló és a "Process Macros" /PMC/, azaz a makrókat kifejtő modul is.

Mivel a makrónyelvvel, mint a programok hordozhatóságának egyik eszközével szerzett tapasztalataink igen kedvezőek voltak, ugy döntöttünk, hogy a CDL-forditó által generált kód egy futtatórendszer-független makrókód legyen. Az uj SEM modul tehát, ahelyett, hogy - amint az szokás - valamilyen konkrét nyelvre forditana, a CDL-szintaxis egyes elemeinek megfeleltethető makróhivásokat állit elő és lát el a szüksé-

ges aktuális paraméterekkel. Egy ilyen rendszerben a konkrét futtatórendszert ezeknek a makróknak a - processzorról processzorra változó - törzse fogja tartalmazni.

Igy e modulok áttervezéséhez elsősorban a PDP11 makróassemblerének sajátosságait kellett figyelembe venni. Mivel a CDL-forditóval több processzoron - igy különféle tipusu mikroprocesszorokon is - futtatható kódot akarunk előállitani, a generált makrókód kialakitásakor egyszerüségre és általánosságra törekedtünk, annak érdekében, hogy az egyes /mikro/processzorokhoz alkalmazandó /és várhatóan megirandó/ kereszt-makróassemblerek létrehozása ne legyen majd nehéz feladat.

Következő lépésként azt a két programösszetevőt készitettük el, melyek a célnyelven tul az alkalmazott processzortól is függenek: a CDL-makrókönyvtárat és a rendszermakró-definiciókat /egyelőre a PDP11/45-re/. Végső soron ezek a komponensek határozzák meg a generált kódot, illetve adják a CDL-programok futtatásához szükséges környezetet.

E részfázis lezárásával 1978. március végéig befejeződött a CDL nyelv PDP11-re való átvitelének első fázisa is. E módszerrel az áthozatalhoz - a szintaktikai hibák kijavitásán tul - a CDL-forditó 20-25%-ának átirására volt szükség.

III. Továbbfejlesztés

CDL-nyelvü programfejlesztő rendszerünk létrehozásának második fázisában további modulokkal bővitjük a CDL-forditót.

A "Command Analyzer" /CA/, azaz a parancsanalizáló modul feladata az, hogy a forditással kapcsolatos kivánságokat /az

input-output file-ok kijelölését, a listázási előirásokat, a forditási opciókat stb./ az operátor a PDP-gépek operációs rendszereiben szokásos módon adhassa meg.

A "Diagnostics" /DG/ és a "Trace" /TR/, azaz a diagnosztizáló és a nyomkövető modulok a CDL-nyelvü programok belövését és hangolását /a gyakran futó programrészletek megkeresését és optimalizálását/ segitik elő. Alapelvünk, hogy a programról minden információt a CDL nyelv fogalmaival kifejezve bocsássunk - lehetőleg interaktiv üzemmódban - a programozók rendelkezésére.

E modulok létrehozásával a CDL-nyelvü programok elkészitését kényelmesebbé és gyorsabbá tesszük.

Továbbfejlesztési terveink másfelől azt tüzik ki célul, hogy a PDP11-en futó CDL-forditóval más processzorokra, igy mikroprocesszorokra is tudjunk programokat fejleszteni. Mint emlitettük, olyan makrónyelvet terveztünk, mely elég általános ahhoz, hogy több processzorra közösen használhassuk /lásd az 1. ábrát/. Mivel a makrónyelv közös, egy ujabb processzor felbukkanása esetén csak a processzorfüggő részeket kell ujakra cserélni. Ezek a következők /lásd a 2. ábrán a CDL-nyelvü programfejlesztő rendszer müködésének sémáját/:

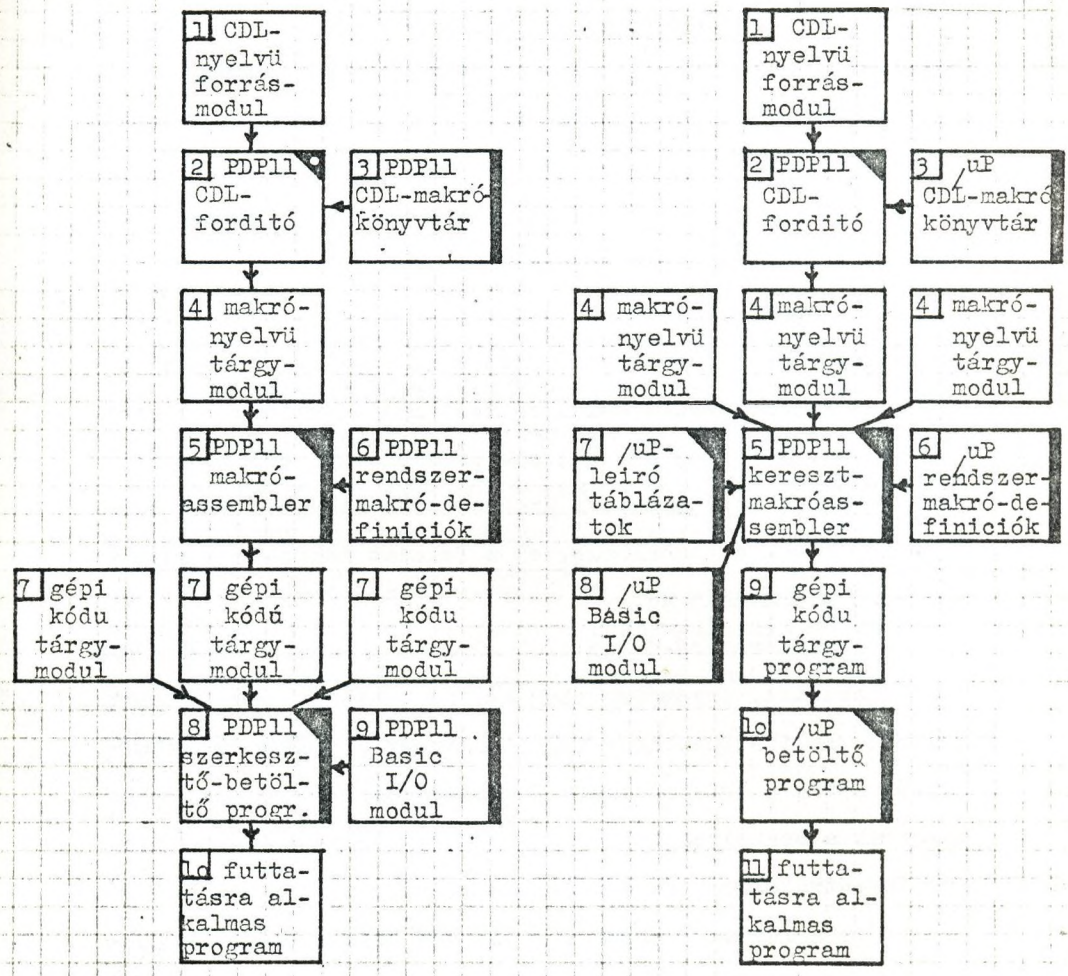
1. Basic Input Output /BIO/ modul,
2. CDL-makrókönyvtár,
3. rendszermakró-definiciók /= futtatórendszer/,
4. makróassembler.

Vegyük sorra, hogy milyen feladatot is jelent a fentiek elkészitése!

A BIO modul a célgép operációs rendszerével való kapcsolatot valósitja meg, ha egyáltalán van a célgépnek ope-

```
    1 'MODULE' EXAMPLE,

    2 'DEFINES''ACTION' DISPLAY LINE,

    3 'APPLIES''PREDICATE' OUTREC,

    4 'COMMON''POINTER' GIVE TEXT:6,

    5 'BODY' 'SWITCH' 'EI',
    6 'NOLIST'

  207 ? A SZABVANYOS MAKROKONYVTAR HOSSZA: 200 SOR ?
  208 'SWITCH''SI',
  209 'POINTER' LINE,

  210 'LIST' LPBUFF(0:40),

  211 DISPLAY LINE+OUTFILE-LP:
  212   INCR+LINE,IS+GIVE TEXT,MAKE+LP+OUTFILE,OUTREC+LP+LPBUFF;
  213

  214 'ELUDUM'

        $PROGS  EX       ;    1
        $MODHE  EXAMPL,EXAMPLE  ;    2
        $DEFIN  DISPLA,ACTION,DISPLAYLINE        ;    3
        $APPLI  OUTREC,PREDICATE,OUTREC ;    4
        $DATAS  EX       ;    4
        $COMDE  EX1JE,6,GIVETEXT         ;   10
        $PROGS  EX       ;   10
        $CONSD  EX1KO,0,TRUE     ;   10
        $CONSD  EX1LU,1,FALSE    ;  209
        $DATAS  EX       ;  209
        $GLOBD  EX39K,LINE       ;  210
        $VECPL  0,40,LPBUFF      ;  210
        $PROGS  EX       ;  210
        $VECAD  EX3AM,40,LPBUFF ;  211
        $SHEAD  DISPLA,ACTION,DISPLAYLINE        ;  211
        $KOTRA  1,OUTFILE        ;  211
        $SZRAG  2,LP     ;  212
        $LOCAL  1        ;  212
        $HEADE  1,1      ;  212
;INCR
        INC     @#EX39K ;  212
;IS
        TST     12+C,MBEG
        BNE     EX3,     ;  212
;MAKE
        MOV     @6(%6),2(%6)     ;  212
        $BLEPS  OUTREC,PREDICATE,OUTREC ;  212
        $BLPLO  2,1,2,LP         ;  212
        $BLPVE  EX3AM,2,LPBUFF   ;  212
        $BLPEN  OUTREC,PREDICATE,OUTREC ;  212
        $JMPGE  EX3,     ;  213
        $JMPGE  EX2,     ;  213
        $LABGE  EX3,     ;  214
        $LABGE  EX2,     ;  214
        $ENDSZ  1,1      ;  214
        $MODEN
```

1. ÁBRA EGY CDL-PROGRAMRESZLET ÉS GENERÁLT KÓDJA



2. ábra A CDL-nyelvü programfejlesztő rendszer müködésének sémája

rációs vagy monitorrendszere. Ha nincs - és a mikroprocesszorokra általában ez jellemző -, akkor a BIO modulnak az elemi I/O müveleteket is létre kell hoznia, azaz meg kell tervezni a célgép I/O rendszerét, esetleg CDL nyelven, több processzorra is alkalmas formában. Igy a BIO modul megirása esetenként elég sok munkát jelenthet.

A CDL-makrókönyvtár átvitele azt jelenti, hogy a felhasználók által is elérhető, általában egyszerü makrókat a célgép utasitáskészletével ujra kell irni. Tekintettel arra, hogy a makrókönyvtár nem tul sok és elég egyszerü makróból áll, továbbá, hogy a modern processzorok alaputasitáskészletében elég sok a hasonlóság, ez a feladat inkább mechanikusnak mondható munkát ad.

A rendszermakró-definiciók átirása a csak a CDL-forditó által használt, ugyancsak elég egyszerü makrók lecserélését jelenti. Emlitettük, hogy tulajdonképpen a rendszermakrók hozzák létre a CDL-programok futtatórendszerét, azaz meghatározzák a szabályok be- és kilépésének, a paraméterek átadásának-átvételének és a változók helyfoglalásának a rendjét is. Ezért a rendszermakrók elkészitése befolyásolja legjobban a célrendszer hatékonyságát.

Végül szóljunk néhány szót a makróassemblerekről. A számunkra érdekes mikroprocesszorok /Intel-8o8o, Zilog-8o, Motorola-68oo/ egyikéhez sem sikerült külső forrásból megfelelő makróassemblert beszerezni. Ezért kézenfekvő megoldásnak látszott, hogy egy olyan, közös makróassemblert tervezzünk és irjunk meg, amely azonos szintaktikáju utasitáskészletet és makródefiniciókat fogad el, a gépi kódu tárgyprogramot pedig a kérdéses processzor utasitáskészletének és

tulajdonságainak táblázattal való megadásából állitja elő.

A programfejlesztő rendszer továbbfejlesztésének következő szintjét a célrendszerekben alkalmazható hibakereső és hibajavitó eszközök megteremtése jelenti, ez azonban nem tartozik a jelen cikk tárgyához.

Irodalom:

1. Havass Miklós: A nagyüzemi software-gyártás és az automatizált rendszertervezés és -szervezés módszereinek és eszközeinek kutatása-fejlesztése és létrehozása /Koncepcióterv/ SOFTTECH sorozat, D1 kötet, SzÁMKI, 1977.
2. A SOFTTECH /Software Technológia/ sorozat további kötetei /2-től 18-ig/ SzÁMKI, 1977-78.
3. CDL programkészitő rendszer a Honeywell-Bull 60/66-os számológépen /Felhasználói kézikönyv/ SzÁMKI, 1976.

Budapest, 1978. május 15.

REAL-TIME MONITOROK GEOFIZIKAI FELDOLGOZÁSOK TÁMOGATÁSÁRA

Horváth József
Számitógépalkalmazási Kutató Intézet

Összefoglalás

Geofizikai mérési adatok real-time feldolgozásához speciális perifériális berendezésekkel kibővitett Rlo rendszert fejlesztettek ki. A tanulmány a speciális konfigurációt vezérlő real-time monitorokat ismerteti. A célberendezések programozott kezelését úgy valósitottuk meg, hogy a monitor felhasználói és operátori interface-e kompatibilis legyen a szabványos real-time monitorokéval /RTDM/. A geofizikai feladatok és a real-time környezet számos olyan hardware funkció kialakitását tette szükségessé, amelyek kezelése nem oldható meg az Rlo konvenciók alapján. Ugyancsak a real-time feldolgozás követelményei teszik élessé a szinkronizálási kérdéseket és a deadlock elleni védelmet. A továbbfejlesztés lehetőségei között szerepel minimális központi tár igényü program futtató monitor kialakitása, ugyancsak a kompatibilitás megtartásával.

1. Bevezetés

A Magyar Állami Eötvös Lóránd Geofizikai Intézetben /ELGI/ mérési eredmények real-time feldolgozása érdekében speciális perifériális egységeket fejlesztettek ki, amelyeket Rlo számitógéphez csatoltak. A számitógép terepesitett kivitelben hajón vagy tehergépkocsiban is dolgozhat.A konfigurációnak nagy mennyiségü adatot /6 sec alatt bejövő, mintegy 3oo KByte/ kell viszonylag rövid idő alatt /12 sec/ feldolgoznia, meglehetősen összetett algoritmusok alapján. A feldolgozó rendszer három komponensből tevődik össze:

- speciális hardware konfiguráció,
- monitorok,
- felhasználói programok.

Az expressz feldolgozást az Rlo real-time diszkes monitorok /RTDM/ hatékonyan támogatják [5], [6], mivel lehetővé teszik a multiprogramozást és a perifériák egyidejü müködtetését is. A speciális berendezések kezelő rutinjait /handlereit/ a SzÁMKI-ban dolgoztuk ki és épitettük be a real-time monitorokba.

A real-time környezet szabta hatékonysági követelményeknek némiképp ellentmondva, törekedtünk a standard perifériákkal való minél nagyobb mértékü kompatibilitás elérésére. A rendszer kifejlesztői azonban a geofizikai feldolgozás meggyorsitása érdekében olyan funkciókat is megvalósitottak a hardwareben, melyek a szokásostól eltérő programozott kezelést igényelnek.

A munka kapcsán értékes gyakorlati tapasztalatokra tettünk szert, a real-time környezetben történő alkalmazás olyan problémák megoldását tette szükségessé, mint erőforrás kezelés, szinkronizálás és deadlock. [1] , [2], [3]. A real-time programozás különösen élesen veti fel az időzitési kérdéseket [4], a külső körülmények változása a programok viselkedését is megváltoztathatja /időfüggő programok/. A továbbiakban elsősorban ezen problémák közül emelünk ki néhányat.

A tanulmány megirása során törekedtünk arra, hogy az Rlo számitógépek hardware, illetve software rendszerének ismerete nélkül is érthetőek legyenek a felmerült problémák és azok megoldásai. Aki azonban az Rlo rendszert jól ismeri, a leirtakat könnyen átültetheti Rlo terminológiára.

2. Konfiguráció

Az egyes hardware egységek részletes ismertetése nélkül, csak a monitorok kifejlesztése szempontjából érdekes néhány részproblémát emelünk ki.

A speciális perifériák többsége a nagy sebesség érdekében közvetlen memória hozzáférésü /DMA/ csatolóval rendelkezik, vagyis az input adatokat közvetlenül a központi tárba irják, illetve az output adatokat onnan olvassák ki.

Minthogy egy-egy átvitel során gyakran a központi tár kapacitásának többszöröse kerül folyamatosan beadásra, vagy kivitelre, úgynevezett váltott pufferes transzfert valósitottak meg a különböző tipusú beadó egységeknél és a mágnesszalag csatolóban. A periféria két, azonos méretü pufferből /vagy pufferbe/ dolgozik felváltva, és megszakitással jelzi, ha puffert vált /egyik puffert megtöltötte, illetve kiüritette már/, vagy ha befejeződött a transzfer.

A feldolgozás meggyorsitása érdekében a konfiguráció egy speciális processzort /aritmetikai egységet/ is tartalmaz, amely nagy méretü tömbökön végez fixpontos számitásokat, pl. skálázást, normál korrekciót, rekurziós vagy konvolúciós szürést. A szürőfüggvény értékeit, illetve a részeredményeket a speciális processzor saját belső tárolója őrzi.

A multiplexált formában bejövő mérési adatokat az igen gyors, DMA csatolású, fixfejes diszk segitségével lehet demultiplexálni és pufferelni a további feldolgozásig.

A számitási eredmények plotteren jelennek meg /ez közvetlenül az Rlo minibuszra csatlakozik, mikroprogramozott vezérlésü/. Ez a periféria egy rajzolási utasitás hatására 3 KByte adatot jelenit meg, ehhez azonban /a sok mechanikus mozgás miatt/ néhány másodpercre van szükség. Minthogy az Rlo számitógépek központi tára a geofizikai alkalmazásoknál szük keresztmetszet, a plotter az adatokat először saját belső tárolójába viszi át, és a tényleges rajzolás ideje alatt már nem igényel központi memória helyet.

A konfigurációhoz tartozó standard perifériákkal itt nem foglalkozunk.

3. Monitorok

A Szeizmikus Monitorok /SZEM, SZEMB, SZEMF/ [7], [8] mind az operátori parancsok tekintetében, mind pedig a felhasználói programok felé nyújtott szolgáltatásaikban kompatibilisek a szabványos real-time diszkes monitorokkal /RTDM/. Tartalmazzák azonban a speciális perifériák kezeléséhez szükséges handlereket és táblázatokat is, ily módon lehetővé téve a speciális perifériáknak a hagyományosokkal megegyező, szabványos kezelését. A perifériák kompatibilitása a következő példákkal illusztrálható:

- a DMA mágnesszalag egység alkalmas szabványos formátumú szalagok /programkönyvtárak, adatok/ létrehozására és beolvasására is;

- a DMA diszk egyszerü irás-olvasás tekintetében a rendszerdiszk adatzónájával azonos módon használható;

- a plotterre szánt output parancsok a periféria hozzárendelés megváltoztatásával mágnesszalagra vagy lyukszalag lyukasztóra is végrehajthatók.

A Szeizmikus Monitorok három változata a következő:

SZEM: program futtatásra alkalmas, kis tárfoglalású monitor,
SZEMB: programok belövését segitő eszközöket is tartalmazó monitor,
SZEMF: fentieken kivül szekvenciális adat-file-ok kezelését is lehetővé teszi.

Tervezzük a program futtató monitor tárfoglalásának további csökkentését is, a futtatáshoz szükséges valamennyi lényeges funkció és a kompatibilitás megtartásával.

A továbbiakban a speciális perifériák kezelésével kapcsolatos néhány kérdést részletezünk.

3.1 Eseménykezelés a plotter használatakor

A plotter egy transzfer elinditása után - a hibás müködéstől eltekintve - mindig két lépésben ad visszajelzést a monitor, és azon keresztül a felhasználói program felé:

1. Befejeződött a rajzolandó adatok átmásolása a plotter belső tárolójába, a központi tárban lévő puffer terület más célra felhasználható.

2. Befejeződött a rajzolás, a plotter új parancs fogadására készen áll.

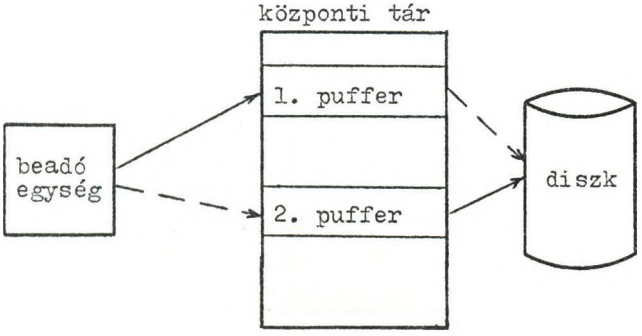
A standard perifériák kezelésénél a monitor egy esemény /lényegében bináris szemafor/ foglaltra, majd szabadra állitásával jelzi egy transzfer kezdetét és befejeződését. A felhasználói programból ezen esemény állapota tesztelhető, illetve szabad állására lehet várakozni.

A plotter esetében az eseményt két lépésben kellene szabadra állitani, ilyen eszközt azonban nem adnak az Rlo monitorok. Megtehetjük, hogy a felhasználói programra bizzuk az esemény másodszori foglaltra állitását /monitor szolgáltatások megkerülésével/, de ez a program blokkolódását is eredményezheti, ha a második szabadra állitás időben előbb bekövetkezett és észrevétlen maradt.

A választott megoldásban eltértünk az Rlo konvencióktól, és két eseményt kapcsoltunk a plotter müködtetéséhez. Mindkettőt a monitor állitja foglaltra a transzfer elinditásakor, szabaddá tételük pedig a fentieknek megfelelően két különböző időpontban történik. Ez a megoldás biztositja a programok blokkolódás nélküli müködését.

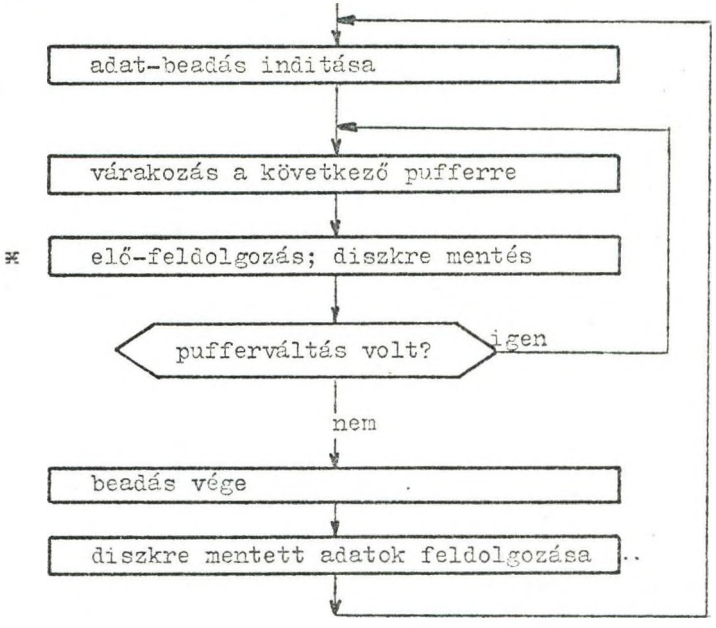
3.2 Szinkronizálási problémák váltott pufferes átvitelnél

A viszonylag rövid idő alatt bejövő nagy mennyiségü mérési adatot a további feldolgozás céljára diszken kell pufferelni. A váltott pufferes beadást és az adatok diszkre mentését az alábbi ábra szemlélteti:



Mialatt a beadó egység az egyik puffert tölti, a másiknak a tartalmát - esetleg némi elő-feldolgozás után - diszkre mentjük. Könnyen belátható, hogy az outputnak biztosan rövidebb idő alatt kell lezajlania, mint egy puffer feltöltődése, különben a beadó egység /amelynek a sebessége nem befolyásolható/ "utoléri" az elmentést, és igy adatot veszithetünk el. Nem biztonságos például egy mágnesszalagról másik mágnesszalagra váltott pufferesen másolni, mivel az egységek sebessége közötti kis eltérés is szinkronizálási hibához vezethet, a program viselkedése időfüggővé válhat.

Tipikus feldolgozást szemléltet a következő blokkséma-részlet /a hibakezelést itt nem részletezzük/:



Amennyiben a *-gal jelölt munkafázis végrehajtási ideje hosszabb egy puffer feltöltési idejénél, adat-vesztés következik be. A beadó fix sebessége miatt ezen segiteni nem lehet, legfeljebb a hiba megtörténtét lehet jelezni.

A probléma bizonyos mértékben hasonló a plotter kezelésénél leirtakhoz, de itt egy transzfer elinditása után többször kell az eseményt szabadra állitani. Nyilvánvalóan képtelenség ennyi különálló eseményt bevezetni. Minthogy a korrekt megoldási lehetőségek meglehetősen bonyolultak /és éppen a kritikus idejü szakaszt terhelnék/, továbbá lényegesen eltérnek az Rlo monitorok filozófiájától, ebben az esetben a felhasználói programra biztuk az esemény újbóli foglaltra állitását, valahányszor úgy találja, hogy még nem fejeződött be a transzfer, tehát az esemény újabb szabadra váltását várhatja a monitortól. Ez a módszer azonban - mint kifejtettük - a programok blokkolódását is kiválthatja, az időzitési viszonyoktól függően.

Az eddigi alkalmazások azt mutatták, hogy a szinkronizálási hiba /adat-vesztés/ gondos programozással elkerülhető. További megfontolások és futási idő mérések alapján dönthető csak el, hogy milyen mértékü védelmet, illetve hiba jelzést érdemes a monitorba beépiteni.

3.3 A speciális processzor, mint erőforrás

Egy müvelet - például konvolúciós szürés - végrehajtásához a speciális processzornak két parancsot kell kiadni: először a szürőfüggvény értékeit a belső tárolójába kell tölteni, majd a számitást kell elvégezni. Multiprogramozási környezetben nyilvánvalóan számitási hibához vezethet, ha ezen két lépés végrehajtása közé beékelődik egy másik programból kiadott parancs, amely megváltoztatja a belső tároló tartalmát. Biztositani kell tehát, hogy a szürőfüggvény betöltésétől a számitások befejeztéig a speciális processzor egyetlen program "kizárólagos tulajdona" legyen. Ezt lehetővé teszi a monitorok által szolgáltatott erőforráskezelés, amelynek segitségével a fenti hiba elkerülhető. A speciális processzort erőforrásnak tekintve és lefoglalva, azt másik programból lefoglalni mindaddig nem lehet, mig pillanatnyi tulajdonosa fel nem szabaditja.

4. Felhasználói programok

A bevezetőben emlitett harmadik rendszer-komponenst az egyes alkalmazásokhoz illesztve, a felhasználónak kell elkészitetenie. Minthogy ezek a rendszerek sok hasonló feldolgozási rész-funkciót tartalmaznak, a jövőben elképzelhető ezek egy részének a monitorokba beépitése is.

5. Köszönetnyilvánitás

Köszönet illeti a rendszert kifejlesztő valamennyi munkatársat az ELGI-ben, különösen Korvin Gábort és Rácz Istvánt. A monitorok megtervezésében és elkészitésében a SZÁMKI-ban jelentős részt vállalt Békéssy Péter, Kozma László és Simonfai László.

Irodalomjegyzék

1. Per Brinch Hansen: Operating System Principles
/Prentice-Hall, 1973./

2. E.G. Coffman, P.J. Denning: Operating Systems Theory
/Prentice-Hall, 1973./

3. James Martin: Design of Real-Time Computer Systems
/Prentice-Hall, 1967./

4. Niklaus Wirth: Toward a Discipline of Real-Time Programming
/Communications of the ACM, Vol. 2o. No. 8., pp. 577-583/

5. Földvári I., Trencsényi I., Vincze S.: RTDM monitor és BATCH parancsláncoló modul - Felhasználói kézikönyv
/INFELOR, 1973./

6. Földvári I., Trencsényi I., Vincze S.: RTDM monitor és BATCH parancsláncoló modul - Fejlesztői kézikönyv
/INFELOR, 1973./

7. Horváth J.: Szeizmikus Monitor - Rendszerterv
/SZÁMKI, 1976./

8. Békéssy P., Horváth J., Kozma L.: Szeizmikus Monitorok - Felhasználói kézikönyv
/SZÁMKI, 1977./

JOB-ACCOUNT /MUNKAELSZÁMOLÁS/ KIÉRTÉKELŐ RENDSZERÉNEK FEJLESZTÉSE A SIEMENS 4004/BS 1000 OPERÁCIÓS RENDSZERÉBEN

Jenei Gyula
Állami Fejlesztési Bank

A korábbi operációs rendszer verziókhoz mérten a Siemens AG a BS 1000 1.3-as verzióban egy lényegesen fejlettebb Job-Account rendszerrel jelentkezett. Az uj Job-Account mindazon /elemi/ adatokat szolgáltatja a felhasználónak, melyek szükségesek egy korszerü munkaelszámoláshoz, illetve a gépleterhelés méréséhez. A JBACOUNT rendszerprogram a SYSACNT nevü lemez file-ba helyezi el a programok futtatása során nyert információkat, amelyből a JAEDTØØ rendszerprogram archiválja, illetve menti ki a kiértékelések kiinduló pontjául szolgáló JBACOUNT file- t.

A BS 1000 V1.3 kétféle kiértékelést tesz lehetővé - a JAEDTØ1 és a JOBLOG3 rendszerprogramok által - ezek azonban - részben a JBACOUNT file szerkezetéből adódóan - nem teszik lehetővé nagyobb időszakokra /pl. dekádra, vagy hónapra/ kiterjedő együttes, összesitett kiértékelést. A Siemens kiértékelő rendszere u.is csak un. SESSION-okra képes összesitéseket adni.. A SESSION a JBACOUNT töltéstől /általában a rendszertöltéstől/ a JBACOUNT-futás befejezéséig /általában a rendszer kilövéséig/ terjedő időszakot foglalja magába, vagyis egy megszakitás nélküli folyamatos gépi üzemi időszakot. Ez a gyakorlatban - két müszakos üzemeltetésnél - 1-2 müszaknyi /6-2o órás/ időtartamot jelent.

Intézményünknél - de más számitóközpontokban is - felmerül az a jogos igény, hogy a Job-Account kiértékelő rendszere tetszésszerinti naptári időtartamokra szolgáltasson összesitett kiértékelést. Nálunk pl. követelményként merült fel a havi szinten összesitett munkaelszámolás, a gépkihasználás, illetve a géplóterhelés mérése.

Ezt a feladatot oldottam meg a SESSION1 nevü /Assembler nyelven irt/ program megirásával, amely 1978 februárjában készült el.

I. SESSION1 program által nyujtott szolgáltatások

A program az alábbi három fő feladatot oldja meg:

1./ JBACOUNT-file átalakitása

A bevezetőben utalás történt arra, hogy a Siemens Job-Account kiértékelő programjai az egyes SESSION-okat egymástol függetlenül, elszigetelten képes feldolgozni, ill. kiértékelni. Több SESSION együttes kiértékelésére a Siemens programjai nincsenek felkészitve. E problémát ugy hidalja át a SESSION1 program, hogy a több SESSION-t tartalmazó JBACOUNT file-nak a kiértékeléshez szükséges valamennyi érdemleges adatát egyetlen SESSION keretébe ágyazva adja ki az output szalagon. A SESSION1 program által átalakitott - egyetlen SESSION-ba integrált - JBACOUNT file-t a rendszer kiértékelő programok már éppugy fel tudják dolgozni, mint bármely más eredeti Job-Account SESSION-t. A JBACOUNT-file átalakitásának eredményeként fel tudjuk használni a Siemens Job-Account kiértékelő rendszerét teljes havi /dekád, ill.heti/ szinten összesitett munkaelszámolások készitésére.

2./ "JOB-ACCOUNT ÖSSZESITŐ" készitése

A SESSION1 program előállitja az 1.sz. mellékletben bemutatott Job-Account összesitőt,amely az alábbi számitott és gyüjtött időmérési alapadatokat, a feldolgozást és a gépleterhelést jellemző mutatókat, továbbá egyéb információkat tartalmazza.

CPU-TIME/központi egység idő/
Ez a központi egység feldolgozási idejének hónap szinten összesitett időadata.

- PROGRAM-TIME/programok összes futási ideje/
Ez az idő a tárgyhónapban futtatott összes programok teljes futási idejének összege. Megjegyzendő, hogy a JBACOUNT ilyen adatot a futtatott programokról nem szállit közvetlenül. A SESSION1 ezt az egyes programfutások inditási és befejezési időpontjainak különbségéből számitja ki.
- ACCOUNT-PERIOD/elszámolási, ill. megfigyelési időszak/
Az Account-Period a számitógép havi szinten összesitett üzemeltetési időtartamát reprezentálja. A Siemens kiértékelő rendszere is számit Account-Periodot, de csak az egyes Session-okra külön-külön, összesités, illetve összesithetőség nélkül. A rendszernek ezt a korlátozottságát a SESSION1 áthidalja és a JBACOUNT-file-ból kiszámolja a tárgyhónap során felhasznált összes gépi üzemidőt.
- A kiszámitott és összesitett időmérési alapadatokból az alábbi három mutatót képezi a program:

1. Multiprogramozás hatékonysági mutatója
amely a PROGRAM-TIME és az ACCOUNT-PERIOD hányadosa. Amennyiben a mutató értéke loo %, ugy átlagosan egy

program futott a megfigyelési időszakban. 100 % feletti mutatóérték utal a multiprogramozás adott átlagos szintjére.

2. Központi egység kihasználása

a CPU-TIME és az ACCOUNT-PERIOD hányadosaként kerül kiszámitásra. Ez a mutató következtetést enged arra vonatkozóan, hogy a tárgyidőszakban milyen jellegü - CPU, vagy I/O-igényes - programok futottak.

3. Központi egység programfutások alatti kihasználását

a CPU-TIME és a PROGRAM-TIME hányadosa adja. Ez a mutató lényegében az előzőnek a multiprogramozás hatékonysági mutatóval korrigált értékét adja.

- Egyéb információként szolgáltatja végül a Job-Account összesitő a nyomtatott sorok számát, valamint a felhasznált leporellólapok mennyiségét.

3./ "JOB-ACCOUNT ELSZÁMOLÁSI IDŐSZAKOK" c.lista készitése

A 2.sz. mellékletben bemutatott lista informálja a vezetést naponkénti tényleges gépórafelhasználásról/a müszakok kezdési és befejezési időpontjáról és időtartamáról/.

II. SESSION1 program müködési elve

A program gépi folyamatábráját a 3.sz. melléklet szemlélteti.

1./ JBACOUNT-file átalakitása/4.sz. melléklet/.

A program a SYS009-es szimbolikus periféria névvel jelzett JBACOUNT-file-ról átmásolja a SYS011-es outputszalagra a legelső Session Job-Account kezdő rekordját, valamint a konfigurációs rekordját és a legutolsó Session Job-Account befejező/záró/ rekordját. Minden további - közbeeső - Job-Account kezdő és záró rekordot, illetve a konfigurációs rekordokat kiiktatja.

Az egyes programfutásokra vonatkozó információkat tartalmazó rekordokat /kezdő, záró, I/O, SPOOL, Monitor, stb./ teljes körben atirja a program az outputra azzal a csekély változtatással, hogy a feladatszámot/Auftrags-Nummer/ a SESSION1 adja 1-töl egyesével folyamatosan, mivel ismétlődő feladatsorszám egy Session-on belül nem fordulhat elő.

2./ A JBACOUNT-file átalakitásával párhuzamosan a program gyüjti a CPU-időt, a nyomtatott sorok számát, a felhasznált leporelló mennyiségét és kiszámitja az un. ACCOUNT-PERIOD-ot az alábbiak szerint:

Minden egyes Session Job-Account kezdő és befejező rekodja előállitási dátumának és időpontjának különbségéből megállapitja az egyes Sessionok időtartamát, mint folyamatos üzemeltetési időtartamokat. Ezek összessége megadja a tárgyhónapra eső összes felhasznált gépidőt./A napváltásokat is korrektül kezeli/.

3./ PROGRAM-TIME /RUN-TIME/ számitás

A SYS011-en előállitott integrált JBACOUNT-file-rol leválassza a program a RUN-TIME számitáshoz szükséges JOB, ill. program kezdő és befejező rekordokat/ezek a rekordok tartalmazzák a futtatott programok töltési, illetve befejezési időpontjait/. A leválasztott állomany a meghivott SORT program által az alábbi szempontok szerint kerül rendezésre: 1. Feladatsorszám/Auftrags-Nr./

2. Programnév
3. Dátum
4. Óraidő

A rendezett állománybol a SESSION1 az egyes programok töltési és befejezési időpont különbsége alapján kiszámitja az egyes programok futási időtartamát /RUN-TIME/,

majd ezek összesitése alapján a megfigyelési időszakban futtatott valamennyi program összes futási idejét/PROGRAM-TIME/.

4./ A kiszámitott és összesitett adatokbol a program végezetül elkésziti a tárgyhónap teljes gépi üzemidejére vonatkozóan a már ismertetett Job-Account összesitőt, valamint a Job-Account elszámolási időszakok c. listát.

III. <u>További speciális rendeltetésü programok a Job-Account kiértékelő rendszer támogatásához/SELMER,RADIO,ACCOUNT3/.</u>

Az elmult időszakban - a SESSION1-en kivül - az alábbi három /szintén Assembler nyelvü/ programot irtam a tárgyalt témakörön belül, melyek a következő funkciók ellátására hivatottak:

1./ <u>SELMER</u> /<u>select</u>, <u>merge</u>/

Különböző JBACOUNT-file-ok összeválogatását, valamint JBACOUNT-file-bol naptári dátum alapján történő leválasztást végez e program. - a SESSION1 program részére, és a rendszerkiértékelő programok részére szükséges JBACOUNT-file összeállitása céljából/különböző időtartamu - havi, dekád, heti, stb. - kiértékelésekhez/.

2./ <u>RADIO</u>

Számitóközpontunk különböző intézmények részére is dolgozik - habár nem elsődlegesen. A külső felhasználók részére munkaelszámolást kell adnunk, ezért un. intézményazonositó kódot rögzitünk a JAC-kártya 3. paraméter-mezőjébe. Ez a program az intézményazonositó kód alapján leválassza a SESSION1-JBACOUNT file-ról a külső felhasználók tételeit,melyekből a kiértékelő programok elkészitik a Job-időelszámolást az egyes felhasználók részére.

3./ ACCOUNT3

Kiértékelő-táblázó program, amely a JAC-kártya 2. paraméter-mezőjébe megadott elszámolási kontószám első három számjegye alapján - ami nálunk a főbb feldolgozási témacsoportokat reprezentálja - feldolgozási témák szerint összesitett kimutatást készit. A feldolgozást 6o témacsoportra osztottuk/ilyenek pl.: adatbank update, kamatszámitás, Cobol programok forditása, stb./. E program által készitett kimutatás a futási időt, a központi egység időt, a nyomtatott sorok számát tartalmazza témánként és összesen, valamint a három összesitett értéknek témánkénti százalékos megoszlását.

IV. Továbbfejlesztett Job-Account kiértékelő rendszer gyakorlati felhasználása.

Intézményünknél a Job-időelszámolás, a gépkihasználás és a gépleterhelés mérése az ismertetett programok segitségével történik.

A kiértékelést egyrészt havi szinten, másrészt a tárgyhónapot kettéosztva éles és tesztidőszakra külön-külön is rendszeresen elvégezzük. Ezenkivül esetenként dekád, heti, napi, müszakonkénti, illetve ezektöl eltérő időtartamu kiértékelést szolgáltatunk. A naponkénti kiértékelés rendszeres bevezetésére a közeljövőben kerül sor.

Az éles futtatások időszakára /ami a hónap első 1o-15 munkanapját öleli fel/ vonatkozó Job-Account összesitő alapján elemzésre kerül a multiprogramozás teljesitett szintje és a számitógép intenziv kihasználása. Amennyiben

a mutatók a gép nem megfelelő leterhelésére utalnak, intézkedés történik a géptermi munka jobb megszervezése céljából.

A vezetőség a Job-Account összesitő és a számitógég extenziv kihasználását regisztráló Job-Account elszámolási időszakok c. kimutatásból áttekintést kap és ellenőrzést gyakorol az üzemeltetési tevékenységnek a Job-Account-ban tükröződő keresztmetszetéről. Megoldásra került a külső felhasználókra vonatkozó munkaelszámolás, valamint a saját feldolgozások megfelelő témánkénti Job-elszámolása.

A kialakitott, ill. továbbfejlesztett Job-Account kiértékelő rendszerben rejlő további lehetőségek kihasználása folyamatban van.

A SESSION1, SELMER, RADIO, ACCOUNT3 programokat adaptáció nélkül is tudnák alkalmazni a többi Siemens felhasználók - BS looo operációs rendszerben.

ALLAMI FEJLESZTESI BANK
[illegible]

DATUM: [illegible] IDO: [illegible]

1.sz. melléklet

```
**********************************************
*                                            *
*  J O B - A C C O U N T   Ö S S Z E S I T Ö *
*                                            *
**********************************************
```

([illegible] - [illegible])

[illegible] ALAPADATOK

CPU-TIME (KOZPONTI EGYSEG IDO): 0083 ORA 53 PERC 05 MP

PROGRAM-TIME (PROGRAMOK OSSZES FUTASI IDEJE): 0339 ORA 57 PERC 24 MP

ACCOUNT-PERIOD (ELSZAMOLASI, ILL. MEGFIGYELESI IDOSZAK): 0276 ORA 59 PERC 45 MP

MUTATOK

MULTIPROGRAMOZAS HATEKONYSAGI MUTATOJA = PROGRAM-TIME / ACCOUNT-PERIOD = 122,8 %

KOZPONTI EGYSEG KIHASZNALASA = CPU-TIME / ACCOUNT-PERIOD = 30,2 %

KOZPONTI EGYSEG PROGRAMFUTÁSOK ALATTI KIHASZNALASA = CPU-TIME / PROGRAM-TIME = 24,6 %

EGYEB INFORMÁCIOK

NYOMTATOTT SOROK SZAMA: 4.221.463

NYOMTATOTT LAPOK SZAMA: 105.321

ÁLLAMI FEJLESZTÉSI BANK

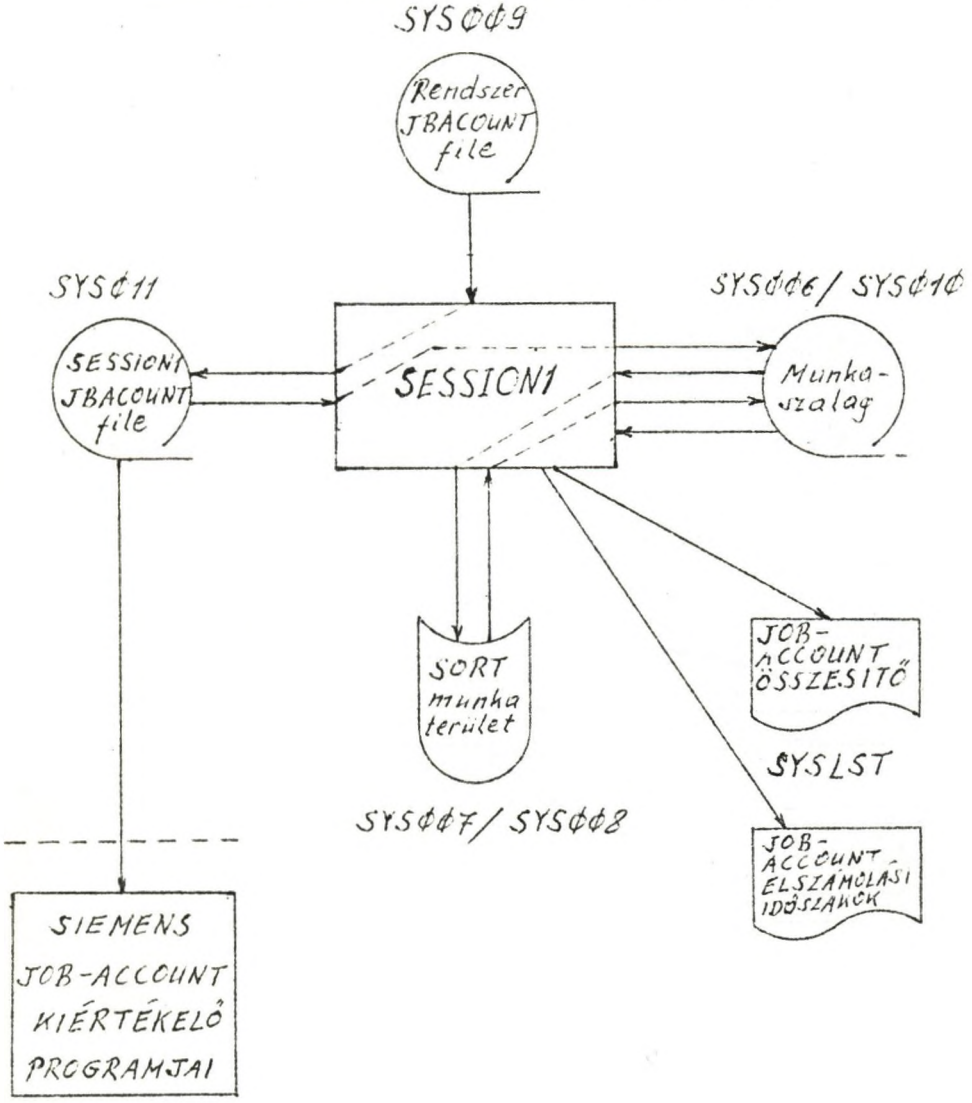
2. sz. melléklet

JOB-ACCOUNT ELSZÁMOLÁSI IDŐSZAKOK

K E Z D E S	B E F E J E Z É S	IDŐTARTAM
78/02/01 [illegible]	78/02/11 10:20:15	[illegible]
78/02/11 11:46:59	78/02/12 [illegible]	[illegible]
78/02/13 08:30:00	78/02/13 08:57:56	[illegible]
78/02/13 09:57:12	78/02/13 22:05:30	[illegible]
78/02/23 08:24:11	78/03/03 19:20:17	[illegible]
78/03/14 21:36:01	78/03/14 21:36:13	[illegible]
78/03/16 07:10:46	78/03/16 07:10:52	[illegible]
78/03/16 16:00:16	78/03/16 16:56:42	[illegible]
78/03/16 18:18:52	78/03/16 19:59:50	[illegible]
78/03/16 21:30:10	78/03/16 21:42:57	[illegible]
78/03/17 01:10:05	78/03/17 05:[illegible]:25	[illegible]
78/03/17 14:36:14	78/03/17 15:[illegible]:8	[illegible]
78/03/17 15:18:12	78/03/17 15:17:42	[illegible]
78/03/17 16:37:55	78/03/17 17:37:42	[illegible]
78/03/17 18:31:09	78/03/17 20:19:32	[illegible]
78/03/17 21:05:15	78/03/17 21:26:25	[illegible]
78/03/20 00:41:52	78/03/20 02:39:08	[illegible]
78/03/20 [illegible]	78/03/20 06:20:22	[illegible]
78/03/21 08:32:47	78/03/21 [illegible]:44:14	[illegible]
78/03/21 11:24:54	78/03/21 21:47:49	[illegible]
78/03/19 06:34:37	78/03/19 10:50:24	[illegible]
78/03/15 [illegible]:21:46	78/03/15 [illegible]:35:28	[illegible]
78/03/19 10:28:19	78/03/19 12:43:[illegible]	[illegible]
78/03/14 [illegible]:22:40	78/03/14 06:28:26	[illegible]
78/03/14 09:22:47	78/03/14 [illegible]:35:29	[illegible]
78/03/14 10:22:08	78/03/14 11:57:24	[illegible]
78/03/14 16:39:09	78/03/14 17:37:46	[illegible]
78/03/22 [illegible]:[illegible]:54	78/[illegible]/15 [illegible]:[illegible]:27	[illegible]

3. sz. melléklet

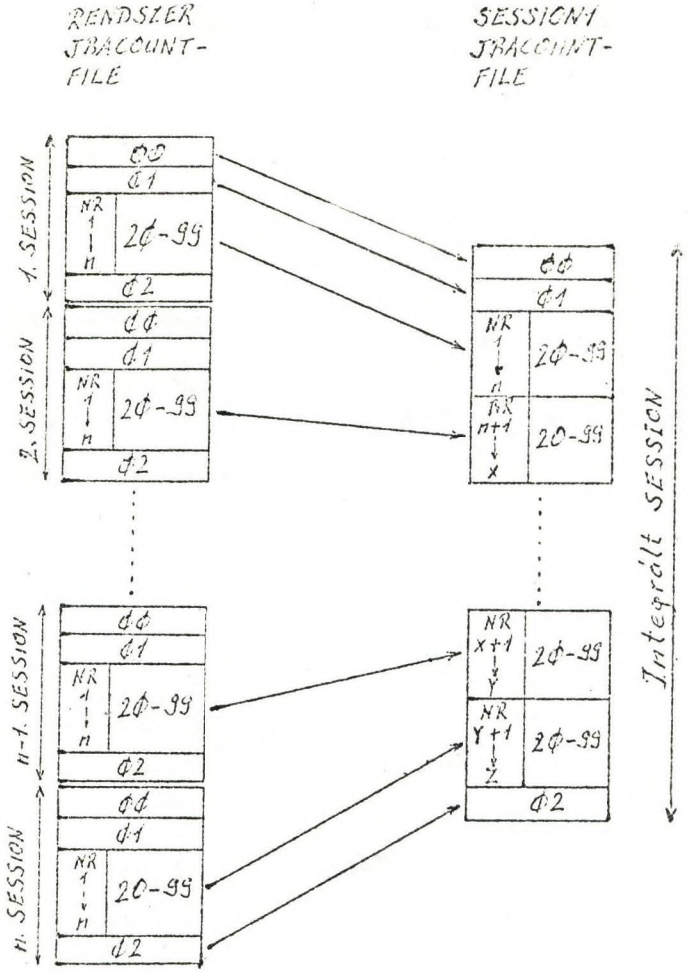
SESSION1 program gépi folyamatábrája



Megjegyzés: A SESSION1 program futtatásához 2 db mágnesszalag egység is elégséges mivel a munkaszalagra csak a rendszer Job-Account file beolvasása, ill. felszabadítása után van szükség.

JOB-ACCOUNT file átalakítása

4. sz. melléklet



Megjegyzések:

ΦΦ = JOB-ACCOUNT kezdő rekord
Φ1 = Konfigurációs rekord
2Φ-99 = JOB, ill. program rekordok
Φ2 = JOB-ACCOUNT befejező rekord
NR = Feladatsorszám (Auftrags-Nummer) JOB-okra, ill. programokra

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 Sokszorositó Üzem, Budapest
Felelős vezető: Büky Józsefné
SZÁMKI 780796