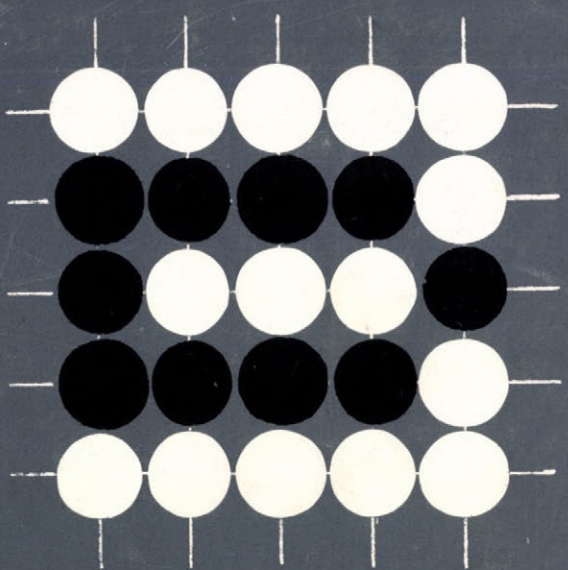
közlemények 15/1975

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest



MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZET

közlemények

ISBN 963 311 015 7

1975. DECEMBER

Felelős kiadó:

Dr. Vámos Tibor
igazgató

OPERÁCIÓS RENDSZEREK ELMÉLETE

Téli Iskola, Visegrád

MTA Számítástechnikai és Automatizálási Kutató Intézet
MTA Számítástudományi Bizottsága

Technikai szerkesztő:

Solt Jánosné
MTA Számítástechnikai és Automatizálási Kutató Intézete

757819 MTA KESZ Sokszorosító. F. v.: Szabó Gyula

TARTALOMJEGYZÉK

CONTENTS

СОДЕРЖАНИЕ

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

OPERÁCIÓS RENDSZEREK

Azoknak a software eszközöknek összességét, melyek a számitógépben lejátszodó folyamatokat vezérlik, operációs rendszernek nevezzük. Formáját tekintve az operációs rendszer egy igen bonyolult algoritmus. Az operációs rendszerek gyors fejlődése, egyre bonyolultabbá válása egy sereg megoldatlan elméleti problémát vetett fel. Ezekkel a kérdésekkel foglalkozik az operációs rendszerek elmélete. E terület sajátos vonása, hogy a gyakorlat nem várakozhat a felvetett kérdések megoldására, a rendszereknek "tüzön–vizen át" el kell készülniük. Ez természetesen nem jelenti azt, hogy az elméleti kérdéseket nem kell megoldani, és előbb–utóbb alkalmazni. Egy–egy ilyen "elméleti" kérdés megoldásának értéke álatában csak milliárdokban mérhető, és a gyakorlatban rögtön realizálható. Ez teszi érthetővé e kutatásokban világszerte bevetett hatalmas energiákat.

Peter Denning klasszikus felosztása alapján az operációs rendszerek funkciói a következőek:

1) Uj folyamatok létrehozása, elinditása illetve folyamatok megszüntetése. (*creating–removing*).
2) A folyamatok előrehaladási feltételeinek biztositása vagyis a végtelen várakozások logikai kizárása (*progress control*).
3) A különleges események feldolgozása (exceptional conditions, interrupts).
4) Erőforrások allokálása (software és hardware egyaránt).
5) Védelem és biztonság.
6) Folyamatok közötti kommunikáció és szinkronizáció.

Mindezek a funkciók az alábbi közös jellemzőkkel rendelkeznek:

a) Párhuzamos tevékenységek (*concurrency*).
b) Automatikus allokáció. Ezt az teszi szükségessé, hogy erőforrás igények jelentkezése véletlenszerű.
c) *Sharing*: erőforrások egyidejű használata több folyamat által. (Fizikai és logikai erőforrásokra egyaránt).
d) *Multiplexing*: Kvantumos időosztás.
e) Aszinkronitás: az események sorrendje nem jelezhető előre.

Azokat az elméleti próblémákat, melyek mindezekből adódnak, két csoportra szokás osztani:

I. Logikai próblémák. Ez azokat a feladatokat öleli fel, melyek megoldása nélkül az operációs rendszer egyáltalán nem képes müködni. Ezen próblémák az egyre újabb lehetőségek és egyre automatikusabb megoldások biztositása miatt tovább sokasodnak. Maga a "logikai" elnevezés nem túl szerencsés. Valójában *különleges tulajdonságú algoritmusok kereséséről*, és *tulajdonságainak bizonyitásáról* van szó. Ez még akkor

is igaz, ha sok publikációban az alkalmazott formalizmus mögött ezt néha nehéz észrevenni.

II. **Optimalizálási problémák.** Ez alatt a méretezés, prioritások, stratégiák megoldását értjük. A rendszer lényegéből eredően ezek a próblémák *valószinüségszámitási* jellegüek. Tekintettel arra, hogy az operációs rendszerekben lezajló folyamatok általában sokkal bonyolultabbak azoknál, mint amelyre kész valószinüségi modelljeink vannak, általában csak részpróblémákra tudunk analitikus megoldást adni. Ezek azonban mégis nagyon fontosak, és még akkor is jól alkalmazhatók, ha sok ideális és megközelitő feltételezésen alapúlnak.

Befejezésül szeretnék utalni arra, hogy az operációs rendszerek elmélete, kb. tiz évvel ezelőtt vált a számitástudomány önálló területévé. Megszületése szorosan kapcsolódik az u.n. harmadik generációs számitógépek elterjedéséhez, azoknak fizikai jellemzőin alapúl. Várható, hogy a technológiai oldal fejlődése további, új, minőségileg más próblémák felmerüléséhez fog vezetni. E tendencia mennyiségi oldala a méretek, az automatizáltság, a bonyolúltság növekedése. Ma már ide sorolhatjuk az operációs rendszerek egy speciális területét az *adatkezelést*, melynek ma még nincsen önálló elmélete, sajátos próblémái és módszerei azonban előbb vagy utóbb lehetővé teszik azok absztrakt megfogalmazását.

Knuth Előd
MTA Számitástechnikai és Automatizálási Kutató Intézete

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

TÖMEGKISZOLGÁLÁSELMÉLETI MODELLEK SZÁMOLÓGÉPRENDSZEREK MATEMATIKAI LEIRÁSÁBAN

Tomkó József

MTA Számitástechnikai és Automatizálási Kutató Intézete

Számológéprendszerek matematikai leirásában igen jelentős szerepük van a tömegkiszolgáláselméleti modelleknek. Ha meggondoljuk a számológépi programok sokrétüségét, változatosságát, mely a lefoglalandó külső, belső memóriaterületek nagyságában, a lefoglalási időtartamoknak a program futása alatti változásában, a végrehajtandó müveletek (aritmetikai és perifériális) egymásutáni változékonyságában mutatkozik meg, akkor szinte magától értetődő, hogy a számológépek matematikai modell-alkotásakor a tömegkiszolgálás különböző, s igen sok esetben ujszerü szkémái kerülnek előtérbe. Valójában az a helyzet, hogy a számológép egységei (központi feldolgozó egység, memória területek, átviteli berendezések/csak korlátos számban állnak rendelkezésre, miközben maguk a programok, s ezek közül az aktiváktól, a különböző egységekre érkező követelmények időről időre felhalmozódhatnak és sajátos sorbanállási-várakozási helyzetek alakulnak ki. A jelen tanulmányban áttekintünk néhány problémát melyek számológépek operációs rendszereinek szervezésekor merülhetnek fel. Ezek közül csak egynek a nyomtatás hatékonyabb szervezésének kérdéseit tárgyaljuk részletesebben. A többi vázolandó problémáról már jelentek meg avagy a közeljövőben fognak megjelenni magyar nyelvü tanulmányok.

Számológépi programok egy általános matematikai modellje. Nagyvonalakban egy számológépet kiszolgáló rendszerként vehetünk figyelembe, amelynek az érkezési igényfolyamatát a futtatásra leadott (avagy a távállomásokon keresztül küldött) számológépi programok áramlata alkotja. A feladat valójában a számológéprendszerekben a kiszolgálásnak olyanformán való megszervezése, hogy egyrészt a felhasználók különböző óhajai is többé-kevésbé teljesülhessenek, másrészt, hogy a számológép egységei a lehető legnagyobb mértékben legyenek kihasználva. Nyilvánvaló, hogy a feladat felvetést jelentősen konkretizálni kell ahhoz, hogy valamilyen megoldásról beszélhessünk. Meg kell tudnunk, például, adni az igényfolyamatot alkotó számológépi programok (a továbbiakban "job"-okként is fogunk rájuk hivatkozni) matematikai modelljét. Amint látni fogjuk ez a matematikai modell változhat az egyes konkrét céloktól, törekvésektől függően. Az alábbiakban kitérünk egy olyan matematikai modellre, amely elég általános leirását adja a számológépi programoknak.

Számológépi programok rendszerint a számológép több egységét foglalják le. Egy programnak szüksége van külső, belső memóriaterületre, a központi feldolgozó egységre, csatornákra, nyomtatóra, stb.. Az egyes egységek lefoglalási ideje általában nem folytonos, hanem szakaszos. Másrészt vannak egységek, amelyeknek csak bizonyos hányadát igénylik a jobok hol rövidebb hol hosszabb időtartamokra. Mindezekre való tekintettel kézenfekvő egy számológépi programot a számológép egységeinek a program által való igénybevételét leiró függvényekből álló

(1) $\underline{p}(t) = \{p_1(t), p_2(t), \ldots, p_N(t)\}$

vektorral jelemezni. N a számológép egységeinek a számát jelöli, $p_i(t)$ $(i = 1,2, \ldots, N)$ pedig megadja, hogy az adott program, saját idejében t idővel előrehaldva, a számológép i-sorszámu egységének milyen hányadára tart igényt. Általában $0 \leqslant p_i(t) \leqslant 1$ $(i = 1, \ldots, N)$. Több egységre, igy pl. a központi feldolgozó egységre, a csatornákra, a nyomtatóra, a komponens függvény csak 0, 1 értékü lehet.

Rendszerint a $\underline{p}(t)$ vektor még a program készitője számára is homályban van s csak nagy általánosságokban tud róla felvilágositást nyujtani. Általában nem egyedi programok leirására van szükség, hanem program sokaságokéra, ezért a $\underline{p}(t)$ vektor komponenseit véletlen függvényeknek, sztochasztikus folyamatoknak kell tekintenünk. Általában ezek a folyamatok összefüggőek, ami a matematikai modellt jelentősen elbonyolitja.

Számológépi programok ezen általános matematikai modellje Knuth Elődtől származik, aki [2]-ben a szokásos leirástól eltérően "Simula" nyelven értelmezi fogalmait. A modell ilyen leirása különösen előnyös, ha az analitikus tárgyalásmód bonyolultsága, szinte járhatatlansága következtében a szimulációs eljárásokhoz vagyunk kénytelenek folyamodni.

Rámutatunk most a sztochasztikus folyamatok néhány osztályára, amelyek szóba jöhetnek az (1) vektor komponens függvényeinek megadásakor. Emlitettük, hogy egy job külső ill. belső memóriaigénye változhat (esetleg a gazdaságosabb helykihasználás érdekében törekedni kell arra, hogy változzon) az időben. Forditsuk figyelmünket csak a belső memória igényre. A belső memória-terület egysége a CDC 3300-as gép esetén a "CØRE". Az igényelt CØRE szám korlátozott, tegyük fel, hogy a felső korlátja L, alsó korlátja legyen l. Egy job belső memória igényének változását irja le a $P_{BM}(t)$ függvény, mely a belső memóriákhoz, mint a számológép egy egységéhez rendelt komponens függvény L-szerese. $P_{BM}(t)$ a l és L közé eső egész számokkal mint állapottérrel rendelkező sztochasztikus folyamat. Kézenfekvő, hogy megadásakor a folytonos idejü Markov folyamatokra gondoljunk, figyelembe véve ezáltal az állapotváltozásokban bizonyos jellegü függőséget is. Valamivel általánosabb matematikai modellhez jutunk, ha $P_{BM}(t)$-t fél Markov folyamatnak tekintjük, mely analitikus szempontból bonyolultabbá teheti a problémát, de szimulációs szempontból nem idéz elő áthidalhatatlanoknak tünő nehézségeket.

Röviden vázoljuk most, hogyan irható le a központi feldolgozó egységhez tartozó komponens függvény. Legyen ezen függvény $p_1(t)$ amely mint már emlitettük csak két értéket vehet fel, 1-et ill. 0-t attól függően, hogy igényli-e a program a központi feldolgozó egységet vagy sem. Más szóval $p_1(t)$ két állapotu sztochasztikus folyamat, melynek megadása tőrténhet két valószinüségi változókból álló sosozattal $(X_i, i \leqslant 1)$, $(Y_i, i \geqslant 1)$ melyek közül az elsőnek az elemei azok az időtartamok, amelyek alatt $p_1(t)$ értéke 1, a második sorozat elemei pedig azok az időtartamok, amelyek alatt $p_1(t) = 0$. Szokás az (X_i, Y_i) valószinüségi változók párjára a folyamat i-edik ciklusaként hivatkozni. A $p_1(t)$ folyamat további jellemzése ciklusainak részletes leirásával, függőségi viszonyuk, eloszlásuk megadásával történhet. A következő pontban

az analitikus kezelhetőség érdekében a ciklusokat függetleneknek és azonos eloszlásuaknak tekintjük. Több matematikai modell jöhet szóba $p_1(t)$ ciklusainak leirásakor. Megemlitjük például, hogy [1]-ben $\{X_i, Y_i;\ (i \geqslant 1)\}$ exponenciális eloszlásu valószinüségi változók. Paramétereik $(\lambda_i, \mu_i)\ (i \geqslant 1)$ szintén valószinüségi változók, melyek függőségi viszonyát elsőrendü autóregressziós séma irja le. Azzal kapcsolatban, hogy melyik matematikai modell tükrözi legjobban a valódi helyzetet még igen keveset tudunk mondani. A kérdés elég bonyolultnak látszik, megválaszolásához a számológéphez leadott programok mélyreható statisztikai elemzése, statisztikai adatok gyüjtése szükséges. Meg kell emliteni, hogy a statisztikai adatok gyüjtése nem egyszerü dolog. Az operációs rendszerekbe eljárások beépitését igényli s emiatt jelentős időveszteséget eredményezhet.

A központi feldolgozó egység kihasználtsága. A számológéprendszerek gazdaságosságának egyik mérőszáma a központi feldolgozó egység (*K F E*) kihasználtsága. Korábban a számológépek *KFE*-ének kihasználtságát elsősorban a futtatandó programok jellege határozta meg. A kihasználtság növelése valójában csak az adatátviteli csatornák adatátviteli sebességének fokozásán keresztül valósult meg. A korszerü számológépek *KFE*-ének és adatátviteli csatornáinak müködése egymástól nagymértékben függetlenitve van, ezért amig egy program részére adatátvitel történik azalatt egy másik program számolási utasitásainak végrehajtását végezheti a *KFE*. Másszóval létrejött a multiprogramozásnak egy változata, melynek jó megszervezésével kihasználtsága jelentősen növelhető.

Multiprogramozásu számológépek *KFE*-e kihasználtságának tanulmányozásában jelentős szerepet kapnak a tömegkiszolgáláselméleti modellek. Az alábbiakban ennek vázlatos illusztrálására térünk ki. A probléma részletesebb tárgyalása képezi a szerző [5] dolgozatának témáját.

Mindenekelőtt ismertetjük a *KFE* kihasználtságának fogalmát. Legyen $\{\xi_t, t \geqslant 0\}$ a *KFE* foglaltságát leiró szotchasztikus folyamat, azaz legyen 1 vagy 0 aszerint, hogy a *KFE* t pillanatban foglalt vagy szabad. A *KFE* [0,T] időszakaszra vonatkozó összfoglaltsági ideje $\int_0^T \xi_t dt$. Igen általános feltevések mellett

$$\frac{1}{T} \int_0^T \xi_t dt$$

sztochasztikusan konstanshoz tart midőn $T \to \infty$. Ugyancsak elég általános feltevések mellett ez a konstans megegyezik a

$$\lim_{T \to \infty} \frac{1}{T} M \int_0^T \xi_t dt = c$$

közönséges határértékkel, melyet a központi feldolgozó egység kihasználtságának nevezünk. Feltevéseink mellett ξ_t regenerációs tipusu folyamat lesz, független ciklusok (foglaltsági és szabad preiodusok) irják le. Ezért a *KFE* kihasználtsága

$$c = \frac{\text{átlagos foglaltsági periodushossz}}{\text{átlagos ciklusidő}} .$$

Figyelmünket csak a *KFE* kihasználtságára koncentráljuk, ezért feltesszük, hogy a multiprogramozásban résztvevő programok számára a számológép más egységei mindig a kívánt időben és mennyiségben rendelkezésre állanak. Igy többek között feltételezzük, hogy a számológép legalább annyi adatátviteli csatornával rendelkezik, ahány program vesz részt a multiprogramozásban. Ezen feltevések miatt nincs szükség a programok előző pontbeli általános statisztikai leirására, elegendő csupán a *KFE* igénybevételét leiró $p_1(t)$ komponens függvény megadása. Analitikus módszerekkel aránylag könnyen meghatározható a *KFE* kihasználtsága, ha feltételezzük, hogy a programok teljesen exponenciális szerkezetüek. Ezalatt azt értjük, hogy a $p_1(t)$ folyamat ciklusai függetlenek, továbbá, hogy mind a számolási időtartamok, X_i-k ($i \geqslant 1$) mind a perifériális berendezésekkel való kapcsolattartás müveleti idejei (input-output, röviden I/0 idők), Y_i-k ($i \geqslant 1$) exponenciális eloszlásuak, X_i-k közös α paraméterrel, Y_i-k közös λ paraméterrel.

Tegyük fel ezekután, hogy n program vesz részt multiprogramozásban, melyek teljes exponenciális szerkezetüek. Legyen α_i, λ_i az i-edik job számolási ill. I/0 idejének paramétere. Minthogy csatornára a joboknak nem kell várakozniuk, ezért, kiszolgálónak tekintvén a *KFE*-et, egy véges forrásu egykiszolgálós rendszerrel találjuk magunkat szemben, amely készülékének foglaltsági valószinüsége adja meg a *KFE* kihasználtságát. Igen egyszerü az utóbbi valószinüséget kiszámitani, ha a programok u.n. homogének, mind α_i-k mind λ_i-k azonosak. Nem homogén jobok-ra a *KFE* kihasználtságának meghatározásakor

$$\sum_{i=1}^{n} \binom{n}{i} i!$$

számu lineáris egyenletrendszert kell megoldani. Kihasználva az egyenletrendszer bizonyos specialitását a CDC 3300-as gépén $n = 5$-ig adott α_i, λ_i paraméterekre kiszámitható a *KFE* kihasználtsága. A megoldandó egyenletrendszert [5] alapján, amelyben az $n = 3$-ra van megadva, az olvasó is könnyedén felirhatja. A megoldás számológépi programját és más, a számológéprendszerek hatékonyságának vizsgálatakor felmerülő számitások programját a szerző a közeljövőben közölni fogja.

Nem szoltunk fentebb a kiszolgálási sorrendről. Homogén jobok esetén, hacsak a *KFE* kihasználtságára vagyunk tekintettel, a *KFE* lefoglalásának sorrendje közömbös. Nem homogén jobokra az emlitett eredmények arra az esetre vonatkoznak, amikor a jobok I/0 idejük befejeződésének sorrendjében foglalják le a *KFE*-et.

Éppen a *KFE* kihasználtságának fokozása érdekében a programokat prioritási osztályokba sorolják, s igyekeznek a multiprogramozást úgy megvalósitani, hogy mindegyik prioritási osztályból legyen egy job aktiv állapotban a gépben. Nem célunk a prioritás bevezetésével előálló helyzet akár rövid ismertetése sem. Az olvasó [5]-ben megtalálhatja a részletes tárgyalást.

Végül még megemlitjük, hogy a korszerü számológépek u.n. időosztásos szervezésben müködnek, mely azt jelneti, hogy egy program csak előre meghatározott q kvantum időre kapja meg a *KFE*-et. Ha ez idő alatt nem fejeződik be a soronlévő számolási idő, akkor a futása megszakad, a *KFE* az adott kiszolgálási elv szerint következő program kiszolgálására tér át,

a félbeszakitott program pedig megint az adott kiszolgálási elvnek megfelelően sorbaáll. E modell részletesebb leirása tanulmányozása alkotja majd a [5] dolgozat második részét. Időosztásos rendszerek *KFE*-ének kihasználtságával foglalkoznak a [3],[4] angol nyelvü dolgozatok.

Egy nyomtatási próbléma. A korszerü számológépeknél a program során keletkező kimenő információ általában nem közvetlenül a keletkezés után nyomtatódik ki. Rendszerint a futóprogramhoz az operativ memóriában egy jól meghatározott terület "buffer" tartozik, melyen a kimenő információ ideiglenesen elhelyezést nyer. Ha a buffer megtelt, akkor a program I/0 fázisba megy át mely tart mindaddig amig csak a buffer tartalma át nem helyeződik egy perifériális berendezésre, rendszerint mágneslemezre, "disk"-re. A program futása ilyen megszakitásokkal megy végbe. A keletkező információ a programozó által kért terjedelmü disk területen gyülemlik fel, melyről a program futásának befejeződése után a nyomtató berendezés kinyomtatja. A mi problémánk azzal az esettel kapcsolatos, amikor a program oly sok kimenő információt állit elő, hogy annak ideiglenes tárolása a disken fizikailag lehetetlen. Ilyen esetekben szokás azt csinálni, hogy a programot futattják mindaddig amig csak a rendelkezésre álló disk terület be nem telt kimenő információval. Amikor már nincs a disken szabad terület, ahova a kimenő információ tárolható volna, akkor a program futása felfüggesztődik, a nyomtató kinyomtatja a keletkezett kimenő információt s miután a szóbanforgó disk terület (az u.n. outfile) felszabadult a program futása újból megindulhat. Igen nagy terjedelmü kimenő információt előállitó program futása többször is felfüggesztődik. Ezáltal a program ideje (az operativ memóriában való tartozkodási ideje) elhuzódik. Másrészt, gyakran az ilyen programok a teljes operativ memóriát elfoglalják, s miután igy multiprogramozásra nincs lehetőség a központi feldolgozó egység kihasználtsága is jelentősen csökken.

Azért, hogy nagy terjedelmü kimenő információt előállitó programok többszöri hüzamosabb időre történő felfüggesztését elkerüljük az eredmények nyomtatását másképpen kellene megszervezni. Ennek egyik módja lehet a következő nyomtatás szervezés.

Mindenekelőtt állapodjunk meg néhány elnevezésben, jelölésben. A program futása során egy buffernyi kimenő információ felgyülemlésének ideje általában valószinüségi változó, melyet ξ-vel (alkalomadtán indexelve) fogunk jelölni. A kimenő információ ideiglenes tárolására szolgáló disk terület egységének azt a területet választjuk, amelyben egy buffernyi információ tárolható. Tegyük fel, hogy a kimenő információ ideiglenes tárolására a disken M terjedelmü terület áll a rendelkezésre. Általában az egységnyi (buffernyi) információ kinyomtatási ideje is valószinüségi valtozó, melyet η-val (ha kell indexelni is fogjuk) jelölünk. Legyen most N egész szám M-nek osztója. Tegyük fel, pl., hogy $M = rN$. A nyomtatás uj szervezése ezekután abból állna, hogy mihelyt a kijelölt M területen N egységnyi kimenő információ felgyülemlett , ennek a nyomtatása kezdődjön meg, de a program fusson tovább, s az esetleg keletkező kimenő információ tárolható a disk terület következő N terjedelmü szeletein. Előfordulhat, hogy a kijelölt disk terület összes r szelete betelik kimenő információval. Ezek egyikének a felszabaditása már folyamatban van, igy a program futásának felfüggesztése legfeljebb egy szeletnyi információ nyomtatási idejéig tart. Igaz, hogy most a program futásának felfüggesztései

gyakrabban fordulhatnak elő. Feladatunk éppen az, hogy összehasonlitsuk a két nyomtatás szervezés effektivitását, melyet a programok összfelfüggesztési idejének várható értékével jellemzünk. Nyilván azt a nyomtatás szervezést tekintjük effektivebbnek, melyre ez a várható érték kisebb.

Mielött a tárgylás részleteibe bocsátkoznánk meg kell emliteni, hogy a nyomtatás fenti szervezése Tőke Páltól ered, s szorosan kapcsolatos a disk tároló terület (scratch pool) dinamikus használatával (v.ö. [7]).

Egy program összfelfüggesztési idejének várható értéke függ a program futása során keletkező kimenő információ mennyiségétől, továbbá egy szeletnyi (N egységnyi) információ felgyülemlésének és kinyomtatásának idejétől. Nyilvánvaló, hogy az utóbbi két időtartam

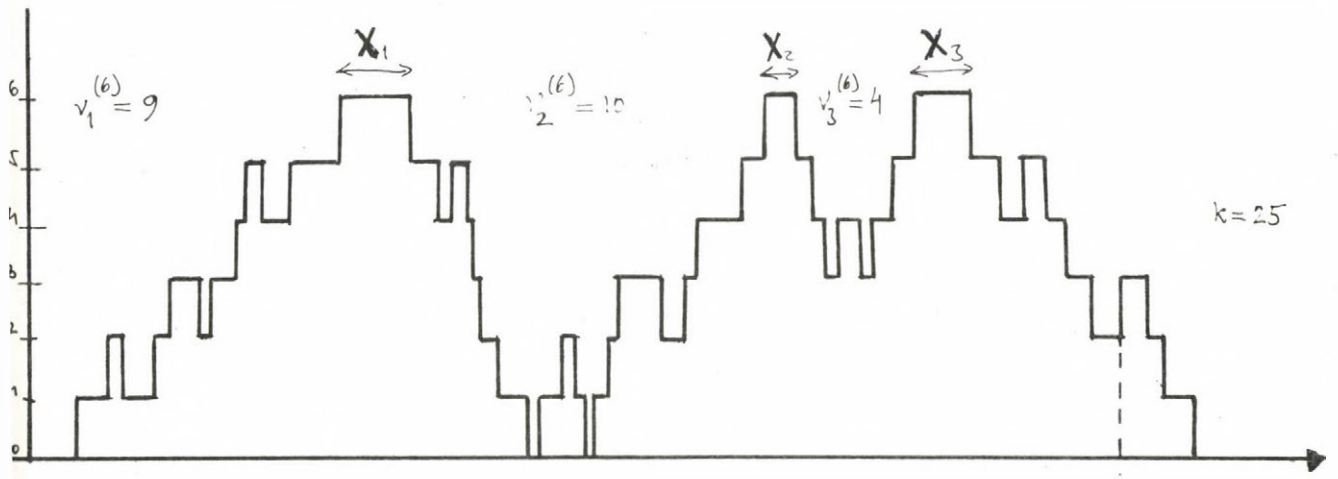
$$\xi(N) = \sum_{i}^{N} \xi_i \quad \text{illetve} \quad \eta(N) = \sum_{i}^{N} \eta_i$$

ahol ξ_i és η_i $(i \geqslant 1)$ a korábban már bevezetett valószinüségi változók. Itt is program sokaságra gondolunk, ezért a kimenő információ mennyiségét egész értékü valószinüségi változónak tekintjük, melynek eloszlását $(r_k\,,\ k \geqslant 1)$ ismertnek tételezzük fel. r_k annak a valószinüsége, hogy egy program futása alatt k egységnyi (buffernyi) információ keletkezik.

Egy program összfelfüggesztési idejének tanulmányozása egy véges sorkapacitásu tömegkiszolgálási modell segitségével történhet. Tekintsük a nyomtatót kiszolgálónak, melyhez az érkező igényeket a disken megjelenő (N egységnyi) információ szeletek képezik. Minthogy maximálisan csak r szelet információ tárolható egyidejüleg a disken, ezért a sorhosszoság legfeljebb $r - 1$ lehet. Az igények érkezése közti időtartamok a disk szeleteinek információval való feltöltődési idejei $\xi_1(N)$, $\xi_2(N)$, . . . , . A kiszolgálási idők az információ szeletek nyomtatási idejei $\eta_1(N)$, $\eta_2(N)$, ... A program futásának ideiglenes felfüggesztése azt eredményezi, hogy ha a sorhosszúság $r - 1$, akkor újabb igény nem jelenik meg az érkezési folyamatban, azt is fogjuk mondani, hogy az érkezési folyamat felfüggesztődik.

Elég áttekinthető, számitástechnikailag aránylag könnyen kezelhető eljárás adható meg az érkezési folyamat (program) összfelfüggesztési ideje várható értékének meghatározására, ha mind $\xi_i(N)$-k mind $\eta_i(N)$-k $(i \geqslant 1)$ független exponenciális eloszlásu valószinüségi változók. Ez a feltevés minden bizonnyal durva közelitést eredményez, nélküle viszont a probléma analitikusan nem tárgyalható. Az alábbi levezetendő eljárás alapján átfogó képet kaphatunk a nyomtatás szervezés effektivitását illetően, másrészt támaszpontra teszünk szert a szimulációs eljáráshoz, annak ellenőrzéséhez.

Rögzitsük most egy program futása során keletkező információ mennyiséget, feltételezvén, hogy ez kN és $(k + 1)N$ között van. $k = 25$-re és $r = 6$-ra az alábbi ábra illusztrálja a sorhosszuság egy lehetséges realizációját. Egyuttal leolvasható az érkezési folyamat felfüggesztésének összideje is.



A realizáció szerint a program 3-szor függesztődik fel, s a felfüggesztődés összideje $X_1 + X_2 + X_3$, ahol X_i-k függetlenek s $\eta_i(N)$-kel azonos exponenciális eloszlásuak. A tekintetbe vett programunkra a felfüggesztődés összideje a

$$(2) \qquad \sum_{i=1}^{L_k} X_i$$

véletlen összeg, melyben szereplő L_k értelmezése a következőképpen adható meg. Jelentse $\tau^{(r)}$ a program első felfüggesztődéséig betelt disk szeletek számát (az érkezési folyamat első felfüggesztődése előtt beérkezett igények számát): $\sigma_i^{(r)}$ $(i \geqslant 1)$ az érkezési folyamat (a program) i-edik és $i+1$-edik felfüggesztődése között beérkezett igények (betelt disk szeletek) számát. Legyen $S_0 = 0, S_1 = \tau^{(r)}$ és $S_n = S_{n-1} + \sigma_n^{(r)}$ $(n \geqslant 2)$. Ekkor kN és $(k+1)N$ közé eső kimenő információval rendelkező program felfüggesztődéseinek száma

$$L_k = \max\{ l \geqslant 1 : l_e \leqslant k \}$$

Minthogy L_K és X_i-k függetlenek, ezért az (2) alatti véletlen összeg várható értéke

$$MX_i ML_k .$$

Ezért a program felfüggesztődési összideje átlagértékének kiszámitásához a

$$H_k = ML_k \qquad (k \geqslant 1)$$

felújitási függvényt kell meghatározni. Ismeretes, hogy a $(H_k, k \geqslant 1)$ sorozat eleget tesz

$$H_k = \sum_{l=1}^{k-r} H_{k-l}\, p_l + \sum_{l=1}^{k} p_k^*$$

rekurziv (felújitási) egyenletnek, ahol

$$p_l^* = P\{\tau^{(r)} = l\}, \qquad p_l = P\{\sigma_1^{(r)} = l\}.$$

Egyszerü számitásokkal megmutatható, hogy a

$$H(z) = \sum_{k=r}^{\infty} H_k z_k, \quad f^{(r)}(z) = \sum_{r}^{\infty} p_l^* z^l, \quad g^{(r)}(z) = \sum_{1}^{\infty} p_l z^l$$

generátorfüggvények között a

$$H(z) = \frac{f^{(r)}(z)}{(1-z)\,(1-g^{(r)}(z))} \tag{3}$$

kapcsolat áll fenn.

Mielőtt a $H_k (k \geqslant 1)$ értékeknek a (3) formula alapján történő meghatározását ismertetnénk, vissza kell térnünk $\xi(N)$ és $\eta(N)$ eloszlásához. Feltevésünk szerint ezek a változók exponenciális eloszlásuak. Paraméterük nyilvánvalóan függenek a disk szeletek terjedelmétől N-től. Természetes feltételezni, hogy e függőségi viszony forditottan lineáris, azaz, hogy $\xi(N)$ eloszlása λ/N, $\eta(N)$ eloszlása μ/N paraméterü exponenciális eloszlás, valamilyen λ és μ konstansokkal. Később látni fogjuk, hogy H_k csak r-en keresztül függ N-től, ezért $\xi(N)$ és $\eta(N)$ paramétereit egyszerüen λ-val illetve μ-vel jelöljük.

Rátérünk most az $f^{(r)}(z)$ és $g^{(r)}(z)$ generátor-függvények meghatározására. Legyen

$$p = \frac{\mu}{\mu + \lambda}, \qquad q = 1 - p,$$

s jelöljön κ egész értékü (nem negativ) valószinüségi változót, melynek eloszlása p-paraméterü geometriai eloszlás, azaz

$$P\{\kappa = i\} = p^i q \qquad (i \geqslant 0).$$

Aránylag egyszerü meggondolásokkal észrevehető, hogy

$$\tau^{(r)} \doteq \tau^{(r-1)} + \sigma^{(r)},$$
$$\sigma^{(r)} \doteq 1 + \sum_{i=1}^{\kappa} \sigma_i^{(r-1)},$$

ahol a $\doteq$ jel azt fejezi ki, hogy a két oldalon álló mennyiségek eloszlásai azonosak. Nyilvánvaló, hogy $\tau^{(1)} = \sigma^{(1)} = 1$ azaz $f^{(1)}(z) = g^{(1)}(z) = z$. A fenti rekurziv képletek alapján pedig

$$g^{(r)}(z) = \frac{zq}{1 - pg^{(r-1)}(z)} \qquad (r > 1),$$

$$f^{(r)}(z) = f^{(r-1)}(z)g^{(r)}(z).$$

Ezekután könnyü megmutatni, hogy mind $f^{(r)}(z)$ mind $g^{(r)}(z)$ racionális törtfüggvények, melyek nevezőinek gyökei mind különbözőek és 1-nél nagyobbak. Legyen ugyanis $D^{(1)}(z) = 1$, $D^{(2)}(z) = 1 - pz$ és

$$D^{(k)}(z) = D^{(k-1)}(z) - pqzD^{(k-2)}(z), \qquad (k > 2).$$

Teljes indukció segitségével könnyen verifikálható, hogy

(i) $D^{(r)}(1) = q^{r-1}$,

(ii) $f^{(r)} = z^r \frac{q^{r-1}}{D^{(r)}(z)}$, $g^{(r)}(z) = \frac{zqD^{(r-1)}(z)}{D^{(r)}(z)}$

(iv) $D^{(2r)}(z)$ és $D^{(2r+1)}(z)$ fokszáma r, s legmagasabb hatványuk (z^r) együtthatója $(-1)^r p^r q^{r-1}$ illetve $(-1)^r p^r q^{r-1}(r + q)$.

E tények felhasználásával most már megmutatható, hogy $D^{(k)}(z)$ gyökei 1-nél nagyobbak és mind különbözőek, továbbá, hogy ha

$D^{(2r)}(z)$ gyökei $z_1, z_2, \ldots, z_r$

$D^{(2r+1)}(z)$ gyökei $w_1, w_2, \ldots, w_r$, akkor

$$w_1^{(r)} < z_1^{(r)} < w_1^{(r-1)} < z_1^{(r-1)} < w_2^{(r)} < z_2^{(r)} < w_2^{(r-1)} < z_2^{(r-1)}$$

$$< \ldots < w_{r-1}^{(r)} < z_{r-1}^{(r)} < w_{r-1}^{(r-1)} < z_{r-1}^{(r-1)} < w_r^{(r)} < z_r^{(r)} .$$

A $H^{(r)}(z)$ generátor-függvénye ezekután ezekután kapjuk a

(4) $$H^{(r)}(z) = \frac{z^r q^{r-1}}{(1 - z)(D^{(r)}(z) - zqD^{(r-1)}(z))}$$

formulát, melynek nevezőjében $D^{(r)}(z) - zqD^{(r-1)}$ l-ed foku, ha $r = 2l$, l + 1-ed foku, ha $r = 2l + 1$. A rekurziv összefüggések felhasználásával megint könnyen megmutatható, hogy (4) jobboldala nevezőjének $z = 1$ kétszeres gyöke, s a többi gyökök 1-nél nagyobbak és mind különbözőek. Ezen megállapitásokra való tekintettel, többszörösen felhasználva a rekurziv összefüggéseket, aránylag egyszerü számológépi programmal meghatározható tetszőleges k-ra a H_k érték. Amint már láttuk ez elegendő a program felfüggesztődései összidejének meghatározásához, minthogy $MX_i = N/\mu$.

Adott $\{r_k, k \geq 1\}$ eloszlással jellemzett programsokasságra a felfüggesztődések összidejének várható értéke

$$\frac{N}{\mu} \sum_{k=1} H_k \rho_k ,$$

ahol

$$\rho_k = \sum_{k \cdot N+1}^{(k+1)N} r_l \qquad (k \geq 0).$$

Irodalom

[1] Arató M.–Knuth E.–Tőke P.: On stochastic control of a multiprogrammed computer based on a probabilistic model. Preprint of the Stochastic Symposium, IFAC, Budapest, (1974)

[2] Knuth E.: Multiprogramozású számitógép rendszerek. Mérés és Automatika, 5, 208–211, (1975).

[3] Narayan Bhat U.–Nance R.: Busy Period of a Time–Sharing System Modeled as a Semi–Markov Process. J. ACM. vol. 18, N° 2, 221–238, (1971).

[4] Purdom P, . . . : Analysis of Two Time–Sharing Qeuueing Models. J. ACM, vol 19, N° 1, 70–91, (1972).

[5] Tomkó J.: Számológépek központi feldolgozó egységének kihasználtságáról. MTA Alkalmazott Matematikai Lapok, N° 1. (1975).

[6] Tomkó J.: Studying Output Organization with a single finite Queue. Proceedings of Computer Science Conference, Székesfehérvár, 121–125, (1973).

[7] Tőke P.: Dinamical Use of Scratch Pool. Proceeding of Computer Science Conference, Székesfehérvár, 117–121, (1973).

Summary

Some queueing models in the mathematical description of computer systems

J. Tomkó

The papare deals with some problems of the mathematical description of computer systems. The problems are modelled as particular queueing systems. A new way of output organization is discussed in details.

Р е з ю м е

Модели массового обслуживания в математическом описании вычислительных систем

Й. Томко

В работе рассматриваются некоторые проблемы математического описания вычислительных систем. Задачи формализируются как схемы массового обслуживания. Подробно изучается организация печати выходной информации.

MTA Számítástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

OPERÁCIÓS RENDSZEREK MÜKÖDÉSÉNEK DIFFUZIÓS KÖZELITÉSE I.

(Általános megjegyzések)

Arató Mátyás

MTA Számítástechnikai és Automatizálási Kutató Intézete

1. Tapasztalatainkról

Intézetünk Valószinüségszámitási és Matematikai Statisztikai Osztálya, valamint a Rendszertechnikai és Programozáselméleti (Software) Osztálya munkatársainak egy kisebb csoportja 1972-ben kezdett el foglalkozni operációs rendszerek kérdéseivel, valamint a megfelelő matematikai modellek tanulmányozásával és különböző tipusu feledatok megoldásával. Erre a következő okok miatt került sor: elsősorban a CDC – 3300-as gép operációs rendszerének ismeretében és a müködési tapasztalatok alapján olyan gyakorlati kérdések merültek fel (a scratch pool (SCR) dinamikus kezelése, a valódi multiprogramozás biztositása, a kvantum idő beállitása stb.), melyek nemcsak az operációs rendszer ismeretét, hanem mély elméleti meggondolások is igényelték. Másodsorban figyelembe kellett venni, ki kellett értékelni azokat a mérési eredményeket, statisztikákat, amelyeket a gép müködésének kezdetétől gyüjtöttünk. Végül a nemzetközi folyóiratokban, konferenciák kiadványaiban megjelent cikkek, az intézetben elhangzott előadások tömegesen vetettek fel olyan kérdéseket és ismertettek olyan megoldásokat, amelyek közel álltak az általunk is tapasztalt tényekhez, problémákhoz.

Mindezek alapján célul tüztük ki az oprációs rendszerek egyes részfolyamati matematikai leirásának megadását. A számitások alapján konkrét, a felhasználók számára haszonnal is járó célkitüzések megoldását is feladatainknak tekintettük. Az utóbbi feladatra két lehetőség is mutatkozott:

1. Az output (elsősorban nyomtatás) disk igényeinek dinamikus kezelése.
2. A gép bővitése alkalmával a paraméterek olyan beállitása, hogy valódi multiprogramozás valósuljon meg. Ez a gép központi (CPU, Central Processor Unit) egységének mintegy 20-30%-osan jobb kihasználását tette lehetővé, előzetes számitásaink szerint.

Az 1. pont alatti feladat részbeni megoldását adja a MASTER 4.0 operációs rendszer. Az általunk (lásd Tőke [20])* kidolgozott elvek alapján müködő változat még nem készült el. A 2. pont alatti célkitüzéseket olyan intézkedésekkel valósitottuk meg, melyek a felhasználók számára nem jelentettek a gépbővités során megszorításokat, de a kényelmük sem növekedett nagy mértékben. A valódi multiprogramozást (átlagosan 2–3 programmal) sikerült elérni.

*Az irodalmi hivatkozásokat a cikk második (angol nyelvü) része végén egységesen adjuk meg.

A CDC gép felhasználói programjai rövid jellemzését az alábbiakban lehet megadni:

A kialakult rendszer szerint természetes követelmény annak megvalósitása, hogy a program–próbák (debugging) ideje alatt lehetőleg nagy periféria–igényü programok is fussanak. Erre az ad lehetőséget, hogy a compute–idő mintegy másfélszerese a csatorna–időnek, igy elsősorban a compute–idő kihasználásával kell törődni. Nagy periféria–igényü, hosszu feldolgozási–igényü programokkal a gép féléves müködése óta rendelkezünk.

A SCR jelenlegi felhasználása két egymástól teljesen különböző felhasználási elv alapján történik. A számolásos (elsősorban FORTRAN-ban iródott) jellegü feladatoknál a SCR felhasználás véletlenszerü és véletlenszerü felszabaditásokkal jár. Az összigény nem jelentős, igy az ingadozás átlaggal és szórással jellemezhető volt. A nagy rendezői és kiirási munkák esetén a SCR felhasználása más-más módon valósul meg. Kiiratásoknál a kiirandó adatok véletlenszerü (feltehetően független exponenciális eloszlások szerint) jelennek meg. Ennek az igénynek jó kielégitése a szeletelt nyomtatás segitségével oldható meg. Rendezési (szortolási) feladatok megoldásánál a SCR igény egy nagyterjedelmü igény formájában, véletlen időtartamra jelenik meg és ezután a feladat további részében ez a terület nem kerül felhasználásra. E két nagy, különböző tipusu, SCR igény statisztikai leirása a dinamikus SCR felhasználás bevezetését teszi szükségessé. Számitásaink és méréseink szerint a SCR felhasználását a következő szabályokkal lehetett korlátozni. Az egy programban felhasználható SCR terület nagysága nem haladhatja meg a 140 szegmens értéket. Az output igényt esetleg külön igény formájában lehet megadni jelezve az output igény nagyságát. A dinamikus SCR felhasználása esetén a programok rendelkezésre álló SCR területét 250-300 szegmens körül lehetett meghatározni. Ez biztositotta a különböző tipusu programok egymás meletti futhatóságának feltételét.

A fenti adatokat folyamatos statisztikai felmérések végzésével egy kiértékelő program alapján kaptuk (lásd Tóth B. – Tőke P. [22]). Ezeknek a statisztikáknak az alapján megállapitható, hogy a gépi kihasználás legfontosabb paraméterei lényegében a várt értékek körül mozogtak és lehetőség volt a kiértékelések alapján extrapolációt végezni a gép leghasznosabb kihasználására vonatkozóan a bővités után.

A gyakorlatban elsősorban a programok száma szerinti statisztikákkal dolgoztunk és nem vettük figyelembe (mivel erre nem volt lehetőségünk) az egyes programfajták milyen hosszu ideig foglalják le a gépet. Erre csak a következő durva statisztikát tudjuk adni. A gép idejének mintegy 1/3 részében szervezési feladatok, tehát sok nyomtatást igénylő feladatok futnak, további 1/3 részban számitás–igényes feladatok futnak, mig a megmaradó 1/3 gépidőt kisebb programok töltik ki. A legutóbbi csoport esetén van értelme a programok száma szerinti eloszlásokkal is foglalkozni.

A gép müködésének matematikai leirásához szervesen hozzátartozik a job–folyam (vagy program–folyam) matematikai leirása is.

A job – folyam statisztikai viselkedésének ismeretében olymódon kell a jobok keverését elvégezni, hogy az összes job együttes átfutási ideje minimális legyen, azon kiegészitő feltétel mellett, hogy egy hét alatt minden beérkező job lefusson. A heti felosztáson, keverésen kivül

a jobok napi keverését (prioritásuk adása, napi legalább egyszeri futás biztositása stb.), az operátorok, illetve az erre kijelölt vezető végzi. Ezen kivül szükség van rövidebb időszakon belül (pl. óránként) algoritmikus job keverésre is.

A job–folyamot egy többdimenziós időben változó véletlen igénybeérkezésnek tekintjük, melynek kiszolgálása, a kiszolgálási idő kezdete, a szervezéstől, keverési eljárásoktól és a gépben müködő operációs rendszertől függ. A job–folyamot ilyen sztochasztikus folyamatnak tekintve, és figyelembe véve, hogy a kiszolgálás folyamán a gép prioritási rendszerétől függően változik az átfutási időtartam, helyesnek látszott egy sztochasztikus modell alkalmazása. Mindez annak ellenére is jó közelitésként használható, hogy a gép belső müködése szigorú logikai felépités alapján történik.

A job–folyam és a belső müködés egy sztochasztikus modellje megtalálható Arató – Knuth – Tőke [3] és Knuth [15], [16] cikkeiben, igy erre részletesen nem térünk ki.

2. A matematikai modellekről

Az alábbiakban a számitógép, illetve annak részei müködésének egyes matematikai modelljeivel foglalkozunk.

a.) A matematikai leirások közül első helyen kell kiemelni a tömegkiszolgálási modellek alkalmazását. Ennek összefoglalása a majdnem teljes irodalom megtalálható Kleinrock-nak az 1974 évi IFIP kongresszuson elhangzott előadásában [12]. Az intézetben végzett ilyen irányu vizsgálatokról Tomkó [21] cikke ad felvilágositást.

b.) A "telitett" tömegkiszolgálási prolémák vezettek el analitikus uton a diffuziós közelitések alkalmazásához. Ez a leírás található meg Gaver – Shedler [8],[9] valamint Kobayashi [13], [14] cikkeiben.

c.) A számitógépben lejátszódó folyamatok sztochasztikus közelitése az időintervallum megfelelő megválasztása esetén lehetővé teszi a közvetlen diffuziós leirások használatát. Erre tettünk javaslatot korábbi cikkeinkben: Arató [1], [2], Arató – Knuth – Tőke [3]. A gép belső müködését leíró szochasztikus folyamatok paraméterei ez esetben a job-ok függvényei azaz ismét véletlen paraméterek. A belső prioritásos rendszer kialakitása, a kvantum megválasztása a statisztikák állandó megfigyelésével valósitandó meg.

A matematikai, különösen a stasztikai jellegü, modellek nagy része jelenleg a sorbanállás matematikai elméletének alapján áll. Ugyanúgy mint más elméletek esetén is ezek a modellek sohasem felelnek meg teljes pontossággal a valódi rendszereknek, hanem azok lényegét kivánják megmutatni. A legfontosabb feltételeket és következtetéseket a meglévő adatok sokaságán nem szokás ellenőrizni. Ez igen nehézkes statisztikai vizsgálatok bevezetését jelentené. Ebben a cikkben (mindkét részében) a multiprogramozású számitógépek belső müködése leirására egy diffuziós tipusú közelitő modellt ismertetünk. Egy ilyen modellből származó következtetések

könnyebben ellenőrizhetőek, másrészről ezen a közelitési alapon nyílik lehetőség a sztochasztikus szabályozás, nem lineáris filtráció és más eredmények alkalmazására. Jólismert, hogy a sorbanállás matematikai elméletének eredményei nem egyszerüek és éppen ez gátolja gyors felhasználásukat is.

Korábbi dolgozataink ban (lásd pl. [1], [2], [3], [15], [16]) megmutattuk, hogy a multiprogramozásu számitógépek érdekes jellemzői adhatók meg néhány közvetlen diffuziós közelités analizise utján. Ezek az erdmények felhasználhatók a CPU (a központi egység, central processor unit) vagy a DTU (adatátviteli egység(ek), data transmission unit) müködésének értékelésében.

Az általunk javasolt sztochasztikus modell, mely az operációs rendszer müködését kivánja közelitően megadni, lehetőséget nyújt a számitógép hosszabb időtartamú müködtetése esetén (nagy megterhelésü job–folyamot feltételezve) a müködés általános értékelésére és szabályozására, néhány paraméter segitségével.

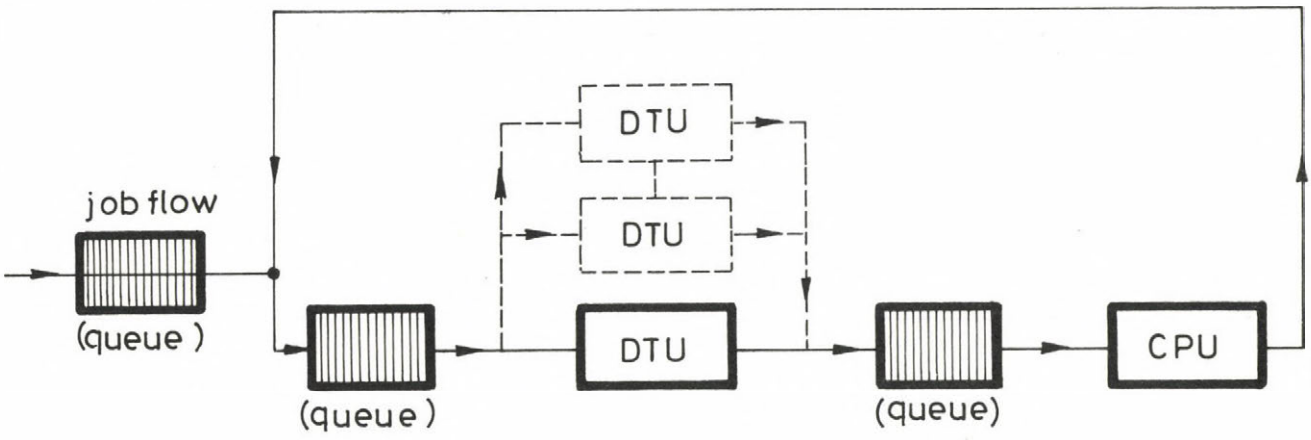
Ebben az esetben a statisztikai analizis során lehetőség van a felesleges matematikai nehézségek elkerülésére, melyek a ciklikus sorbanállási modelleknél jellemzőek (lásd pl. Gaver – Shedler [8], [9] vagy Reiser – Kobayashi [17]). A ciklikus sorbanállási modelleknél a nehézségek legjelentősebb forrása az adatátviteli müveletek nem–Markov mivoltából adódik.

A ciklikus sorbanállási modellek diffúziós közelitése is mutatja ezen hozzáállás egyszerü következtetési eredményeit (lásd [8], [13]). Ezek az eredmények természetes módon vetik fel annak szükségességét, hogy az operációs rendszerek müködésének leirására közvetlen diffúziós közelitéseket használjunk (a sorbanállási modellek nélkül). Ebben a cikkben egyszerü bizonyitásokat kivánunk bemutatni a diffuziós közelités jóságára és azt is megmutatjuk, hogy modelljeink egyszerüen kezelhetőek multiprogramozású rendszereknél. A statisztikai kiértékelések és a nemlineáris sztochasztikus szabályozás lehetőségeinek bemutatása ugyancsak feladata a cikknek. Standard ismereteket tételezzünk fel és a szokásos leirást alkalmazzuk (a) a programok viselkedésére a központi egységben, (b) az információ elérésére az adatátvitelnél, (c) a tárolási hierarchiák megadására. Eredményeink hasznosaknak tünnek egyes elemek vagy nagyobb egységek müködési paraméterei kiválasztására is.

A [15], [16] dolgozataink ban javasolt valószinüségi leirást használjuk a továbbiakban. Feltételezzük, hogy nemcsak a Brown–mozgás, hanem általános diffuziós folyamatok is felléphetnek a számitógépek müködési leirásában. Ennek egyik bizonyitéka, hogy gyakran mérhetőek a kovariancia függvények és ezek lényeges eltérést mutatnak a Brown–mozgás kovariancia függvényétől.

A modellben szereplő programok (job-ok) a központi memória valamely részében vannak elhelyezve, éppúgy mint az operációs rendszer programjai. A lehetséges elhelyezhető programok száma a multiprogramozás foka, amelyről feltesszük, hogy konstans és K-val jelöljük. Ez természetes feltevés leterhelt rendszereknél.

Feltételezzük, hogy (mint pl. [9], [13]-ban) a K program a központi egység és az adatátviteli egységek által alkotott ciklusban müködik. Az egyes programok vagy a központi egységre várnak vagy ott állnak kiszolgálás alatt, melynek végeztével adatátviteli egységre kerülnek. Az adatátviteli egységről, a szükséges információ megkapása után, a program visszatér a központi egységre. Ez a ciklikus folyamat meghatározatlan ideig folytatódik. Ha egy program kiszolgálása befejeződik és kikerül a rendszerből, helyére a program–folyamból új program kerül a központi memóriába. Az 1. ábra szemléletesen mutatja a folyamatot.



1. ábra

A programok viselkedésére vonatkozóan a következő feltevésekkel élünk:

a.) a központi egységben történő kiszolgálási idők független valószinüségi változók (esetleg nem azonos eloszlásuak);

b.) az adatátviteli egységek kiszolgálási idejei ugyancsak független valószinüségi változók;

c.) a központi egység és adatátviteli egységek kiszolgálási idejei egymástól teljesen függetlenek.

Megjegyezzük, hogy függő valószinüségi változók esetén nem jutunk Brown–mozgás közelitére, hanem általánosabb diffúziós folyamatra. Feltételezzük, hogy a központi egység előtti sor és az adatátviteli egységek előtti esetleges sor kiszolgálása a FIFO (először érkezett–először kerül kiszolgálásra) elv alapján történik, ha ezt másként nem szabályozzuk.

Az előbbiekben leirtak szemléltetésére és a nagyságrendek érzékeltetésére példaként vázlatosan ismertetjük az akadémiai CDC gépnek a leiráshoz szükséges adatait.

A CDC – 3300 gép egy négy programot ($K = 4$) megengedő rendszerének sematikus általánositásából adódó modell a következő. A kvantum idő, melyet Q-val jelölünk jelenleg 1/10 sec. Az I/0 megszakitások adminisztrációja, mind az input, mind az output adminisztrációnál $\frac{1}{2}\,\mu$ sec. Ezen adminisztráció végzése alatt nincs még kvantum megszakitás sem. Esetünkben az adatátviteli egységeket a csatornák jelentik, melyek a perifériális egységekkel kötik össze a központi egységet. Feltételezve egy abszolut prioritásos kiszolgálási rendszert (mely

I/0 megszakitásnál érvényes) az egyes programokról a következőket tételezzük fel:

az 1. program (job) CPU kiszolgálási ideje 1μ sec átlagos idejü és a csatornán való tartózkodás ugyancsak 1μ sec átlagos idejü valószinüségi változó.

a 2. job 5μ sec átlagos CPU igénnyel és 4μ sec átlagos idejü csatorna tartozkodásu:

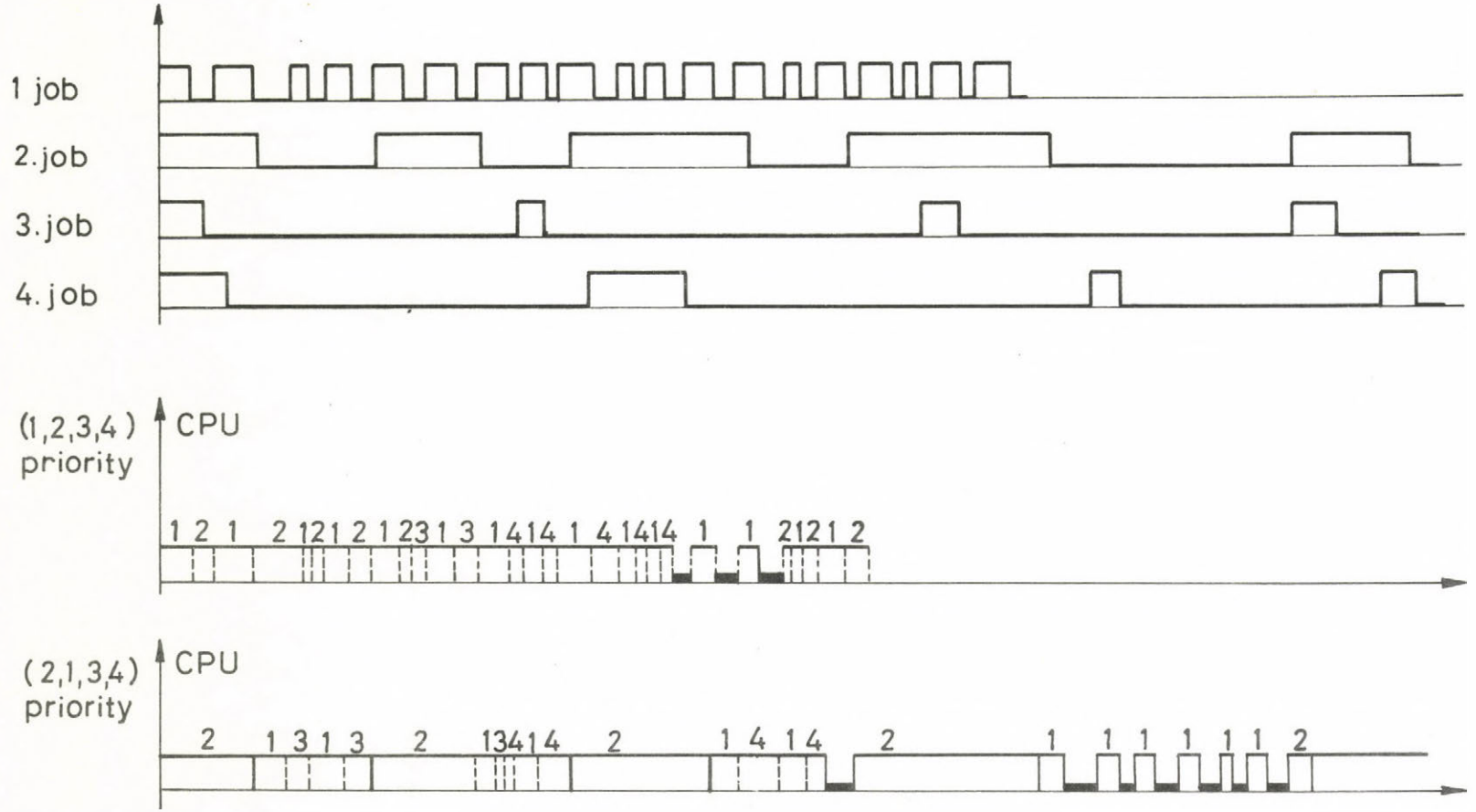
a 3. job 11μ sec átlagos idejü CPU kiszolgálással míg a csatornán 10μ sec átlagos idővel van,

a 4. job 2μ sec átlagos idővel a CPU-ban és 10μ sec átlagos idővel a csatornán van.

Ha feltételezzük, hogy az abszolut prioritási rendszer az első esetben (1, 2, 3, 4) a második esetben pedig (2, 1, 3, 4) és a CPU kihasználtságot és a csatornák foglaltságát a két prioritás esetén valamint a round–robin elv alapján egy kvantum megszakitás környezetében ábrázoljuk, érdekes következtetésekre jutunk (v.ö. 2. ábra).

A rövid csatorna idejú és sok megszakitásos programok dominálnak. A szemlélet alapján azt lehet várni, hogy a CPU kihasználtsága annál a prioritási rendszernél jobb, melyben az 1. job megelőzi a 2-t. Ekkor ugyanis a 2. job csatorna ideje, az 1. job csatorna ideje és CPU ideje alatt is folyik. Ellenkező prioritás esetén az 1. job csatorna ideje lényegében a 2. job csatorna ideje alatt folyik és nem folyik a CPU ideje alatt. A gyors I/0 változásnál a kvantumnak nincsen jelentős szerepe a CPU kihasználtságában. A kvantum a jelentős CPU igényü feladatoknál játszik lényeges szerepet, amikor az I/0 megszakitások és CPU idők nagyságrendje a kvantum, vagy annál nagyobb. Ilyen esetekben a kvantum jelentősen módositja a sorrendet, és a prioritás elveszti elsődleges meghatározó szerepét. A 2. ábrán egy szimulációs eredményt ábrázoltunk, ahol az eloszlások exponenciálisak.

A rendszer leírásának első közelítésében nem vesszük figyelembe az I/0 megszakitásokkal járó adminisztrációt, mivel a jelenlegi rendszerben azt a CPU végzi és a CPU idejében is történik az elszámolás. Amennyiben a megszakitások száma nagy, és ez az adminisztráció jelentős, természetes az olyan rendszerek kiépitése, ahol ezt az adminisztrációs feladatot nem a CPU, hanem egy külön egység végzi.


1 job
2.job
3.job
4. job
(1,2,3,4)
priority
CPU
1 2 1 2 121 2 1 231 3 1414 1 4 1414 1 1 2121 2
(2,1,3,4)
priority
CPU
2 1 3 1 3 2 13413 4 2 1 4 1 4 2 1 1 1 1 1 1 2

2. ábra

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

ON THE DIFFUSION APPROXIMATION OF OPERATING SYSTEMS II.

Mátyás Arató

Computer and Automation Institute, Hungarian Academy of Sciences

I. INTRODUCTION

In the first part of the paper (in hungarian) we gave a brief account of our experiments on CDC – 3300. In this part we give the mathematical description of a diffusion approximation of the operating systems.

First we try to indicate the intuitive reasons of the diffusion approximation. It is well known that the Brownina – motion process $\omega(t)$ $(0 \leqslant t < \infty)$ may be obtained as the limit of the following process (see e.g. Feller [6]). Let $N(t)$ denote a Poisson process, with parameter λ, i.e. the distribution of the time interval between two events be exponential and the random variables are independent.

After an event the particle moves up or down with the value a, with probability 1/2. Let $\omega_{\lambda,a}(t)$ denote the displacement of a Brownian particle at time t, then

$$\omega_{\lambda,a}(t) = \sum_{n=0}^{N(t)} \eta_n$$

where

$$\eta_n = \begin{cases} a, & P(\eta_n = a) = 1/2 \\ -a, & P(\eta_n = -a) = 1/2 \end{cases}$$

The following diagram indicates a possible realization:

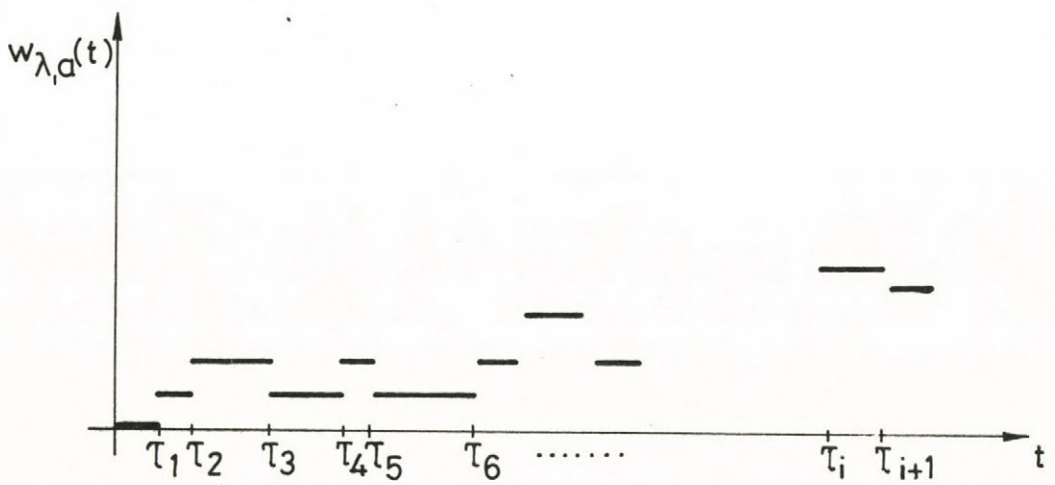


Fig. 1.

τ_i denotes the i-th jump time of the particle, where

$$P(\tau_i - \tau_{i-1} < t) = 1 - e^{-\lambda t}.$$

With standard methods it can be proved that

$$\log E\, e^{i\omega_{\lambda,a}(t)\cdot s} = 1/2\, s^2 \sigma^2 t + a\Theta |s|^3 \sigma^2 t$$

where $\sigma^2 = a^2\lambda$, $|\Theta| \leqslant 1$. Assuming that $\lambda \to \infty$, $a \to 0$ and $a^2 \cdot \lambda = \sigma^2 = \text{const}$ we obtain

$$\log E\, e^{i\omega_{\lambda,a}(t)\cdot s} \to -\frac{1}{2} s^2 \sigma^2 t$$

This means (heuristically) that the limit process $\omega(t) = \lim \omega_{\lambda,a}(t)$ is a Brownian motion. The exact interpretation and proofs of such type of theorems are found in Gihman–Skorohod's [10] (§3 ch. 9) or Borovkov's [4] books (§24). In the latter the "heavy traffic" conditions of queuing theory are investigated.

1. The CPU utilization problem.

First let us assume that there are one CPU and two DTU-s and there are 2 jobs in the computer. For the first job, the CPU and DTU service times η_{1i} (resp. ξ_{1i}) have the distribution

$$P(\eta_{ni} < t) = P(\xi_{1i} < t) = 1 - e^{-\lambda_1 t}$$

and they are independent. For the second job, the distributions are

$$P(\eta_{2i} < t) = P(\xi_{2i} < t) = 1 - e^{-\lambda_2 t}.$$

It seems that one of the most interesting utilization problems is connected with the absolute priority rule. This means that if job 1 has absolute priority then, upon an I/0 interrupt, the CPU is assigned to job 1 if it wants to take it* and the second job is waiting or it is in the DTU. As the system has two DTU-s, there is no queue before them. If, however, job 2 had absolute priority then it would no wait for the CPU. The first priority rule will be denoted by $v(1,2)$ and the seconed by $v(2,1)$.

In the sequel we assume that the mean service time of job 1 is much less than of job 2. This requirement is usually satisfied in practice and, for this case, we prove the following statement.

* The so-called pre-emptive priority for CPU

Theorem 1. *We assume that* $1/\lambda_1 << 1/\lambda_2 << 1$, *then the priority rule* $\nu(1,2)$ *gives the CPU utilization time* ζ_t *at absolute time* t *as*

$$\zeta_t^{\nu(1,2)} = \sum_1^{N(t)} \eta_{1i} + \sum_1^{N(t')} \eta_{2i}, \tag{1.1}$$

where

$$t'' = t - \sum_{i=0}^{N(t)} \eta_{1i} < t,$$

which is asymptotically normally distributed with parameters.

$$E\zeta_t^{\nu(1,2)} \approx \frac{1}{2}t + \frac{1}{2} \cdot \frac{2}{3}t = \frac{5}{6}t \tag{1.2}$$

$$D^2\zeta_t^{\nu(1,2)} \approx t\left(\frac{1}{\lambda_1} + \frac{2}{3} \cdot \frac{1}{\lambda_2}\right) \tag{1.3}$$

Further, the priority rule $\nu(2,1)$ gives the CPU utilization time

$$\underline{\zeta}_t^{\nu(2,1)} = \sum^{N(t)} \eta_{2i} + \sum^{N(t'')} \eta_{1i}, \tag{1.4}$$

where

$$t'' = t - \sum_{i=1}^{N(t)} \eta_{2i} < t,$$

which asymptotically has also Gaussian distribution, wich parameters

$$E\zeta_t^{\nu(2,1)} \approx \frac{1}{2}t + \frac{1}{2} \cdot \frac{1}{2}t = \frac{3}{4}t \tag{1.5}$$

$$D^2\zeta_t^{\nu(2,1)} \approx t(\frac{1}{\lambda_2} + \frac{1}{2} \cdot \frac{1}{\lambda_2}) \tag{1.6}$$

Proof. The random variable $N(t)$ (resp. $N(t)$) has Poisson distribution with parameter $\frac{\lambda_1}{2} \cdot t$ (resp. $\frac{\lambda_2}{2} \cdot t$). First we prove that the logarithm of the characteristic function of the stochastic process

$$x_1(t) = \zeta_{t,1}^{\nu(1,2)} = \sum_{i=1}^{N(t)} \eta_{1i}$$

has the from

$$\log Ee^{isx_1(t)} = ist \cdot \frac{1}{2} - \frac{1}{2}s^2 t\frac{1}{\lambda_1} + \Theta|s|^3 t \frac{\lambda_1}{2} \frac{3!}{\lambda_1^3}. \tag{1.7}$$

This follows from the relations

$$Ee^{isx_1(t)} = Ee^{is\sum_{k=1}^{N(t)}\eta_{1k}} = EE[e^{is\sum_{k=1}^{n}\eta_{1k}} | N(t) = n] =$$

$$= \sum_{n=0}^{\infty} Ee^{is\sum_{k=1}^{n}\eta_{1k}} \cdot \frac{e^{-\frac{\lambda_1}{2}t}\left(\frac{\lambda_1}{2}t\right)^n}{n!} =$$

$$= \sum_{n=0}^{\infty} (Ee^{is\eta_{11}})^n \frac{e^{-\frac{\lambda_1}{2}t}\left(\frac{\lambda_1}{2}t\right)^n}{n!} =$$

$$= e^{\frac{\lambda_1}{2}[Ee^{is\eta_{11}} - 1]}$$

and (using the moments of the exponential distribution and the expected value)

$$E(e^{is\eta_{11}} - 1) = isE\eta_{11} - \frac{1}{2}s^2E\eta_{11}^2 + \Theta|s|^3E|\eta_{11}|^3 =$$

$$= is\frac{1}{\lambda_1} - \frac{1}{2}s^2\frac{}{\lambda_1^2} + \Theta|s|^3\frac{3!}{\lambda_1^3},$$

where $|\Theta| \leqslant 1$.
In the same way

$$\log Ee^{isx_2(t)} = ist\frac{1}{2} - \frac{1}{2}s^2t\frac{1}{\lambda_2} + \Theta|s|^3t\frac{1}{\lambda_2^2} \tag{1.8}$$

where

$$x_2(t) = \sum_{i=1}^{N(t)} \eta_{2i}$$

In the second step, there is a great difference between the priority rules ν (1,2) and ν (2,1) as the sums

$$\sum_{i=1}^{N(t')} \eta_{2i} \quad \text{and} \quad \sum_{i=1}^{N(t'')} \eta_{1i}$$

behave in different ways. See Fig.2.
In case $\nu(1,2)$ the variables η_{2i} remain further exponentially distributed with parameter λ_2, but on the CPU time axis the DTU periods ξ_{2i} run quicker and are exponentially distributed with parameter

$$\tilde{\mu}_2 = \lambda_2 \frac{2\lambda_1}{\lambda_1} = 2\lambda_2$$

(the reason is that the compute time of the first job overlaps with the DTU time of the second job).

With a similar argument as used to prove (7), one can also prove that for process

$y_1(t) = \sum_{i=1}^{N(t'')} \eta_{2i}$ (note that t" is also a random variable)

$$(1.9) \qquad \log Ee^{isy_1(t)} = ist\frac{1}{2}\cdot\frac{\tilde{\mu}_2}{\mu_2+\lambda_2} - \frac{1}{2}s^2t\frac{1}{2}\frac{2\tilde{\mu}_2}{\mu_2+\lambda_2}\cdot\frac{1}{\lambda_2} + \Theta|s|^3\frac{t}{2}\frac{3}{\lambda_2^2} =$$

$$= ist\frac{1}{2} - \frac{1}{3}s^2t\frac{2}{3}\frac{1}{\lambda_2} + \Theta|s|^3\frac{t}{2}\frac{3}{\lambda_2^2}$$

(7) and (9) prove (2) and (3). In case of $\nu(2,1)$, η_{1i} is further exponentially distributed with parameter λ_1. The DTU periods ζ_{1i} remain the same during a DTU period of job 2 (here are only end effects), and during the CPU period of the second job both the CPU and DTU periods of job 1 are staying (waiting). Only end effects may be at the beginning of the CPU period. This means that with a similar argument as in proof (7) we obtain for process

$y_2(t) = \sum_{i=1}^{N(t'')} \eta_{1i}$ (note again t'' is a random variable)

$$(1.10) \qquad \log Ee^{isy_2(t)} = ist\frac{1}{2}\cdot\frac{1}{2} - \frac{1}{2}s^2t\frac{1}{2}\frac{1}{\lambda_1} + \Theta|s|^3\frac{t}{2}\frac{3}{\lambda_1^2}$$

Relations (8) and (10) prove (5) and (6). The Theorem is proved.

Heuristically Theorem 1 involves that process $\zeta_t^{\nu(1,2)}$ and $\zeta_t^{\nu(2,1)}$ converge (weakly) to Brownian motion processes and have different local parameters.

REMARK 1. Let the CPU and DTU periods for job 1 be exponentially distributed with parameters λ_1 and μ_1 and let they be for the second job λ_2, μ_2. With a similar argument as in Theorem 1 we may prove that, if $\mu_1 \gg \mu_2$ priority $\nu^{(2,1)}$ leads to the CPU utilization time $\zeta_t^{\nu(1,2)}$, which has an approximate normal distribution with parameters

$$(1.11) \qquad E\zeta_t^{\nu(1,2)} \approx t\frac{\mu_1\lambda_1}{\mu_1+\lambda_1}\frac{1}{\lambda_1} + t\left(1-\frac{\mu_1}{\mu_1+\lambda_1}\right)\cdot\frac{\tilde{\mu}_2}{\mu_1+\lambda_2},$$

$$(1.12) \qquad D^2\zeta_t^{\nu(1,2)} \approx t\frac{2\mu_1}{\mu_1+\lambda_1}\frac{1}{\lambda_1} + t\left(1-\frac{\mu_1}{\mu_1+\lambda_1}\right)\frac{2\tilde{\mu}_2}{\mu_2+\lambda_2}\frac{1}{\lambda_2}.$$

where

$$\tilde{\mu}_2 = \mu_2\frac{\mu_1+\lambda_1}{\mu_1}.$$

When priority rule $\nu(2,1)$ is in force we have

(1.13) $$E\xi_t^{\nu(2,1)} = t\frac{\mu_2\lambda_2}{\mu_2+\lambda_2}\frac{1}{\lambda_2} + t\left(1-\frac{\mu_2}{\mu_2+\lambda_2}\right)\frac{\mu_1}{\mu_1+\lambda_1}$$

and

(1.14) $$D^2\zeta_t^{\nu(2,1)} = t\frac{2\mu_2}{\lambda_2+\mu_2}\frac{1}{\lambda_2} + t\left(1-\frac{\mu_2}{\mu_2+\lambda_2}\right)\frac{2\mu_1}{2\mu_1+\lambda_1}\cdot\frac{1}{\lambda_1}$$

It is remarkable that, with $\mu_2 \leqslant \mu_1$ the priority rule $\nu(1,2)$ is always better then $\nu(2,1)$, as (with the notation)

$$p_i = \frac{\mu_i}{\lambda_i+\mu_i} \qquad (i=1,2),$$

(1.15) $$p_1 + (1-p_1)\frac{\frac{p_1\mu}{p_1\mu_2+\lambda_2}}{p_1\mu_2+\lambda_2} > p_2 + (1-p_2)p_1 = p_1+p_2-p_1p_2$$

REMARK 2. Theorem 1 suggests that there should be an analogue of it for more general service distributions. Suppose, for example, that the independent, not necessarily indentically distributed random variables η_{ki} and ξ_{ki}, $k = 1,2$ $i = 1,2,\ldots$, have finite second moments then it is possible to deduce the statement of Theorem 1 assuming only that $E\xi_{1i} \ll E\xi_{2i}$ (for all is).

In case $E\xi_{1i} = b_1 \sim E\xi_{2i} = b_2$ there are examples, with non exponental distributions, when priority rule $\nu(2,1)$ is better then $\nu(1,2)$ and $b_1 < b_2$ fulfils (see Tomkó [21]).

REMARK 3. It is possible to deduce the Brownian motion approximation for more than 2 DTU-s. Here we shall not give the results.

Theorem 1 above is a useful aid for giving the time interval where the Brownian motion approximation holds. In order to use the result for exponential service times, we need to assume $t \gg \lambda_2$.

The major result of Theorem 1 is in the relation between the priority rules. The theorem estimates "how much" time the CPU must spend in service for the different priority rules. Using these estimates for instants $t_1 < t_2 < \ldots < t_n < \ldots$, where $t_i - t_{i-1} \sim c\,\lambda_2$ (with $c \gg 1$) it is possible to construct a stochastic control model with Gaussian random variables, as it was proposed in our paper [3].

A further approximation is the following. The discrate time process $\zeta_{t1}, \zeta_{t2}, \ldots \zeta_{tn}$ may be handled as a continuous process, if n is large enough. In this case the statistical investigation of the CPU utilization process ζ_t means the following. Let ζ_t denote the CPU utilization time in absolute time t, then it satisfies the equation

(1.16) $$d\zeta_t \doteq mdt + \sigma d\omega(t)$$

where m and σ depend on the priority rule $\nu(i_1,\ldots,i_k)$. To estimate these parameters, one has to know the number of interrupts for every job and their own CPU utilization time,

which depends on the priority rule. To guarantee CPU service, for every job, the computer has a cyclic service with quantum lenght (with round–robin service rule). The round–robin rule ensures the short turn around of short jobs and, in our description, the possibility to measure the CPU utilization time for every job.

2. An approximation of the cyclic queue model

By using the cyclic queue model for one CPU and one DTU (see Fig.1. in part I.), Gaver and Shedler [8], [9] and independently Kobayashi [13], [14] examined the distribution of the number of programs $N_c(t)$ present at the CPU at time t, including those queued in addition to the program currently being serviced. Throughout this section it will be assumed that we have one CPU and one DTU.

Let $A(t)$ represent the number of arrivals at the CPU in $(0,t)$ and $D(t)$ the number of CPU departures in $(0,t)$. We assume that $N_c(0) = 0$ and

$$N_c(t) = A(t) - D(t)$$

The $A(t)$ and $D(t)$ processes are approximately normally distributed with means $E(A(t)) = t/E(\xi_1)$, $E(D(t)) = t/E(\eta_1)$ and variances $D^2(A(t)) = tD^2(\xi_1)/[E(\xi_1)]^3$, $D^2(D(t)) = tD^2(\eta_1)/[E(\eta_1)]^3$. Here ξ_i means the service time in DTU, η_i the same in the CPU. We assume that $E\xi_i > E\eta_i$ and that the ξ_i-s are identically distributed independent random variables and the same is supposed for η_i, $i = 1,2, \ldots$. The $A(t)$ and $D(t)$ processes are not independent: for the special case $E(\xi)/E(\eta) \approx 1$ ("heavy traffic" condition) it can be proved by the method of Borkovov [4] (§ 24) that $N_c(t)$ is approximately a Brownian–motion process and satisfies the equation

$$dN_c(t) = \mu dt + \sigma^2 d\omega(t) \tag{2.1}$$

where $\omega(t)$ is the standard Brownian–motion process with drift

$$\mu = 1/E(\xi) - 1/E(\eta) \tag{2.2}$$

and infinitesimal variance

$$\sigma^2 = D^2(\xi)/[E(\xi)]^3 + D^2(\eta)/[E(\eta)]^3$$

The $N_c(t)$ process (neglecting the boundary effects at 0 and K which cause the dependence of $A(t)$ and $D(t)$ too) has to be in the interval $[0,K]$, $0 \leqslant N_c(t) \leqslant K$, i.e. 0 and K are reflecting barriers. This means

$$P(N_c(t) < 0) = 0 \quad \text{and} \quad P(N_c(t) \leqslant K) = 1. \tag{2.3}$$

If the number K of programs is unlimited, the behaviour of a cyclic queue model is well

approximated with an ordinary sigle–server system, in which there is no restriction upon the number of waiting customers (see Feller [5], 194–198)

(2.4) $\quad M_n = \max(0, S_1, S_2, \ldots S_n)$

where

$$S_n = X_1 + \ldots + X_n, \qquad X_i = \eta_i - \xi_i, \qquad E(\eta_i - \xi_i) = -\delta < 0 \ ,$$
$$D^2(\eta_i - \xi_i) = \sigma_0^2 \ .$$

Now, and this is new in our treatment (not used by Gaver–Shedler [9]), we use the well known formulas of sequential hypothesis testing for the Brownian motion process. Let $\omega(t)$ denote the standard Brownian motion process with $\omega(0) = x$, $E\omega(t) = 0$, $E\omega^2(t) = 1.t$, and two barriers $A < x < B$. Let

(2.6) $\quad \lambda^x(t) = x + \frac{r}{\sigma^2}(\sigma\omega(t) - \frac{r}{2}t), \quad r > 0, \quad \sigma > 0$

be a process, and let $\tau^x_{A,B}$ be the first time point (Markov–point) where $\lambda^x(t)$ goes out from $[A,B]$ i.e.

(2.7) $\quad \tau^x_{A,B} = \inf\{t \geqslant 0 : \lambda^x(t) \bar{\in} (A,B)\}$

Then (see Shiryayev [19] p. 182)

(2.8) $\quad P\{\lambda^x(\tau^x_{A,B}) = B\} = \frac{e^x - e^A}{e^B - e^A} \ .$

From this formula we obtain, if $A \longrightarrow \infty$.

(2.9) $\quad P\{\lambda^x(\tau^x_{-\infty,B}) = B\} = e^{x-B}$

and further, if $x = 0$,

(2.10)

$$P\{\lambda(\tau_{-\infty,B}) = B\} = e^{-B} \ .$$

Using the Brownian–motion approximation for S_n and M_n we obtain that, for $n \longrightarrow \infty$ (using theorem 2.ch.IX.§3 in Gihman–Skorohod's book [10]).

$$P\{W_n > \sqrt{n}\,B\} = P\{M_n > \sqrt{n}\,B\} =$$

$$= P\left\{ \max_{1 \leqslant i \leqslant n} \left(\frac{\delta_i + i\,\delta}{\sqrt{n}\,\sigma_0} - \frac{i}{\sqrt{n}} \frac{\delta}{\delta_0} \right) > \frac{B\sqrt{n}}{\sigma_0 \sqrt{n}} \right\} \approx$$

$$\approx P\left\{ \sup_t \omega_t > t\frac{\delta}{\sigma_0} + \frac{B}{\sigma_0} \right\} \approx$$

$$\approx P\left\{ \omega_\tau - \tau \frac{\delta}{\sigma_0} = \frac{B}{\sigma_0} \right\} =$$

$$= P\left\{ \frac{2\delta}{\sigma_0} \left(\omega_\tau - \tau \frac{\delta}{\sigma_0} \right) = \frac{2\delta}{\sigma_0^2} B \right\} = e^{-\frac{2\delta}{\sigma_0^2} B}.$$

Further the proof is the same as in the cited paper [7]. The number Q of customers in the queue is the number that arrive during the waitting time of an arbitrary customer. We investigate the stationary distributions of both W_n and Q. If $G(x)$ is the distribution of ξ and* represents Stieltjes convolution, then

(2.11) $$P\{Q \geqslant k | W = x\} = G^{k^*}(X)$$

and

(2.12) $$P\{Q \geqslant k\} = \int_0^\infty G^{k^*(x)}\, e^{-cx}\, c\,dx = C[\hat{G}(c)]^k$$

where $\hat{G}(c)$ is the Laplace–Stieltjes transform of G, evaluated at c. Relation (12) includes the following theorem:

Theorem 2. *The number Q of jobs in the CPU queue has asymptotical exponential distribution;*

(2.13) $$P\{Q \geqslant x\} \approx Ce^{x \ln G(c)},$$

where

$$c = \frac{2\delta}{\sigma_0^2}$$

This means that, under heavy traffic conditions ($\rho = E(\xi)/E(\eta) \sim 1$), the stationary distribution of the number of costumers in the system is exponential. This result is quite the same as in Gaver–Shedler's [8] diffusion approximation, where they obtained

(2.14) $$P\{Q \geqslant x\} = C' e^{\frac{2\mu}{\sigma^2} x}$$

where μ, σ are given by (2).

When $\sigma \sim 0$ then with the Taylor series expasion, we can easily prove that (12) and (14) give the same approximation. In this case $-\delta \sim \mu$ and $\sigma_0^2 \sim \sigma^2$.

The reader may find numerical results in Gaver and Shedler's cited papers.

In our paper [3] we have used diffusion type processes (first and second order autoregressive processes) for the approximation of the number of I/0 interrupst over a long time period of the CPU. The above model shows the legality of such heuristic approximations.

The same first order, Gaussian, autoregressive model with discrete time parameter may be used for the working set size $\omega(t,T)$ introduced by Denning [5]. This means that $\omega(t,T)$ is normally distributed and it satisfies the stochastic difference equation

$$\omega(t+1,T) = \rho\, \omega(t,T) + \epsilon(t+1)$$

where $\epsilon(t)$ is an independent, normal random sequence. T is the window and it is fixed. In the sequence $r_{t-\tau+1}, \ldots, r_t$ the number of different page numbers (where r_i means the number of page at time i) is denoted by $\omega(t,T)$. The working set principle for memory management means a dynamical page treatment in the main memory. Here we do not elaborate on this problem.

References

[1] M. Arató, On computing technolgy for mathematical statistics and stochastic processes and its application for evaluating computer system performance, Proceedings of the Computer Science Conference, Székesfehérvár, Hungary 231–237 (1973).

[2] M. Arató, Diffusion approximation for multiprogrammed computer systems. Computer and Math. with Applications (in print)

[3] A.A. Borovkov, Stochastic processes in queuing theory (in Russian), Nauka, Moskwa (1972).

[4] M. Arató, E. Knuth, P. Tőke, On Stochastic Control of a Multiprogrammed Computer based on a Probabilistic Model, IFAC Symposium on Stochastic Control Budapest, 305–311 (1974).

[5] P.Denning, On modeling program behavior, AFIPS, Conf.Proc.Vol. 40, 937–944, (1972).

[6] W. Feller, An Introduction to probability theory and its applications, Vol.II, Wiley, New York (1966).

[7] W. Freiberger, Statistical computer performence evaluation, Academic Press (1972).

[8] D.P. Gaver and G.S.Shedler, Processor Utilization in Multiprogramming systems via Diffusion Approximations, Oper. Research, 21 No. 2, 569–576 (1973)

[9] D.P. Gaver and G.S. Shedler, Approximate Models for Processor Utilization in Multiprogrammed Computer Systems, SIAM J. Comp.2, No.3, 183–192 (1973).

[10] J.I. Gihman and A.V. Skorohod, Introduction to the theory of stochastic processes (in Russian), Nauka, Moskwa (1965).

[11] R.Haji, G.F. Newell, A relation between stationary queue and waiting time distributions, J.Appl. Probability 8, 617–620 (1971).

[12] L. Kleinrock Resuorce allocation in computer systems and computer–communication networks IFIP Congress 74, Inf. Processing 1, 11–18.

[13] H. Kobayashy, Applications of the diffusion approximation to queuing networks I. Equilibrium Oueue Distributions, Journal ACM 21, No. 2, 316–328 (1974).

[14] H.Kobayashi, Applications of the diffusion approximation to queueng networks II. Nonequilibrium distributions and applications to computer modeling, Journ. ACM, 21, 459–469 (1974).

[15] E.Knuth, Multiprogramozású számitógép rendszerek, (in Hungarian), Mérés és Automatika (1974) 5, 208–211.

[16] E.Knuth, A structured computer system model (in print, Computers and Mathematics with applications, this volume).

[17] M.Reiser and H.Kobayashi, The Effects of Service Time Distributions on System Performance, Information Processing IFIP, 230–234 (1974).

[18] R.J. Rodriguez and J.O. Dupuy The evalution of a time–sharing page demand system AFIPS, Vol.40., 759–765.

[19] A.N. Shiryayev, Statistical sequential analysis (in Russian) Nauka, Moskwa (1969).

[20] P.Tőke, Some remarks concerning the simulation of main parts of the master operating system, and about the queuing models of the I/0 system, Proceedings of the Computer Science Conference Székesfehérvár, Hungary, 121–123 (1973).

[21] J.Tomkó, Processor utilization study (in print, Computer and Mathematics with applications, this volume).

[22] B.Tóth and P. Tőke, A scratch–pool kihasználtságának mérése

Р е з ю м е

О диффузионном приближении операционных систем I–II

Матиаш Арато

В статьях даётся общее описание опыта статистических измерений на машине CDC 3300 в Институте Вычислительной Техники и Автоматизации ВАН. Показывается, что приближение работы операционной системы с помощью диффузионных процессов является естественным в длинном периоде работы машины. Такое приближение использовалось раньше в статьях Gaver-Shedler и Kobayashi для моделей теории очереди.

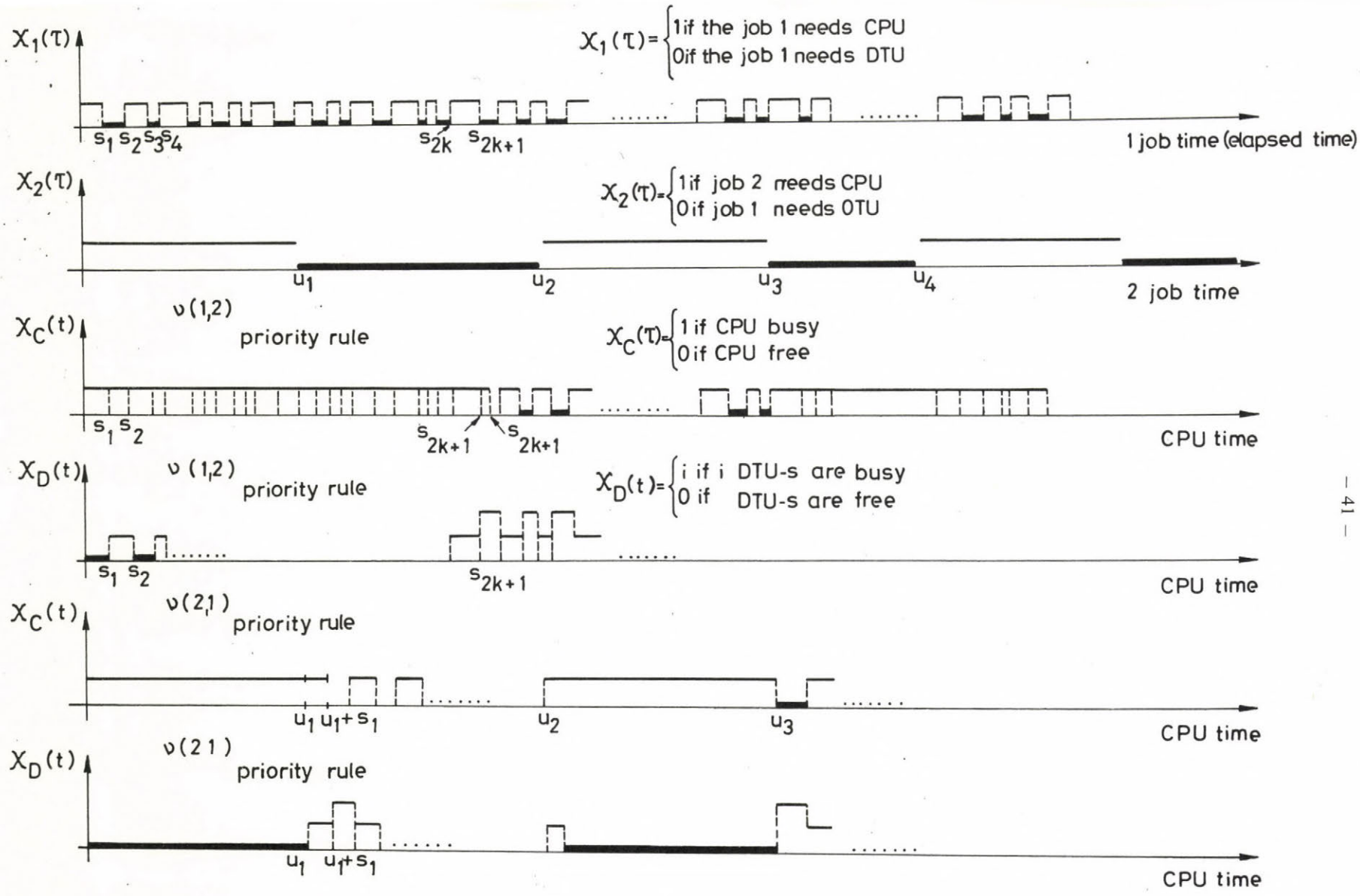

X1(τ)
X1(τ) = 1 if the job 1 needs CPU
0 if the job 1 needs DTU
s1 s2 s3 s4
s2k s2k+1
1 job time (elapsed time)
X2(τ)
X2(τ) = 1 if job 2 needs CPU
0 if job 1 needs DTU
u1 u2 u3 u4
2 job time
XC(t)
ν(1,2) priority rule
XC(τ) = 1 if CPU busy
0 if CPU free
s1 s2
s2k+1 s2k+1
CPU time
XD(t)
ν(1,2) priority rule
XD(t) = i if i DTU-s are busy
0 if DTU-s are free
s1 s2
s2k+1
CPU time
XC(t)
ν(2,1) priority rule
u1 u1+s1 u2 u3
CPU time
XD(t)
ν(2 1) priority rule
u1 u1+s1
CPU time

Fig. 2.

MTA Számítástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

Об опыте разработки и исследования одного класса автоматизированных систем обработки данных

И.В.Сергиенко, И.Н. Парасюк

В настоящем докладе обобщается опыт разработки, исследования и использования универсально-специализированных систем обработки данных на примере таких систем этого класса, как автоматизированная система обработки статистических данных [1] и ее новая модификация - система УСОД [2, 3]. Эти системы разработаны в Институте кибернетики АН УССР за период с 1966 по 1973 год. Системы функционируют на ЭВМ семейства М-220 и ориентированы на обработку статистических данных.

В связи с тем, что система УСОД обладает более широкими возможностями и более совершенной структурой, чем система [1], приведем некоторые параметры именно системы УСОД.

Система УСОД расчитана на стандартный комплект оборудования ЭВМ М-220. Математическое обеспечение этой системы размещено на одной МЛ. В процессе работы система УСОД использует оперативную память общим объемом 8К, внешнюю память на МБ общим объемом 24К. Система может принимать задание через терминальное устройство типа телетайпа. Общий объем системы около 50К стандартных машинных слов. Временные показатели системы УСОД обрабатывает задание пользователя и создает алгоритм решения среднего пакета задач в течение 15-16 сек; в общем случае время работы управляющей программы не превышает одной минуты. Время счета зависит от объема статистических данных (в среднем 5-10 мин.).

В основе метода построения системы лежит идея рационального сочетания принципов универсальности и специализации, сущность которого состоит в следующем. Для каждого класса допустимых задач обработки данных разрабатывается банк программных модулей и вычислительные схемы методов решения задач данного класса. Программные модули и вычис-

лительные схемы увязываются в систему с гибкой и открытой для изменения структурой и координируются некоторой универсальной управляющей программой. Такая программа должна быть способна по заданию, написанному на некотором входом языке системы (как правило, сходном с естественным языком), создать вычислительную схему алгоритма реализации задания, а затем согласно этой вычислительной схеме увязать соответствующие программные модули в программу и обеспечить ее выполнение (или выполнение некоторой другой операции системы). Кроме того, управляющая программа должна автоматически собирать и накапливать опыт функционирования системы, изучать его и осуществлять адаптацию системы в направлении оптимизации параметров качества ее функционирования, а также находить компромиссное решение в выборе уровней универсальности и специализации при определении классов допустимых задач обработки данных.

Всесторонне изучив проблему автоматизации процесса обработки данных с помощью ЭВМ авторы пришли к выводу, что система обработки данных должна решать задачу, постановку которой можно вкратце сформулировать следующим образом.

Существуют некоторые конечные множества: множество типов данных $D = d_k$, множество задач $Z = z_j$, образующее класс допустимых задач обработки данных множества D, и множество алгоритмов $A = A_i$ решения задач множества Z . В зависимости от возникающей ситуации требуется решить некоторой подмножество (пакет) задач $Z \subseteq Z$. При этом, в зависимости от характера исследований, появляется необходимость решить экстремальную задачу по выбору наилучшего алгоритма решения задач данного пакета в смысле некоторого критерия оптимальности (например, минимизировать время решения задачи, минимизировать стоимость решения задачи, построить наилучший по точности решения алгоритм и др.). К тому же мощности множеств Z и D с течением времени существенно меняются: появляются новые элементы этих множеств, некоторые из них заменяются или исключаются, к алгоритмам множества A предъявляются новые требования. Это приводит к необходимости тщательного изучения алгоритмов множества A, связей между ними, которые как показывает опыт, для некоторых классов задач бывают довольно тесными.

Кроме того, создаваемая система обработки данных должна позволять решать целый ряд актуальных задач: создание максимума удобств потребителям, достижение разумного компромисса в определении возможностей этих средств, учет взаимосвязи методов решения задач одного класса, экономия высококвалифицированного труда программистов, достижение приемлемой эффективности обработки данных, представление возможностей для постоянного развития и совершенствования средств обработки данных, их комплексного применения с другими элементами математического обеспечения ЭВМ и др.

Учитывая интересы потребителей системы УСОД, возможные способы ее использования в вычислительном процессе, а также необходимость в постоянном ее развитии и совершенствованиии, разработчики предусмотрели следующие режимы работы:

I. Компоновка рабочей программы решения пакета задач и организация счета по ней.

2. Компоновка рабочей программы решения пакета задач и выдача ее на носители информации.

3. Совместное функционирование с другими системами обработки данных (режим сопряжения)

4. Изменение математического обеспечения системы.

5. Выдача системы и ее составных частей.

6. Обработка и использование "опыта" работы системы.

7. Организация отладка программных модулей.

Средства общения потребителей с системой УСОД можно разделить на два уровня. Первый уровень – это язык директив. В частности, в реализованном варианте системы УСОД задание описывается с помощью пяти самостоятельных предложений, расположенных в произвольной последовательности и поступающих в систему через телетайп:

- РАЗДЕЛ МО $b_i (i = 1,2,\ldots,r)$:
- СЧЁТ ПО РАБОЧЕЙ ПРОГРАММЕ или ВЫДАЧА РАБОЧЕЙ ПРОГРАММЫ;
- ДЛИНА ИСХОДНЫХ ДАННЫХ N;
- ТИП ОБРАБАТЫВАЕМЫХ ДАННЫХ $d_k (k = 1,2,\ldots,m)$:
- ЗАДАЧИ ОБРАБОТКИ $z_1, z_2, \ldots, z_n$.

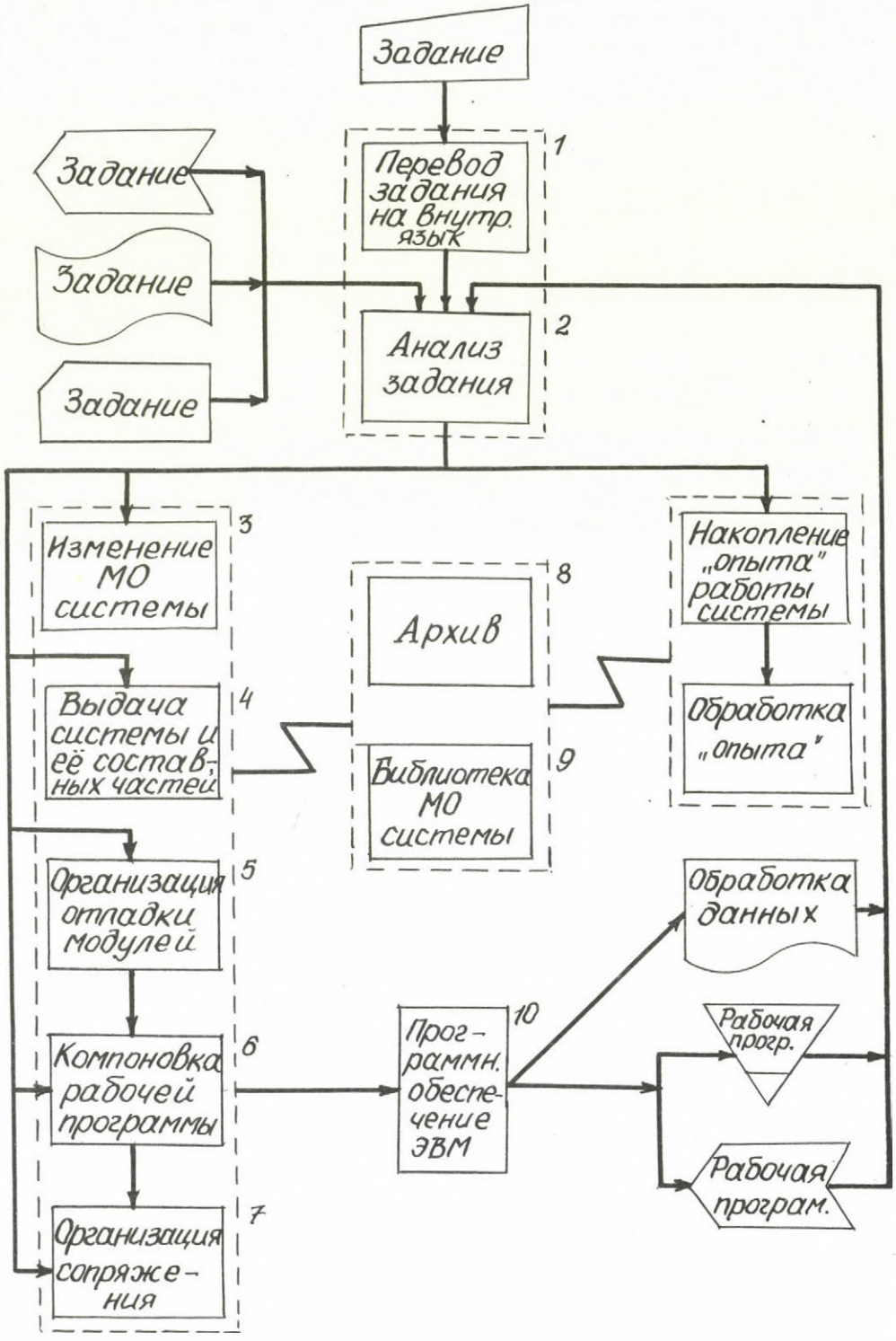

Задание
Задание
Задание
Задание
Перевод задания на внутр. язык
1
Анализ задания
2
Изменение МО системы
3
Выдача системы и её состав-ных частей
4
Организация отладки модулей
5
Компоновка рабочей программы
6
Организация сопряже-ния
7
Архив
8
Библиотека МО системы
9
Накопление „опыта" работы системы
Обработка „опыта"
Прог-раммн. обеспе-чение ЭВМ
10
Обработка данных
Рабочая прогр.
Рабочая програм.

Рис. 1.

Признаком конца сообщения служит знак "=".

Второй уровень средств общения потребителей с УСОД – собственно внутренний язык УСОД, представляющий собой последовательность информационных векторой, компонентами которых служит совокупность закодированных директив задания [3].

Принципиальная схема функционирования систем этого класса может быть представлена блок-схемой рисунка I. Такая система (в дальнейшем система S) состоит из подсистем трёх групп. Первая группа подсистем предназначена для организации взаимосвязи с абонентом, настройки системы на определенный режим работы для выполнения задания и обеспечения выполнения этого задания. Кроме того на подсистемы этой группы возлагаются функции организации оперативной функциональной взаимосвязи системы S с системами-абонентами. В эту группу, как правило, входят следующие подсистемы: подсистема анализа входных сообщений, синтеза вычислительных схем алгоритмов обработки данных и координации работы составных частей системы (блоки I, 2); подсистема компоновки программных модулей в рабочую программу и организации процесса обработки данных (блок 6); подсистема организации интерфейсов (блок 7).

На вторую группу подсистем возлагаются функции практической реализации идей самоорганизации системы и ее совершенствования. В эту группу входят: подсистема накопления опыта работы системы и его предварительной обработки (блок II); подсистема обработки накопленного опыта, выработки проекта совершенствования системы и практической реализации этого проекта (блок I2).

Третья группа подсистем представляет банк данных системы S Эту группу составляют следующие подсистемы: подсистема изменения математического обеспечения системы (блок 3); подсистемы выдачи составных частей системы (блок 4) (эта подсистема зачастую выступает в роли генератора систем определенной конфигурации и ориентированных на решение определенных классов задач); подсистема организации отладки программных модулей (блок 5) (включение этой подсистемы в состав системы S оказалось целесообразным, так как при этом имеется возможность одновременно опробовать программные модули в совместной работе

с имеющимся в системе программными модулями, а также значительно повысить эффективность отладки); информационная подсистема, которая из тактических соображений разделена на архив (блок 8) и библиотеку математического обеспечения системы (блок 9).

Все перечисленные подсистемы в разной степени используют элементы программного обеспечения ЭВМ, в связи с чем на схеме указан блок IO и показана его функциональная взаимосвязь с подсистемами системы S.

Этот блок играет важную роль в системе S, так как от заложенных в него возможностей существенно зависят закладываемые возможности в систему .

Функциональная взаимосвязь между выделенными подсистемами указана на схеме, в связи с чем подробно на этом останавливаться не будем.

Как уже отмечалось, для систем обработки данных такого класса проблемное математическое обеспечение представляется системой программных модулей. В связи с этим представляет интерес метод, с помощью которого для заданного класса задач обработки данных можно было бы построить оптимальную в смысле некоторого критерия оптимальности систему программных модулей.

Авторами данного доклада разработан Ψ -метод построения системы программных модулей для заданного класса допустимых задач обработки данных (конечного множества задач), которая является оптимальна в том смысле, что количество входящих в нее программных модулей минимально и минимально число операций в алгоритмах решения класса задач, построенных с помощью программных модулей данной системы.

Основная идея Ψ -метода состоит в следующем. Каждый алгоритм решения задачи представляет с помощью графа, вершинами которого служит совокупность функциональных операторов [4], составляющих алгоритм, а множество дуг указывает на информационные и управляющие связи между ними. В этих графах сначала осуществляется укрепнение вершин, путем заключения подграфов со свойствами α - и β -циклов, решающих узлов в одну вершину, а затем, используя операцию объединения графов,

строится общий (единый) граф, представляющий алгоритм решения всего класса задач. В этом объединенном графе укрупняются вершины, путем объединения линейных участков с определенными свойствами. В результате получим так называемый модульный граф, в вершинах которого заключены фрагменты алгоритмов, являющиеся программными модулями (т.е. фрагменты, удовлетворяющие свойствам упорядоченности и автономности). Модульный граф содержит вершины-модули трех групп: модули типов данных, операционные и эргативные модули. Модули типов данных предназначены для первичной обработки данных, которая заключается в преобразовании данных определенного типа (формата) в стандартный формат. Операционные модули осуществляют основную обработку данных, эргативные модули – окончательную обработку данных, которая заключается в выдаче полученных результатов в удобной для пользователя виде. Оказалось, что построенную таким образом систему программных модулей можно усовершенствовать, если использовать накопленный опыт работы системы S, в частности закон распределения вероятностей участия модулей системы в процессе решения задач. Процесс усовершенствования заключается в реализации преобразования, состоящего в объединении смежных вершин модульного графа с равными вероятностями в одну вершину. Усовершенствованная система программных модулей будет оптимальна в следующем смысле: как и прежде, число ее программных модулей минимально; математическое ожидание выполнимого числа операций в процессе решения задач минимально.

Изучив свойства, системы программных модулей, задаваемой модульным графом, удалось построить метод автоматического синтеза вычислительных схем обощенных алгоритмов совместного решения пакетов задач обработки данных (т.е. произвольных подмножеств функционально связных задач) и метод автоматической компоновки программных модулей в рабочую программу, реализующую обощенный алгоритм (этот алгоритм позволяет осуществлять также автоматическую сегментацию в память ЭВМ рабочих программ).

Системы изучаемого нами класса имеют довольно широкий диапазон своего применения. Особый интерес вызывает организация так называемых программных комплексов. В этом случае другие системы и проблемные программы рассматриваются как абоненты системы S.

У нас определенный опыт организации программных комплексов, т.е. организации оперативной функциональной взаимосвязи с модулирующими системами. Имеются в виду системы, в результате работы которых накапливаются статистические данные о некотором моделируемом объекте. С целью дальнейшей обработки и анализа этого статистического материала целесообразно использовать систему, предназначенную для этой цели. Нами организован программный комплекс СЛЭНГ-УСОД, являющийся функциональным сопряжением системы УСОД и системы СЛЭНГ [5], разработанной в Институте кибернетики и предназначенной для моделирования систем с дискретными событиями.

Основная идея организации сопряжения состоит в следующем. В моделирующую систему СЛЭНГ включен программный модуль автоматического перебора вариантов накопленной статистики, преобразование данных с исходного формата в стадартный и программной организации сообщения задания системе УСОД. В системе УСОД в свою очередь был открыт доступ к соответствующему формату данных и включен программный модуль загрузки обрабатываемых данных в рабочую область. Обработка данных всего набора осуществляется программным комплексом последовательно и автоматически. Набор задач для обработки каждого из вариантов данных указывается пользователем в зависимости от поставленных целей.

Вызывает определенный интерес вопрос исследования данного класса систем. В частности, важно уметь ответить на следующий вопрос: что влияет и каким образом на эффективность данной системы. Под эффективностью понимается затрата ресурсов ЭВМ на решение задач одного класса и скорость решения этих задач. Нами проведены исследования в этом направлении и получены некоторые интересные результаты. В частности, удалось показать, что экономия системой времени и памяти в процессе комплексного решения задач пропорциональна коэффициенту связности класса задач [4]. А такие показатели, как скорость построения алгоритмов обработки данных, окупаемость средств оптимизации программ обработки данных и др. прямо пропорциональны уровню агрегации программных модулей.

В заключение перечислим вкратце возможности системы УСОД.

Действующая версия системы УСОД позволяет решать задачи из четырёх классов. В первый класс включены задачи параметрической обра-

ботки данных и проверки **гипотез.** Туда входят такие задачи, как вычисление центральных и начальных моментов, вычисление медианы, моды, эксцесса и многих других параметров. Проверка гипотез на согласие эмпирического закона распределения с такими теоретическими законами, как биноминальный, нормальный, логарифмически-нормальный, Пуассона, Вейбула, Релея, равномерный и др. Для этой цели используются критерии Романского, Пирсона, Колмогорова-Смирнова и ряд других. Всего в этом классе $II0_8$ задач. Система программных модулей этого класса содержит $2I2_8$ программных модулей, в том числе 5 модулей типов данных, 75_8 операционных модулей и $II0_8$ эргативных модулей.

Второй класс составляют задачи парного корреляционно-регрессионного анализа. Среди них такие задачи, как, например, вычисление корреляционного отношения и его надёжности; вычисление линейного коэффициента корреляции и его надёжности, вычисление оценки математического ожидания и дисперсии факториального и результативного признаков, вычисление межгрупповой и внутригрупповой дисперсии и др. В этом классе содержится большое число задач определения отношений зависимости между факториальными и результативными признаками (уравнений регрессии). Среди уравнений регрессий укажем, например, следующие:

$$y = a_1 x + a_2, \quad y = a_1 \ln x + a_2, \quad y = \frac{1}{a_1 + a_2},$$

$$\ln y = a_1 x + a_2, \quad \ln y = a_1 \ln y + a_2,$$

$$y = a_1 x^2 + a_2 x + a_3, \quad y = a_1 \ln x^2 + a_2 \ln x + a_3,$$

$$\ln y = a_1 \ln x^2 + a_2 \ln x + a_3, \quad y = \frac{1}{a_1{}^2 + a_2 + a_3},$$

$$y = a_1 x^3 + a_3 x^2 + a_3 x + a_4.$$

В качестве критериев согласия используются индексы корреляции и детерминации, остаточная дисперсия, максимум отклонения, оценочные значения и др.

Всего во втором классе числится 104_8 задачи обработки статистических данных. В системе программных модулей этого класса содержится 235_8 программных модулей, в том числе 2 модуля типов данных, 127_8 операционных модулей и 104_8 эргативных модулей.

Третий класс содержит задачи множественного корреляционно-регрессионного анализа. Среди них такие, как вычисление множественных коэффициентов корреляции, корреляционное отношение, оценок математического ожидания и дисперсий, остаточной дисперсии, критерии Фишера, Стьюдента и др. С помощью задач этого класса можно установить функциональную связь между 20 факториальными признаками и одним результативным. При этом форма связи может быть выбрана в виде (I) или (*и*) в виде (2).

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots + a_m x_m, \qquad (I)$$

$$\begin{aligned} y = {} & a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots + a_m x_m + \\ & + a_{11} x_1^2 + a_{22} x_2^2 + a_{33} x_3^2 + \ldots + a_{mm} x_m^2 + \\ & + a_{12} x_1 x_2 + a_{13} x_1 x_3 + \ldots + a_{1m} x_1 x_m + \\ & + a_{23} x_2 x_3 + a_{24} x_2 x_4 + \ldots + a_{2m} x_2 x_m + \\ & \ldots + a_{m-1} a_m x_{m-1} x_m . \end{aligned} \qquad (2)$$

При чем, если форма связи задана полиномом (I), при имеется возможность оперативно с помощью клавиатуры ЭВМ задавать различные трансформанты его членов. Например, каждый из членов полинома (I) может быть трансформирован по одной из следующих функций $\left\{x_i, x_i^2, \ln x_i, \frac{1}{x_i}, e^{x_i}\right\}$. Если форма связи задана полиномом (2), то с помощью клавиатуры ЭВМ можно указать произвольный набор его членов, воздействие которых не существенно.

Всего в третьем классе содержится 33_8 наименования задач. Система программных модулей этого класса содержит 35_8 программных модулей в том числе: I модуль типов данных, 20_8 операционных модулей и 16_8 эргативных модулей.

В четвертый класс включены задачи спектрально-корреляционного анализа случайных процессов. Среди них: вычисление ненормированной автокорреляционной функции, вычисление нормированной автокорреляционной функции, вычисление нормированной и ненормированной спектральной плотности, вычисление интервала корреляции, вычисление параметров авторегрессии, вычисление взаимной ковариационной функции, вычисление взаимной корреляционной функции и многие другие. Всего в этом классе 36_8 задач. Система программных модулей этого класса содержит 150_8 программных модулей, в том числе 10_8 модулей типов данных, 77 операционных модулей и 41 эргативных модулей.

Литература

[1] А.А. Михайлшин, И.Н. Парасюк, И.В. Сергиенко, Н.И. Тукалевская: Автоматизированная система обработки статистических данных, Изд-во Института кибернетики АН УССР, ч, I,П,К., 1970.

[2] И.В. Сергиенко, Н.И. Тукалевская, И.Н. Парасюк: Универсальноспециализированная автоматизированная система обработки данных на ЭВМ, Изд-во Института кибернетики АН УССР, ч.I, П, К., 1973.

[3] И.Н. Парасюк, И.В. Сергиенко, Н.И. Тукалевская: Универсльно-специализированная автоматизированная система обработки данных на ЭВМ (система УСОД), ж. "УСиМ", №2, Изд-во "Наукова думка", К., 1974.

[4] И.Н. Парасюк, И.В. Сергиенко: Некоторые вопросы разработки и исследования одного класса универсально-специализированных автоматизированных систем обработки данных, ж. "Кибернетика", №6, Изд-во "Наукова думка", К., 1973.

[5] В.М. Глушков, Л.А. Калиниченко, Т.П. Марьянович, В.М. Москаленко, М.А. Сахнюк, СЛЭНГ - система программирования для моделирования дискретных систем, Изд-во Института кибернетики АН УССР, К., 1969.

Összefoglaló

Automatikus adatfeldolgozó rendszerek kialakitásának tapasztalatai
I.V. Szergienko, I.N. Paraszjuk

A dolgozat az Ukrán Tudományos Akadémia Kibernetikai Intézetében M-220 tipusu számitógépre kidolgozott statisztikai programcsomag leirásával foglalkozik. A programcsomag hipotézisek vizsgálatával, korrelációs, regressziós analizissel és spektrálanalizissel kapcsolatos számitásokat végző modulokból, és az ezeket koordináló vezérlőprogramból áll. Automatikusan rögziti, feldolgozza és hasznositja a müködése során szerzett tapasztalatokat.

A szerzők ismertetik a programcsomag tervezésénél használt algoritmust is, amely lehetővé teszi a programmodulok és az ezeket alkotó müveletek számának minimalizálását. Röviden szó esik a rendszer müködése során szerzett tapasztalatokról is.

Summary

On the experiences of automatic data processing systems
I.V. Sergienko, I.N. Parasuk

A statistical program package developed in the Institute of Cybernetics, Ukranian Academy of Sciences for M-220 computer is dealt with. The package consists of programs, dealing with solution of problems connected with testing hypothesis, correlation, regression analysis, spectral analysis and a main program controlling them. The experience obtained during the work of the package is fixed, processed and used automatically.

The authors give the algorithm, used for the planning of the package and making possible to minimize of the necessary programs and operations included into them. In the paper there is a short description of the authors' experiences obtained by their practical work with the package.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

REAL TIME OPERATING SYSTEMS PROBABILISTIC MODELS

Adam WOLISZ

Department of Complex Automation Systems of the Polish Academy of Sciences

I. INTRODUCTION

The majority of lectures presented during this winter school dealed with mathematical problems of big computers operating systems (O.S.) for data processing or scientifical computations.

This paper aimes to discuss some problems connected with control of technological plants, one of the most rapidly growing areas for computers applications. The following considerations have been written rather from the position of a control system designer, then the computer manufacturer.

Basing on such a biased point of view some classification of real – time O.S. will be suggested and their mathematical models will be discussed.
Software used in process control systems can be divided into three parts:

- basic software, including computer oriented O.S. We shall include here programs for system generation, drivers for various I/O devices, loaders, compilers, memory allocation routines as well as interrupt handlers. All those are usually provided by a computer manufacturer, together with apropriate hardware.
  Often the computer oriented operating systems enable various modes of operation, such for example as batch processing for background activities or time sharing features.
- Utility programs, prepared for data acquisition, direct digital control optimization and so on. They are usually based on some mathematical models of technological processes, and are different for various plants, however they can be created using some languages or programme packages for process control. Those programes are usually prepared by a specialized designe office or software manufacturer in close cooperation with the user.
- Plant oriented O.S. enabling in an arbitrary time epoch to recognize the situation in the computer and in the controlled plant, so as to choose an proper program to be executed at this time.

Situation of the computing system is understood as a set of signals denoting demands for execution of various programmes, which are generated either by the system itself or more frequently by the environment. The plant oriented O.S. determines the service disciplines imposed in the system utilizing all features provided by the hardware and basic software as interrupts (including their enabling and deaseabling mechanism) real time clock etc. Implementation of the service discipline as well as switching the control from one programme

to another is usually connected with some overhead caused mainly by the basic software utilization.

Implementation of any utility programme is possible using various plant oriented O.S. and various hardware together with it is basic software. The decision is usually made by the system designer.

In the following considerations we shall restrict ourselves only to the plant oriented O.S.

II. Requirements for process control O.S.

The examination of different O.S. is usually done with respect to following requirements:

a) The possibility of obtaining proper response times, which is an essential one for control applications,

b) Flexibility, understood as a set of features enabling systems developement in the sense of introducing sequentially additional functions as for example data logging, control of some technological processes, inventory control, and optimization of the full instalation. Software developement should be possible without enforcing any changes in previously debugged and well running parts.

c) Reliability and foult tolerancy. One must be prepared for existance of some bugs in new introduced programmes. System's structure should minimize their influance on the other parts of software. There can also occure some hardware demages and in this case one of the following actions should be started:

   – system reconfiguration and further work in the full range of functions (in the most developed systems).

   – System reconfiguration and fulfilling only the most important functions.

   – Fixing all outputs in some predetermined "secure" positions and switching over to a manual control.

In any case some diagnostic tests should be executed and their results edited.
It is rather difficult to express properly requirements b) and c) in a formal way however such attempts are done mainly on the basis of graph theory and reliability theory. Some notices in this area can be also done for various structures basing on programmers and systems analysts experience.
As far as response times and utilization factor of computing systems are concerned the queuing theory approach was very usefull.

III. Real time O.S. queuing models – general remarks

An overview of deterministic congestion models was presented in paper [1] so now we shall constrain ourselves only to probabilistic ones.
An essential problem for any queuing problem is determining the type of sources that means theirs dimension N defined as the number of demands which can be generated without any service being completed and their interarrival time distribution. Investigation of various queuing disciplines for the general case of N dimensional sources is fairly difficult, and as the matter of fact not often met in practical cases. Therefore the boundary case, an infinite dimensional source which leads to an open queuing model is usually considered. In the control applications quite an important seem to be one–dimentional sources fitting perfectly to working conditions of some sensors (no technogical parameter's limit value can be twice violated without any control action being fulfilled in the meantime, no piece of material can disappear until it is taken away . . .). Both those cases give some upper and lower limit for solutions of the general N–dimensional case.
The interarrival time distributions are usually assumed to be exponential ones, because of their very convenient Markov property. The basic question if it is a good approximation for processes occuring in computer systems was investigated in [2].
The author proved positive the statistical hypothesis of interarrival times being exponentialy distributed using experimental data as well as sofisticated statistical tests, and also demonstrated that results obtained from a queuing model under such an assumption are very close to reality. The vast majority of models can be described mathematically for the case of arbitrary service distribution and as such there are generally valid.
What we usually ask for is the mean responce time. However for control applications the variance and the probability of exceeding some predetermined value can be also of great importance. Therefore the distribution of response time or more frequently it's Laplace–Stieltjes transform is exactly what we need. It is quite similar problem with the utilization factor for a computer system.

IV. Real time O.S. classification

A detailed description of O.S. structures in terms of their service disciplines will be possible only after some closer investigation of request types [3].
The demands for service can be devided into three main groups as presented on fig. 1.
The internal requests include parity checking, power failure, CPU real overflow etc. as well as requests for communication with auxiliary memory originating in the actually executed programme.
The real time clock requests have a well known interpretation. The external requests can be devided into two types.
First type includes requests which service consists only of memorizing or editing some pieces of data connected with a predetermined location.

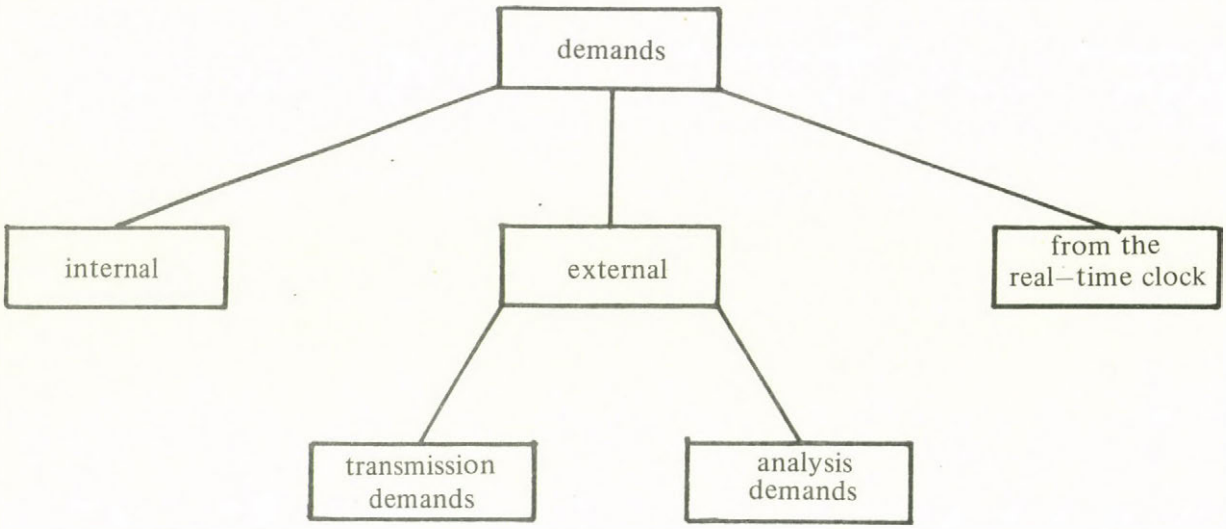


Fig. 1. Classification of demands in real time O.S.

Usually after completing of such a transmission a return to previously executed programme is desired (if no another request of higher priority occured meanwhile). These requests having a short service time shall be referred to as transmission requests.
The second type includes requests which service is not connected with data transmission but represents merely signals for activisation of a proper programme. These requests shall be referred to as analysis requests.

Due to their nature demands from two former groups have usually a preemptive priority over external calls which is imposed utilizing the interrupt feature while with regard to the third group of requests service discipline three types of O.S. can be distinguished:

1) *Event – oriented O.S.* which are nowadays the most frequently used ones. The priority of any request is evaluated immidiately after it's generation (it means when the appropriate event happens) and the highest priority request is chosen for service.
Scheduling algorithms applied in control systems are usually based on the fixed priority approach in contradiction to big general purpose computers, where dynamical priorities or at least periodical reevaluation of fixed priorities are often to be spotted.
The first approximate description of an event–oriented system gives a well known congestion model with several infinite – dimensional sources served under the preemptive – resume discipline (e.g. [4]).
This model is pretty far from reality because three main factors are anticipated namely:
   a) existance of nonpreemptive parts in the executed programmes
   b) overhead connected with preemption
   c) finite type of sources.

Taking into account the first of this factors leads to a multiphase service systems with preemptive and nonpreemptive phases successively [5].
Some attempts to consider the second factor were done also for the case of preemptive overhead [5] [6].
However in practical cases we should rather assume nonpreemptive structure of the overhead time. Some research in this area aiming at obtaining a full description of a multiphase system in which every service consists of some set of preemptive and nonpreemptive parts with nonpreemptive overhead added to every service interruption as well as resuming of the preempted programme execution was done in the Department of Complex Automation Systems, Polish Academy of Sciences, and the results would be published in the nearest future. Also an interesting simulational research reported in [7] should be mentioned here.
As far as the finite type of sources is concerned one classical system, the repairman problem has been deeply investigated. However it seems that a much more interesting case is a set of one–dimensional sources, with various parameters of the generation rate nad various service – time distributions. Such a scheme which fits to the presented also by Tomko [8] multiprogramming computer system with a constant number of non–homogeneous jobs having independent I/O facilities has been investigated in [9], [10] for the preemptive resume, head–of–the–line and discretionary priority disciplines. (Such a discipline which is preemptive before any tagget demand complets an amount of service equal to some predetermined value, and than it becomes the head of the line discipline until the service is completed is called discretionary discipline. It is a special case of a multiphase system.) The utilization factor of the system as well as the probability density function of waiting time, response time an occupation time has been found (an analysis for the preemptive – resume discipline is given in the Appendix).
All this models are studied either by the imbedded Markov chains analysis or using the supplementary variable technique. It seems to the author that the second one combined with Gaver's concept of completion time is more hopefull fo various priority disciplines. As it was shown in [4] using this method one may obtain the full description of busy period process and afterwords due to the renewal theory the general process description can be found. In that way the transient state of the system can be also investigated. Some more research in this field connected mainly with overhead time consideration especialy in the finite – dimension sources models in needed.

2) The Time – oriented O.S. which characteristic feature is an analysing the situation in the controlled process with regard to all external demands in arbitrary chosen by the scheduling algorithm time epochs independently of the moment of their generation (without making use of the interrupt feature). Among the scheduling algorithms applied in such a system one can distinguish four groups which will be discussed here.

a) Synchronious O.S.

In this solution the utility software is divided into several segments being executed in a cyclic manner. The sequence of their execution as well as time quantum for every

segment is predetermined by the system designer, while synchronization is assured by the clock. When the predetermined time quantum elapses all the vital information is stored, and control is passed to a next segment.
Time schedule for such an O.S. is plotted on Fig. 2.

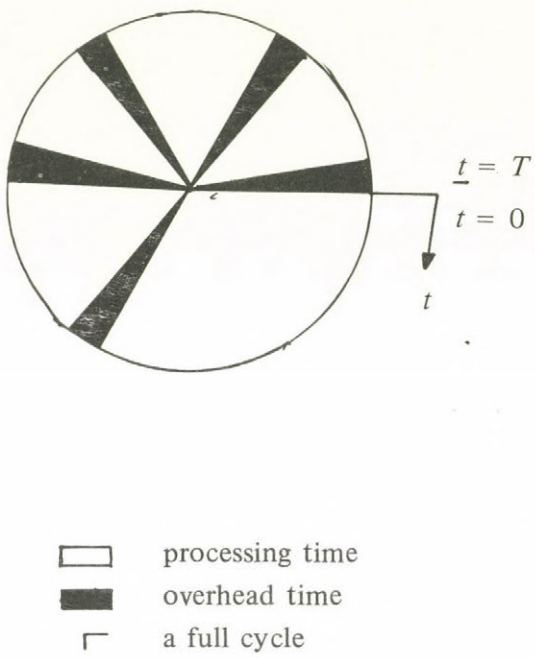
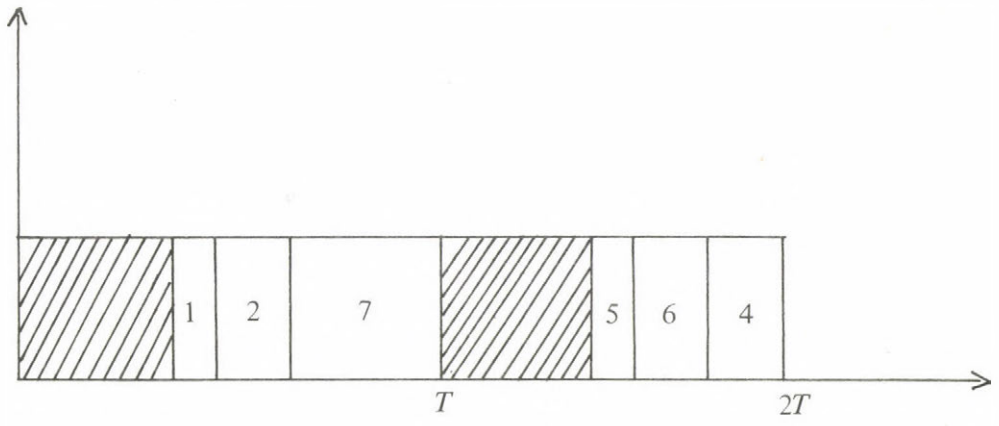


Fig. 2. Time schedule for an synchronious O.S.

An ettempt to present some analitical results for this system as well as evaluation of it's features is given in [11].

b) O.S. with periodical cycle

This scheduling algorithm can be considered as a special case of the previous one, but it should be discussed due to it is wide range of applications. As it is presented in the Fig. 3. every T seconds a fixed programme package is executed, which consists of some highly time dependent jobs (as for example data logging, direct digital control etc.) and the rest of time is devoted to non periodical jobs served under any introduced discipline.

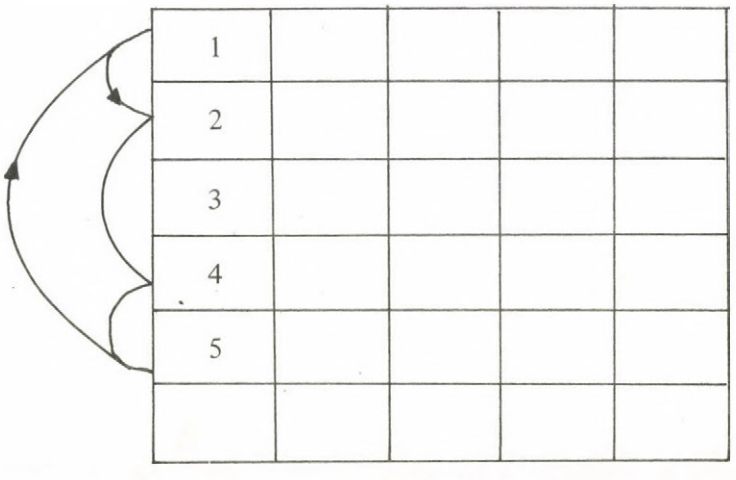


▨ periodical jobs package
1, 2, . . . numbers of nonperiodical jobs

Fig. 3. O.S. with periodical cycle

c) The sequential O.S.

In such a structure (which is often used in chemical industry) job table is created which contains full description of all jobs, (Fig. 4.).



1, 2, . . . , job numbers

Fig. 4. Searching the job table in a sequential O.S.

We shall mention here only the most important components of a job description such as an enable / disable flag (which informs if the job is to be considered et all), time of the nearest desired activisation, period of activisation, starting adress location, continuation adress (for a preempted job).
This system also works with a predetermined cycle. Every given time Quantum it starts searching the jobs' table from the very beginning with respect to enabled jobs (so the position in the table enforces priority within a cycle).
If any of the jobs has a desired activisation time smaller then system clock state (it means that a demand is in Queue) it is executed, and it is activisation period is added to the desired activisation time.
After the time of the cycle elapses a jump to the top of the table takes place regardless of the actually being executed job's number and state. This structure has so far no elaborated queuing theory description.

d) O.S. with time slicing
The most commonly used time – sliced O.S. for real time applications are time – sharing models having a very large bibliography. For the industrial control applications a modification of them is sometimes used, namely a priority system in which service is given in quanta, and after every quantum completion a job having the highest priority is chosen for further service (sometimes it can be continuation of the previous job's service). Of course the choise of a proper Quantum is essential for this case. This case is under author's investigation.

3) Mixed structure
There exists also a class of systems which serve the transmission demands using interrupt system directly after their generation while analysis demands are considered in time epochs chosen by the scheduling algorithm, regardless of the moment of demands generation.
Scheduling algorithms for analysis demands are the same as in the case of time oriented O.S.

V. Final Remarks

Not attempting to give any exact comparison of the discussed structures one can present some outlines of their main features.
Usually the shortest response time can be obtained (for high priority demands) in the event oriented O.S.
However in this case the variance of response time can be expected to be bigger than in some time – oriented O.S. which introduce a deterministic factor in to their operation as well.
In the event oriented O.S. the ratio of overhead time changes with the systems'load and increases distinctively in heavy traffic conditions while for example the time–oriented synchronous system spends on the overhead a certain constant amount of time.

As far as the requirements (b) and (c) of the paragraph II are concerned it is often pointed out by various authors [12], [13] that debugging of an event – oriented system as well as it is later development is very time – consuming and enlarging it's reliability is quite difficult.

All it leads to a conclusion that the ratio of time oriented O.S. or mixed type O.S. will probably increase in the nearest future as far as the process control is concerned.

One can also state that mathematical description for plenty service disciplines is either not as detailed as required or does not exist at all so the choise of a structure is often done by the system designer basing only on his experience. Therefore some more research in this area seems to be necessary.

It should be mentioned that all results presented in this paper were obtained in cooperation with Mgr inz.Tadeusz Czachorski to whom I am also very gratefull for his valuable remarks concerning it's shape.

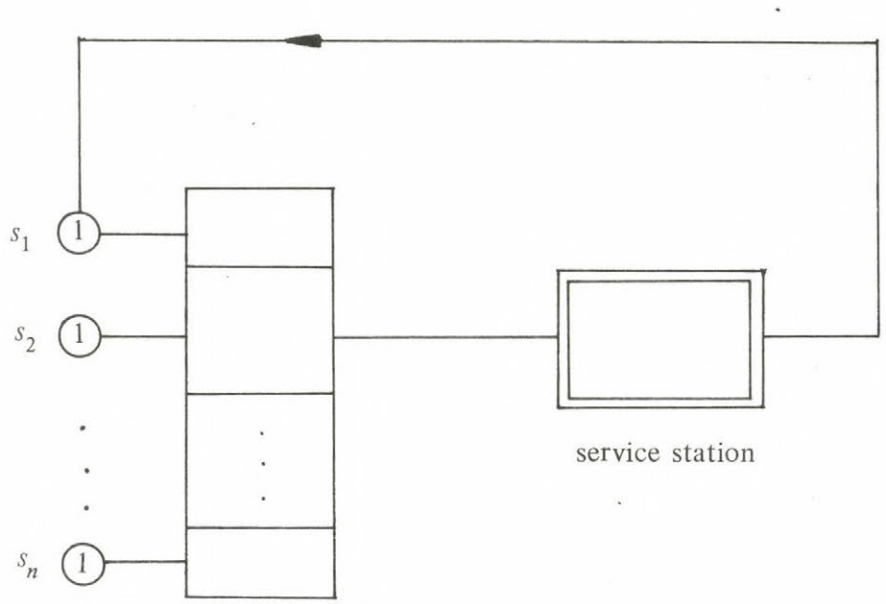
Appendix

We shall discuss here as example a system of one dimensional sources served under a preemptive – resume discipline (Fig. A. 1.). The investigation will be done using Jaiswall's method. For each source we shall assume the time a demand spends within the source to have negative exponential distribution.

(1) $A_i(x) = P_r\{\tau \leqslant x\} = 1 - e^{-\lambda_i x}$ $\quad i = 1, 2, \ldots, n$

and the service time duration probability density function

(2) $S_i(x) = P_r\{x < t < x + dx\}$ $\quad i = 1, 2, \ldots, n$



A set of one – dimension sources

Fig. A. 1.

We shall assume that a demand from a source s_i has priority over demand from a source s_j if and only if $i<j$ $i,j=1,2,\ldots,n$.
The considered sheme describes also a multiprogramming system, with a constant number of nonhomogeneous jobs having independent I/O facilities, discussed by Tomko [8] in the case of preemptive resume service discipline.

Let us first investigate the operation of a service station with a single source Fig. A. 2.

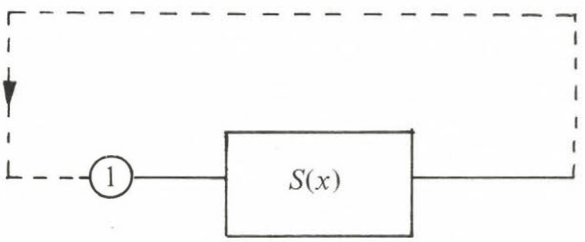


Fig. A. 2.

1) *A Simple scheme*
Let $p(x,t)$ be the density of probability that at time t the busy period (which started at time $t=0$ due to demands generation) is in progress and the elapsed service of the unit is between x and $x+dx$.
$b(t)$ – be the probability density that the busy period (that means a time interval during which the server remains serviving demands without any pouse) which started at $t=0$ terminates between t and $t+dt$.
One can easy verify, that $p(x,t)$ satisfies the equation

$$p(x+\Delta,t+\Delta)-p(x,t)[1-\eta(x)\Delta]=0 \tag{3}$$

where

$$\eta(x)=\frac{S(x)}{1-\int_0^x S(u)du}$$

tending with $\Delta\to 0$ we obtain an difference – differential equation

$$\frac{\partial\rho(x,t)}{\partial x}+\frac{\partial\rho(x,t)}{\partial t}=-\rho(x,t)\cdot\eta(x) \tag{4}$$

subject to initial condition

$$\rho(x,0)=\delta(x)$$
$$\rho(0,t)=0 \qquad \text{for} \qquad t\neq 0$$

(4) yields

$$\overline{\rho}(x,s) = e^{-sx}\, e^{-\int_0^x \eta(x)dx} \quad *$$

(5)

$$b(t) = S(t)$$

(6)

2) *A scheme with initial period process*

Let us assume, that at $t = 0$ an initial period having density function of its duration $\Omega(t)$ starts. During this period a demand can be generated, but it is service may start only after completion of the initial period, and in that case the busy period b^{Ω} is a convolution of a busy period $b(t)$ (as in the previous case) and initial period.

If no demand is generated during the initial period $b^{\Omega}(t)$ is equal to $\Omega(t)$.
It can be easily proved that

$$\overline{b}^{\Omega}(s) = \overline{\Omega}(s) \cdot \overline{S}(s) + \overline{\Omega}(\lambda + s)[1 - \overline{S}(s)] \tag{7}$$

$$\overline{p}^{\Omega}(x,s) = [\overline{\Omega}(s) - \overline{\Omega}(s+\lambda)]e^{-sx - \int_0^x \eta(u)du} \tag{8}$$

where b^{Ω}, p^{Ω} stand in this scheme for similar values as in the previous paragraph.

Similar to infinite dimensional source models we shall introduce the notion of completion time, defined as the duration of period that begins from the instant the service of a demand starts and ends at the instant the server becomes free to take the next unit generated from the same source.
For preemptive– resume disciplines the completion time is equal to the period between starting and finishing the service of any demand (inluding time of premptions).
The probability density function of completion time duration for demands from s_j (called j – type demands) can be found by following inference:
The probability of n preemptions during the service of i–type demands p_n is

$$p_n = \frac{(\Lambda_{j-1} \cdot y)}{n!}\, e^{-\Lambda_{j-1} \cdot y} \tag{9}$$

* We shall denote the Laplace – transform of any function $\varphi(t)$ by $\overline{\varphi}(s)$

$$\overline{\varphi}(s) = \int_0^{\infty} e^{-st}\varphi(t)dt$$

where

$$\Lambda_{j-1} = \sum_{i=1}^{j-1} \lambda_i$$

y – the lenght of i – type service.

Any preemption has the same duration, with probability density function $\gamma_{i-1}(t)$ which represents the lenght of busy period for a system with $(i-1)$ sources.
(As existance of lower priority sources does not influance the service of high – priority ones)

If so, than

$$C_j(x) = \int_0^x \sum_{n=0}^{\infty} e^{-\Lambda_{j-1} y} \frac{(\Lambda_{j-1} y)^n}{n!} \gamma_{j-1}^{*n}(x-y) \cdot S_j(y)dy \tag{10}$$

where γ_{j-1}^{*n} denotes a convolution of n times $\gamma_{j-1}(t)$
(10) yields

$$\overline{C}_j(s) = \overline{S}_j\{s+\Lambda_{j-1}[1-\overline{\gamma}_{j-1}(s)]\} \tag{11}$$

the expected value and second moment are

$$E(c_j) = E(S_j)[1 + \Lambda_{j-1} E(\gamma_{j-1})]$$
$$E[(c_j)^2] = E(S_j^2)[1 + \Lambda_{j-1} E(\gamma_{j-1})]^2 + \Lambda_{j-1} E(S_j) E(\gamma_{j-1}^2)$$

$\gamma_j(t)$ can be obtained as

$$\gamma_j(t) = \frac{\lambda_j}{\Lambda_j} B_j(t) + \frac{\Lambda_{j-1}}{\Lambda_j} B_j^{\gamma_{j-1}}(t) \tag{12}$$

(where $\overline{B}_j$ and $B_j^{\gamma_{j-1}}$ are given by (6), (8) with substitutions

$S(t) \longrightarrow c_j^{(t)}$
$\Omega \longrightarrow \gamma_{j-1}$
$\lambda \longrightarrow \lambda_j$

after the following inference:
The busy period γ_j may be started either by j-th type demand (with probability $\frac{\lambda_j}{\Lambda_j}$) or by any demand of the higher priority type (with probability
The substitution of completion time instead of service time originates from preemption which can occure during service. (12) yields after transformation and proper substitutions

$$E(\gamma_j) = \frac{\lambda_j}{\Lambda_j} E(c_j) + \frac{\Lambda_{j-1}}{\Lambda_j} \{E(\gamma_{j-1}) + E(c_j)[1 - \overline{\gamma}_{j-1}(\lambda_j)]\} \tag{13}$$

Busy periods starting epochs can be treated as regeneration points of a renewal process. The time duration between two regeneration points $f(t)$ is given by a convolution

$$f(s) = \gamma_j(t) * \Lambda_j e^{-\Lambda_j t} \tag{14}$$

Assuming that at $t = 0$ a busy period starts, transform of the renewal density $h_j(s)$ can be expressed as

$$\overline{h}(s) = \frac{\overline{f}(s)}{1 - \overline{f}(s)} = \frac{\Lambda_j \cdot \overline{\gamma}_j(s)}{s + \Lambda_j[1 - \overline{\gamma}_j(s)]} \tag{15}$$

Let $e(t)$ be the probability of the system being idle at time t of the general process, consisting of busy and idle periods.

$$e_j(t) = \gamma_j(t) e^{-\Lambda_j t} + h(t) * \gamma_j(t) * e^{-\Lambda_j t} \tag{16}$$

$[1 - e(t)]$ is exactly the utility factor in any given time t For $t \to \infty$ the stationary state probability of the server being idle is

$$e_j = \lim_{t \to \infty} e_j(t) = \frac{1}{1 + \Lambda_j E(\gamma_j)} \qquad (\text{if } E(\gamma_j) < \infty) \tag{17}$$

We shall now calculate the distribution of time duration between j–th type demands generation and the epoch when it's service starts (waiting time) denoted as $v(t)$, in the stationary state.

The probability of type j demand taking part in the busy period γ_j is

$$P_{j,\gamma_j} = \frac{\lambda_j}{\Lambda_j} + \frac{\Lambda_{j-1}}{\Lambda_j}[1 - \int_0^\infty e^{-\lambda_j x} \gamma_{j-1}(x)dx] = \frac{\lambda_j}{\Lambda_j} + \frac{\Lambda_{j-1}}{\Lambda_j} \cdot [1 - \overline{\gamma}_{j-1}(\lambda_j)] \tag{18}$$

as j–type demand can be served at most once during γ_j.

The conditional probability of busy period beginning with γ_{j-1} under the condition that type j demand takes part in the busy period is

$$P_{c,j} = \frac{\frac{\Lambda_{j-1}}{\Lambda_j}}{\frac{\lambda_j}{\Lambda_j} + \frac{\Lambda_{j-1}}{\Lambda_j}[1 - \overline{\gamma}_{j-1}(\lambda_j)]} = \frac{\Lambda_{j-1}}{\lambda_{j-1} + \Lambda_{j-1}[1 - \gamma_{j-1}(\lambda_j)]} \tag{19}$$

and the conditional probability of busy period starting with j–type demands service under the condition that j–th type demands take part in the busy period

$$P_{N,j} = \frac{\lambda_j}{\lambda_j + \Lambda_{j-1}[1 - \overline{\gamma}_{j-1}(\lambda_j)]} \tag{20}$$

Waiting time is equal to 0 if a busy period starts j–type demand service or equal to τ with probability density function $u(\tau)$ if a busy period starts with γ_{j-1} (of course we tell only about such a busy periods in which j–type demand takes part).
Aiming to find an formula for $u(\tau)$ one has to realize, that busy period is duration x for $(j-1)$ types demands has a density function $\gamma_{j-1}(x)$ while a time period y between its start and generation of j–type demand has a density function

$$\lambda_j e^{-\lambda_j \cdot y}$$

Yet a stochastic variable $\tau = x-y$ has the density function [14]

$$u_j(\tau) = \int_{-\infty}^{\infty} \gamma_{j-1}(x-y)\, 1(x-y) \cdot \lambda_j e^{\lambda_j y}\, 1(-y)dy \tag{21}$$

where

$$1(x) = \begin{cases} 1 & \text{for } x \geqslant 0 \\ 0 & \text{for } x < 0 \end{cases}$$

(21) yields

$$\overline{u}_{j(s)} = \lambda_j \frac{\overline{\gamma}_{j-1}(s) - \gamma_{j-1}(\lambda_j)}{\lambda_j - s}$$

Finaly

$$\overline{v}(s) = P_{c,j} \cdot \overline{u}_j(s) + P_{N,j} \cdot 1 \tag{22}$$

which after proper substitutions yields

$$E(v) = \frac{\Lambda_{j-1}}{\lambda_j + \Lambda_{j-1}[1 - \overline{\gamma}_{j-1}(\lambda_j)]} \cdot \frac{1}{\lambda_j} [\lambda_j E(\gamma_{j-1}) + \overline{\gamma}_{j-1}(\lambda_j) - 1]$$

The probability density $R(\eta)$ of the response time being equal to η can be easily calculated as

$$R_j(\eta) = v_j(\tau) * c_j(\tau)$$

One can also find occupation time of the server which will be not further discussed here.

References

[1] I. Labetoulle "Computer scheduling problems" preprints of this winter school.

[2] De Cegama "A methodology for computer model building " Fall Joint computer Conference, (1972), pp. 299–301.

[3] A. Wolisz, T. Czachórski, "On the reduction of interrupts in the synthesis of real–time operating systems" Podstawy sterowania vol. 4 no. 3 pp. 289–304 (in Polish).

[4] N. K. Jaiswal "Prioririty Oueues" Academic Press, New York, (1968).

[5] Б.В. Гнеденко и другие ''Приоритетные системы обслуживания'' Издателбство Московского университета, Москва, 1973.

[6] Linas Schrage, "A mixed priority queue with applications tothe analysis of real–time systems" Oper. Res. vol. 17 no. 4 pp. 728-742.

[7] А.И. Никитин, СА Шестяков ''Влияние диспетьерского времени на харакмеристики операционной системы с приори тетной дисуиплиной бслуживания'' Управляющие системы и Машины, No. 1 Vol. 1 (1972).

[8] Tomkó József "Tömegkiszolgálás elméleti modellek a számitógépek rendszerek a matematika leirásába" preprints of this winter school.

[9] A. Wolisz, T. Czachórski, "Priority Service Systems for a one dimension sources set" Podstawy sterowania vol.3 no. 4 (1973), pp. 291–32) (in Polish).

[10] T. Czachórski, A. Wolisz "The discretionary discipline of one dimensional sources service" Podstawy sterowania, vol. 4 no. 2 (1974), pp. 115–130 (in Polish).

[11] J.Golubowicz"The system of time and memory allocation for demands service" Podstawy sterowania, vol. 4 no. 4 (1974), (in Polish).

[12] P. Burnet, PA. Kidd, A. Lister, "Simulation of real–time programme faults" The Computer Journal, vol. 17, no. 1 (1973).

[13] Hagnes Cookbook "Please don't interrupt me while I am computing" Computer, (december 1973).

[14] A. Papoulis, "Probabilisty, Random Varriables and stochastic Processes" Mc Graw Hill, (1965).

Összefoglaló

Real – time operációs rendszerek valószinüségszámitási modelljei

A. Wolish

A dolgozat ismerteti az operációs rendszerekben lezajló folyamatokat és az ezekre alkalmazható sorbanállási modelleket. A real – time operációs rendszerekre egy osztályozást ad a kiszolgálás módjától függően. Befejezésül egy egyszerü preemptiv kiszolgálási stratégia analitikus vizsgálatát adja.

Р е з ю м е

Вероятностные модели "real-time" операционных систем

А. Волиш

В работе излагаются процессы происходящие в операционных системах и применяемые к ним модели теории очередей. Дается классификация "real-time" операционных систем по режиму обслуживания. Наконец аналитически исследуется стратегия с автоматным приоритетом.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

SCHEDULING ALGORITHMS IN CONTROL DATA'S NETWORK OPERATING SYSTEM (NOS)

K. Tantscher

In a modern operating system the two objectives of best utilization of computer resources and reasonable service for time sharing users (response time!), which are to a certain extent exclusive, create the necessity for effective scheduling algorithms.

All really implemented algorithms have to live in a certain environment – the given hardware. Since the hardware architecture of the CDC CYBER 170 Series is rather unique, we shall start with a brief description thereof.

Fig. 1 shows that a CYBER 170 System consists of a Central Processing Unit (CPU), a Central Memory (CM), at least 10 Peripheral Processors (PPU), at least 12 Data Channels and some peripheral equipment. The Extended Core Storage is optional and treated by the Operating System like any rotating mass storage.

The uniqueness of the system starts with PPU's which are independet processors with their own memory (4K 12bit bytes) and access to CM.

Their main purpose is to relieve the CPU from I/0 – work and operating system functions.

This leads to a system lay–out as shown in Fig.2.

Two PPU's are dedicated to the operating system: PPØ contains the MONITOR at all times. This is the routine which keeps track of all time dependent and periodic functions of the system. PP1 is dedicated to drive the CRT–Console,thus giving the operator the various system–status information in clear text.

The remaining 8 PP's form a pool and can be assigned on a demand basis.

According to Fig.3 the CPU is time shared over all active jobs in CM.

Figures 4, 5, 6 and 7 illustrate the Control Point Concept. Central Memory can be assigned in any size (no partitioning or paging!). For convenience in addressing the system assigns memory in increments of 100 words. (1 word = 60 bit, max. memory size = 262K words).

Each Control Point has an area of 200 words in low core which contains status and accounting information of the job presently running at this Control Point. The "Exchange Jump" instruction which is used to switch the CPU from one to another job saves the register contents in the first 16 words of this "Control Point Area" in low core. In case all Control Points are occupied the system has the possibility to swap the field length of a job together with all his information contained in low core to disk in order to free space for a higher priority job.

Figures 8 – 11 show the external effects of the scheduling system on the example of one job.

As jobs are read in (e.g. from a card reader), they are placed in the INPUT–Queue on disk with an initial or entry priority. An aging routine increases this priority according to a given aging rate. After some time, the priority will be higher than the priority of presently running jobs and the job will be scheduled for execution, that means he will be assigned to a Control Point and loaded into Central Memory. There he will stay with the upper bound priority of the INPUT–Queue until his allowed CM–time slice or CPU–time slice has elapsed. At this point in time his priority will be lowered to the lower bound priority of the INPUT– Oueue. This makes him a candidate for being swapped out. When the scheduling program decides that there are other, higher priority jobs to run, it will initiate the swap out of this job. Now he enters the ROLLOUT–Queue (only the first time!). The aging begins again until the schedule decides to swap him in again and so forth.

The reason to give a job the lower bound priority of the INPUT–Queue the first time he enters the ROLLOUT–Queue, is to give short jobs a shorter turnaround time.

Fig.12 shows that there are 5 classes of jobs depending on their origin. For each class the priority parameters for the queues, the ageing rates and the service parameter as CPU time slice and CM time slice can be set independently. The setting of the parameters is done at assembly time of the system but they can be changed by the operator in the running system. Additional "Delay Parameters" control how often scheduler and priority aging program have to be called.

Summarizing the facts up to now we can see that there is a dynamic scheduling of jobs being in INPUT and ROLLOUT–Queue in concurrency with active jobs.

One slightly different strategy is used in the case of interactive jobs performing output–operations. To free memory from these jobs the output is done from disk rather than from a memory buffer. That means an interactive job which has used his time slices and has output data available for the terminal is swapped to disk regardless of the load of the system and the priority level of other jobs. A special PP–routine will then route the output from disk to the terminal.

All these operations described are performed by routines which are dynamically loaded from disk to any free PPU.

The scheduler (1SJ) can be called by several other routines after certain events which have managed the system status. Monitor calls him at least every time his delay period has elapsed (Fig.13).

Figures 14 and 15 give an overview of the schedular folw.

First thing 1SJ does is to check if the priority aging routine 1SP has to be called. If yes, 1SP will increase the priorotes of all jobs in the queues and after completion call 1SJ back.

The next task of 1SJ is to do necessary housekeeping functions of the active jobs. That is mainly to establish a memory map.

If there is any active job requesting more storage we try to fulfil the request. If not, we search the queues for a possible candidate to swap in. This is done by algorithm 1 (Fig.17).

Note that on occurrance of equal priority jobs the one with the bigger memory requirements is preferred ("best fit").

If there is no candidate to be found, 1SJ will perform the EXIT functions (Fig.16).

If there is a candidate, we try to satisfy his memory requirements. Either there is already enough free memory then 1SJ branches to SCJ4, or it tries to free space using algorithm 2. (Fig.18). The interesting fact of this algorithm is that 1SJ searches the active jobs starting with lowest priority (TACP is sorted in descending order of priorities) and collecting their used space. If he found enough but too much, he searches once again downwards the list if one of these lower priority jobs could live in the excess memory, so it would not be necessary to swap him. This, at that point very simple task, ensures best memory usage while avoiding unnecessary swaps.

Starting at SCJ4 finally 1SJ tries to find a Control Point for the candidate to swap in using algorithm 3 (Fig.19). This algorithm looks for a best fit Control Point with minimum swaps or stroage moves.

The EXIT functions of 1SJ try to schedule a candidate (by repearting the whole thing if there was no success the first time), but also to keep scheduling activities in reasonable limits.

The system default values are 1 second recall time for the schedular and 16 seconds for the priority aging routine.

Összefoglaló

A CDC NOS – operációs rendszerének scheduling algoritmusai

K. Tantscher

A dolgozat ismerteti a CDC Network Operating System scheduling algoritmusait. Ezek az algoritmusok új, elméleti tudományos eredményeken alapulnak, és egyidejüleg törekednek az erőforrások hatékony kihasználásán és a minél rövidebb kiszolgálás biztositására.

Резюме

Алгоритмы расписании операционной системы CDC NOS

К. Танчер

В работе дается обзор об алгоритмах расписании NOS. Эти алгоритмы основываются на новых теоретических научных результатах и одновременно они стараются использовать эффективно ресурсы, и обеспечивать наиболее кароткое обслуживание.

CDC CYBER 170 SERIES

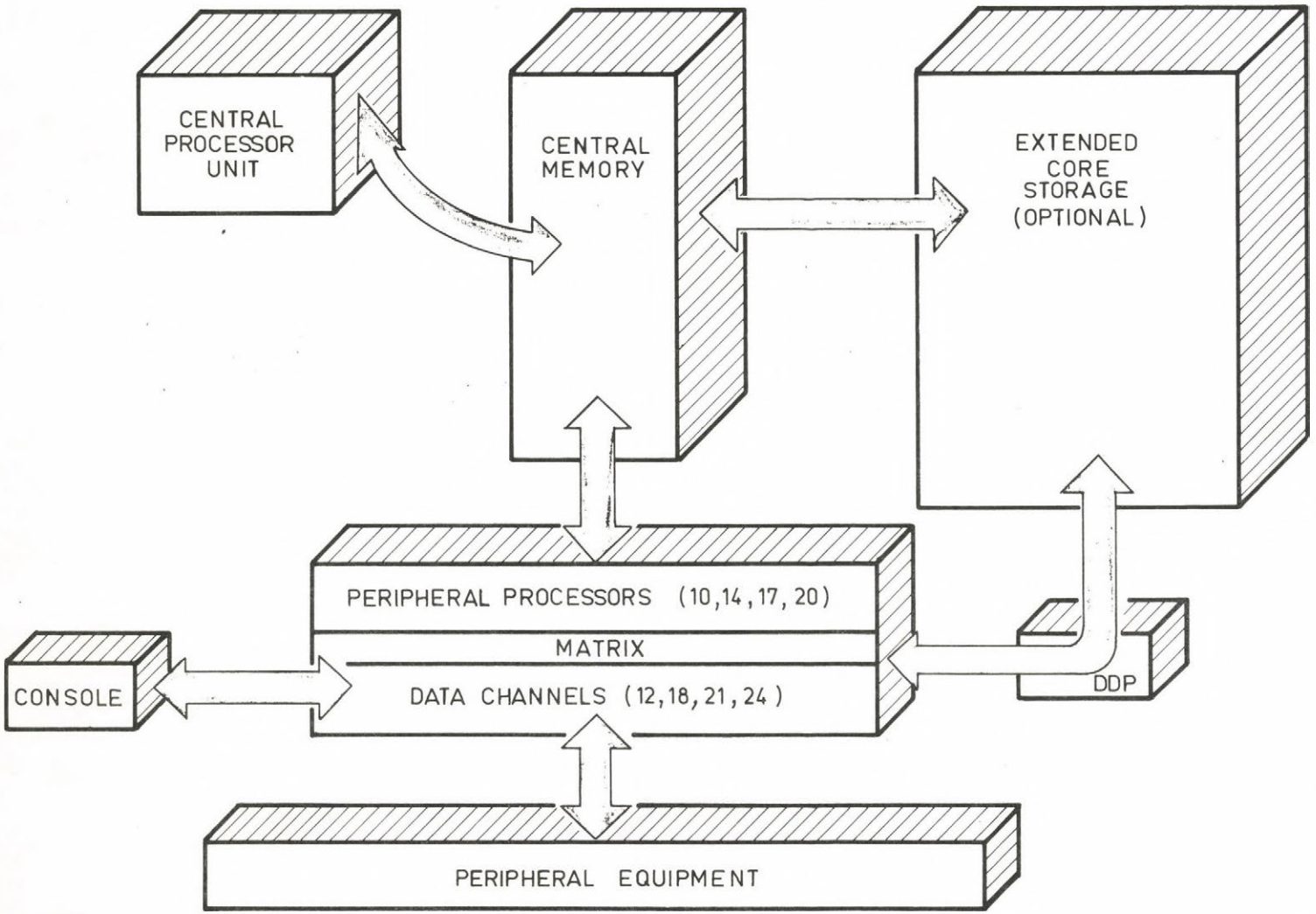


Fig.1.

SYSTEM LAY-OUT

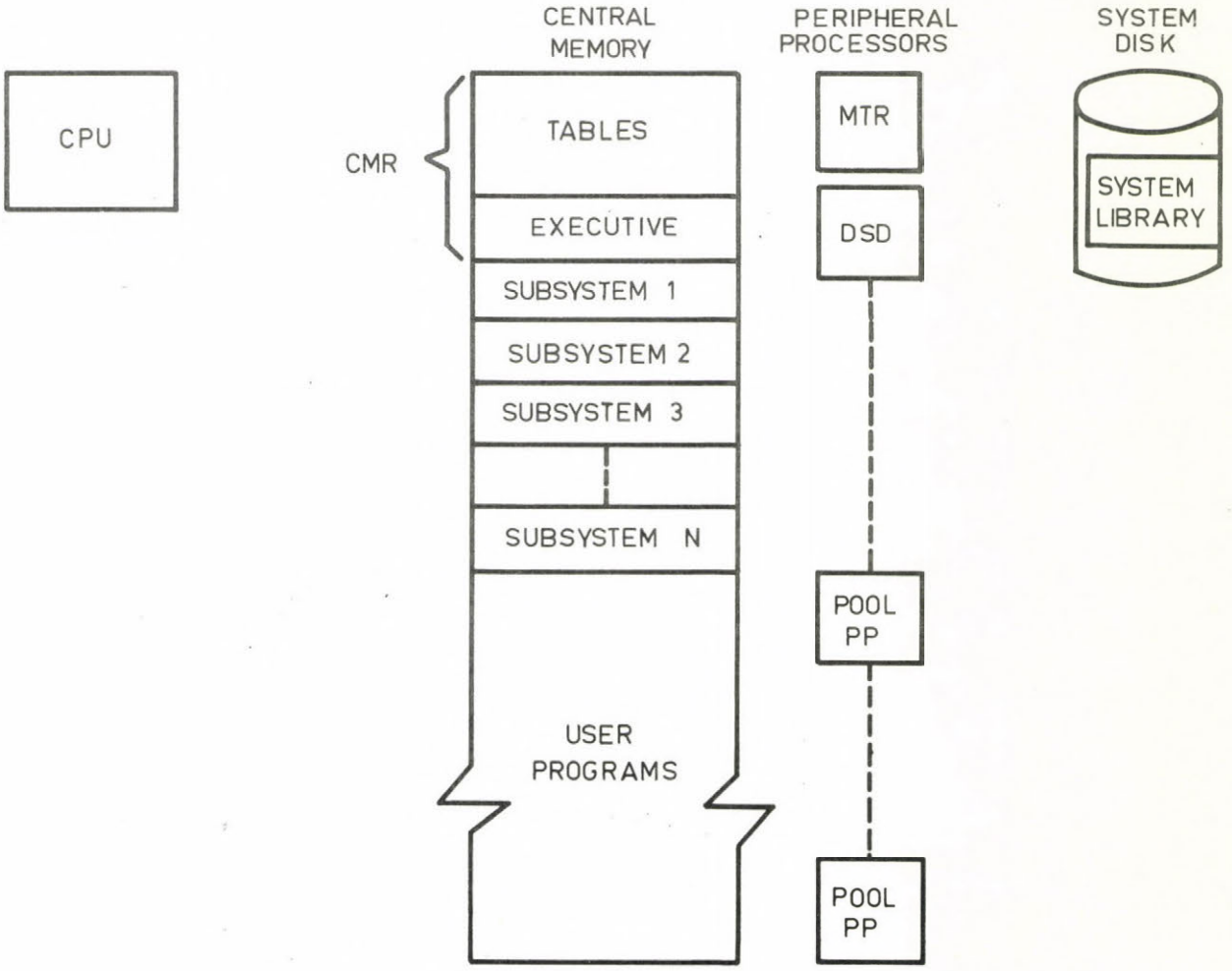


Fig.2.

MULTI - PROGRAMMING

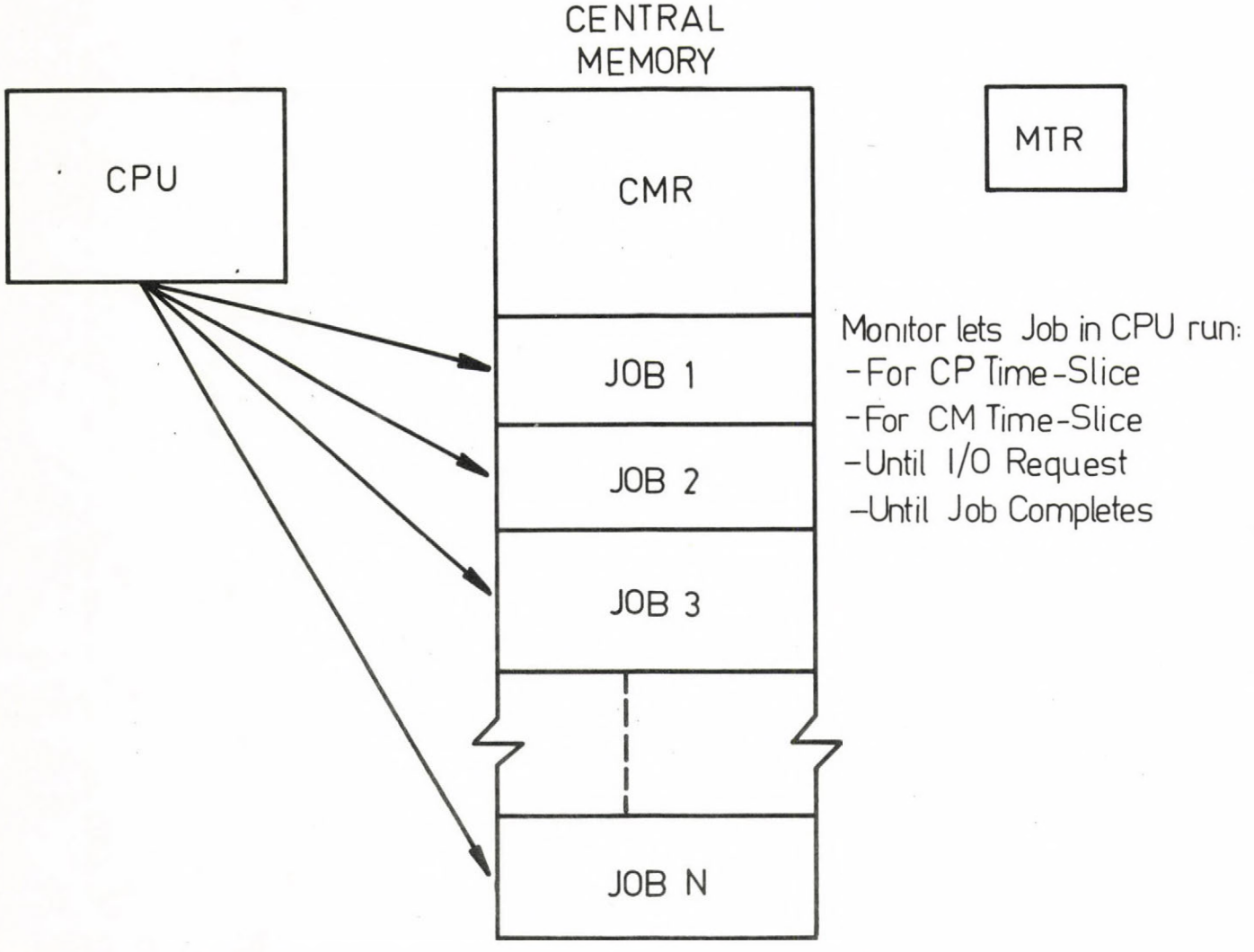


Fig.3.

CONTROL POINT CONCEPT

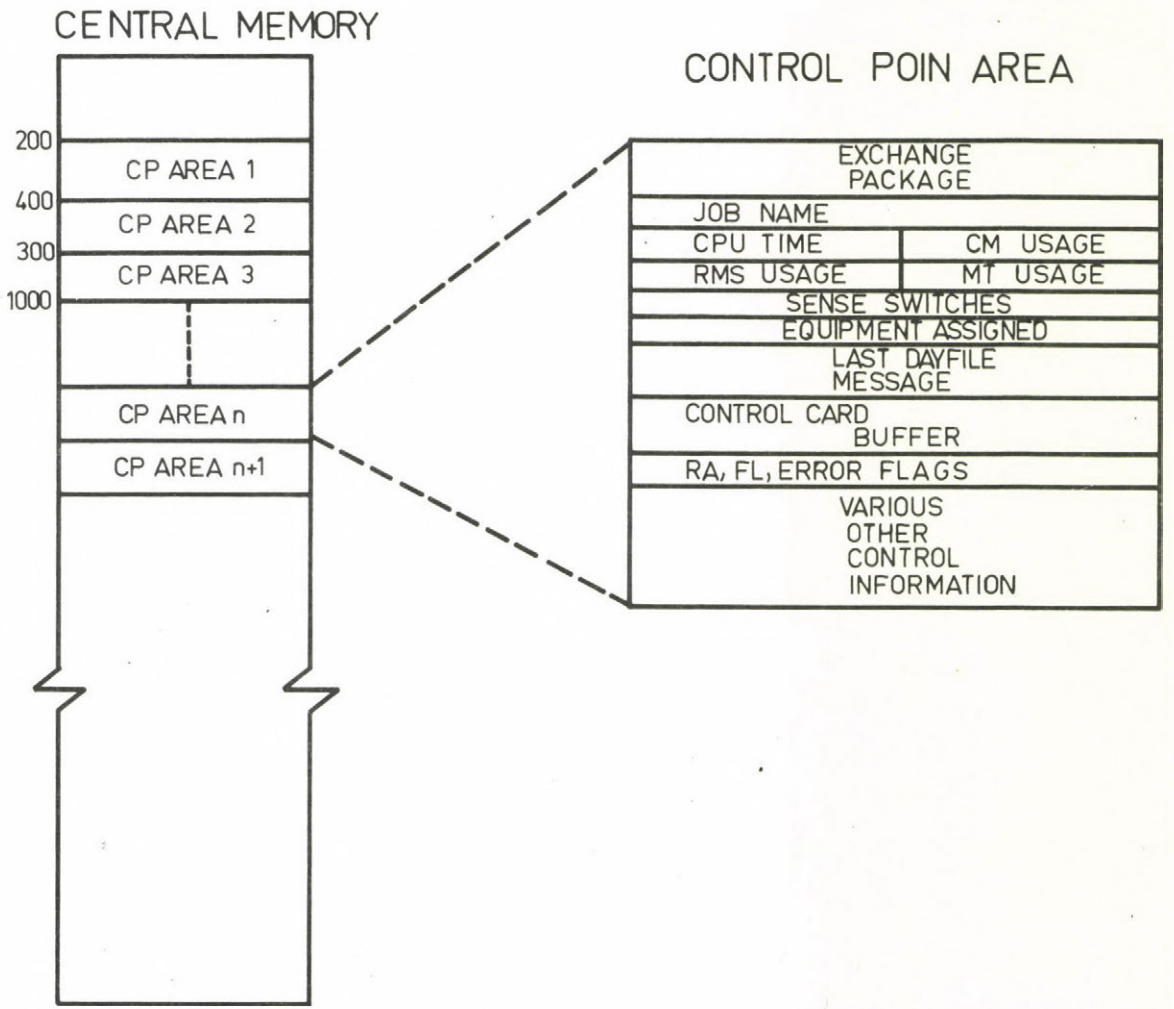


Fig.4.

MEMORY PROTECTION

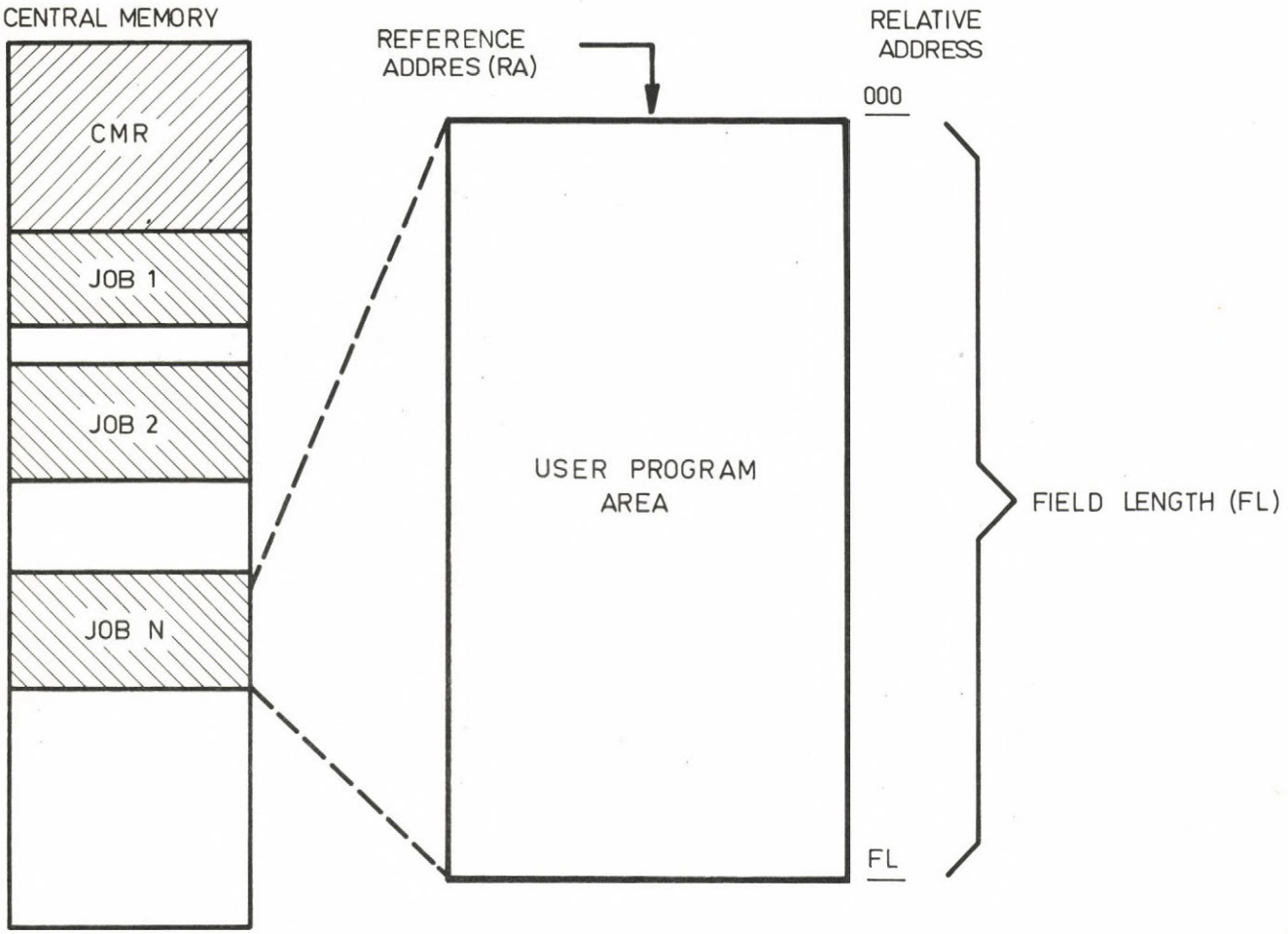


Fig.5.

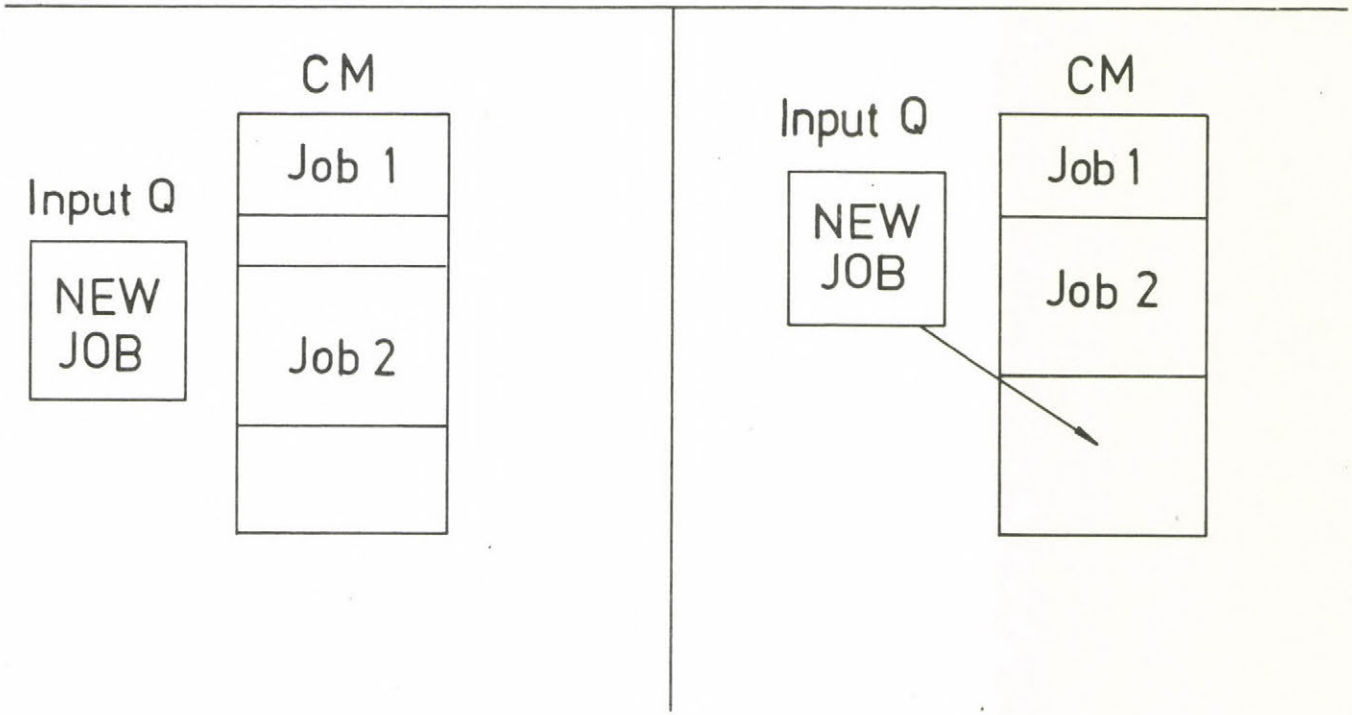

STORAGE MOVES
CM
Input Q
NEW JOB
Job 1
Job 2
CM
Input Q
NEW JOB
Job 1
Job 2

Fig.6.

SWAPPING

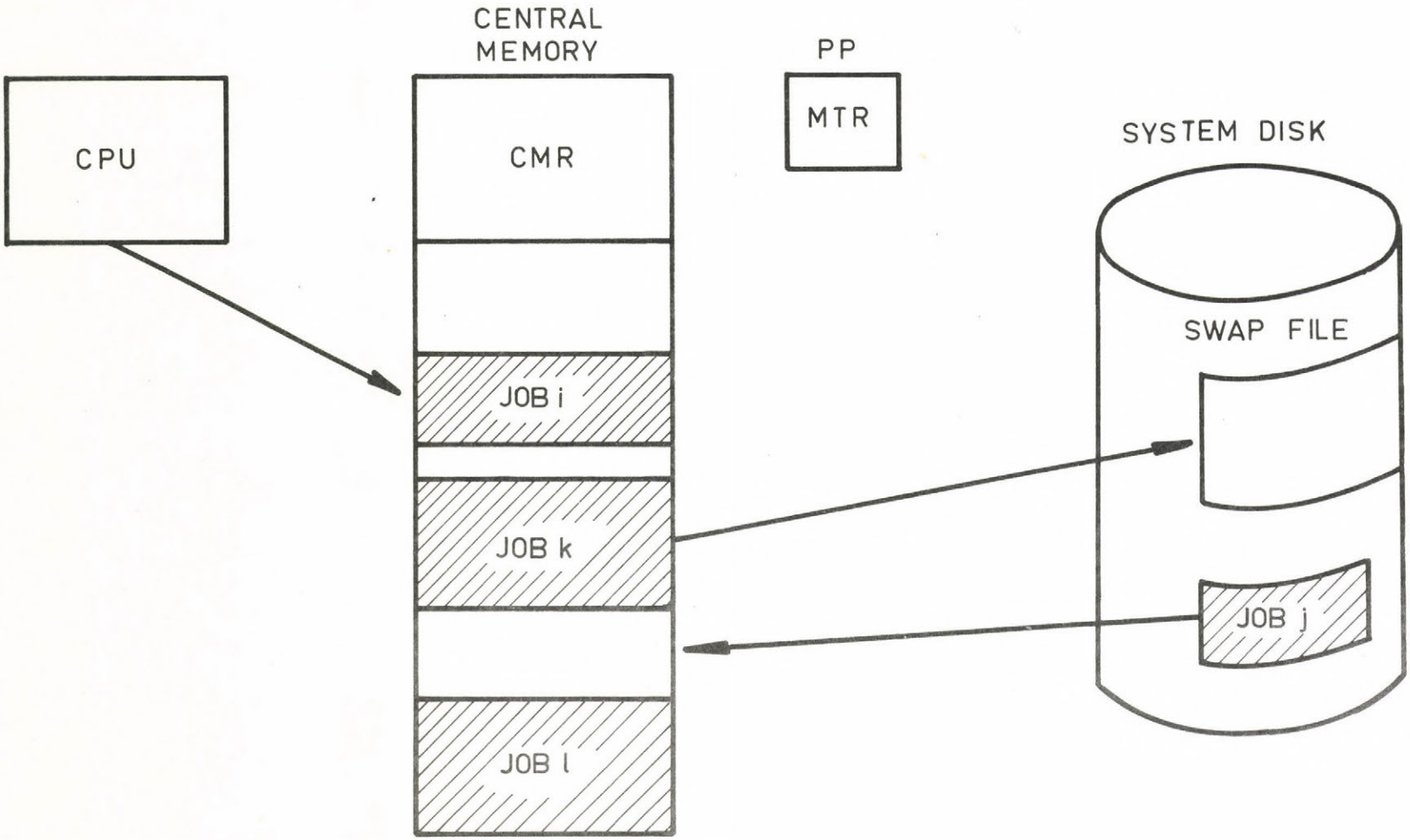


Fig.7.

JOB SCHEDULING-ENTRY INTO SYSTEM

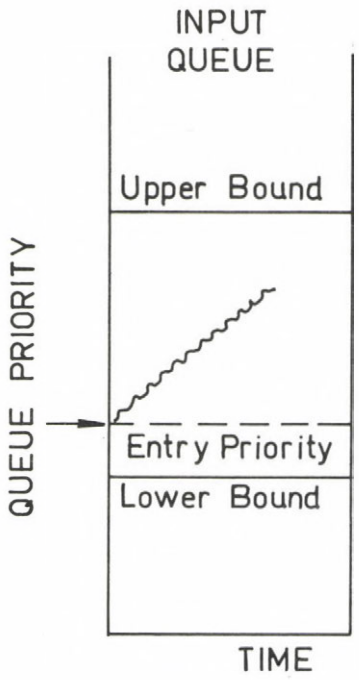


Fig.8.

JOB SCHEDULING-ENTRY INTO A CONTROL POINT

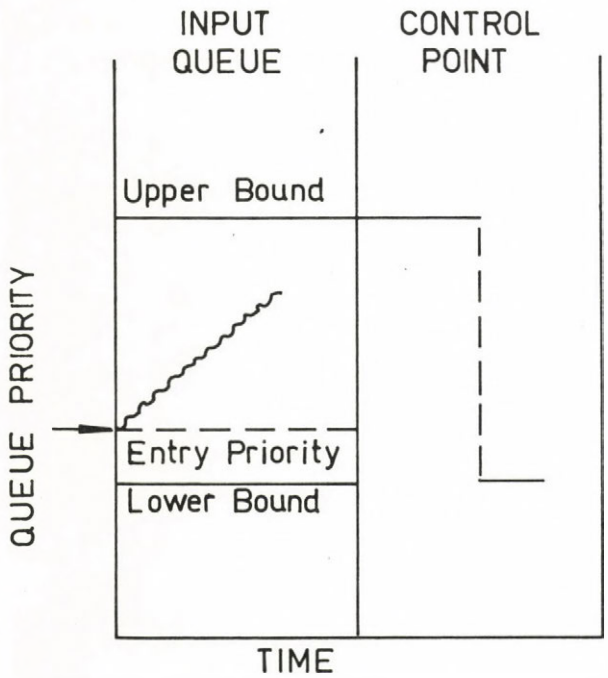


Fig.9.

JOB SCHEDULING-ENTRY INTO ROLLOUT QUEUE

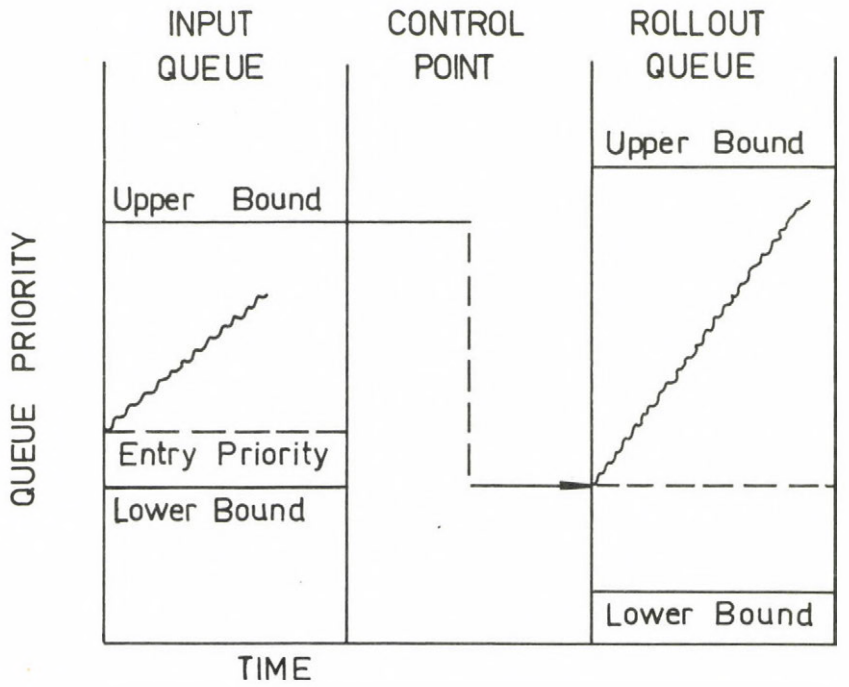


Fig.10.

JOB SCHEDULING –
LEAVING AND RE-ENTERING ROLLOUT QUEUE

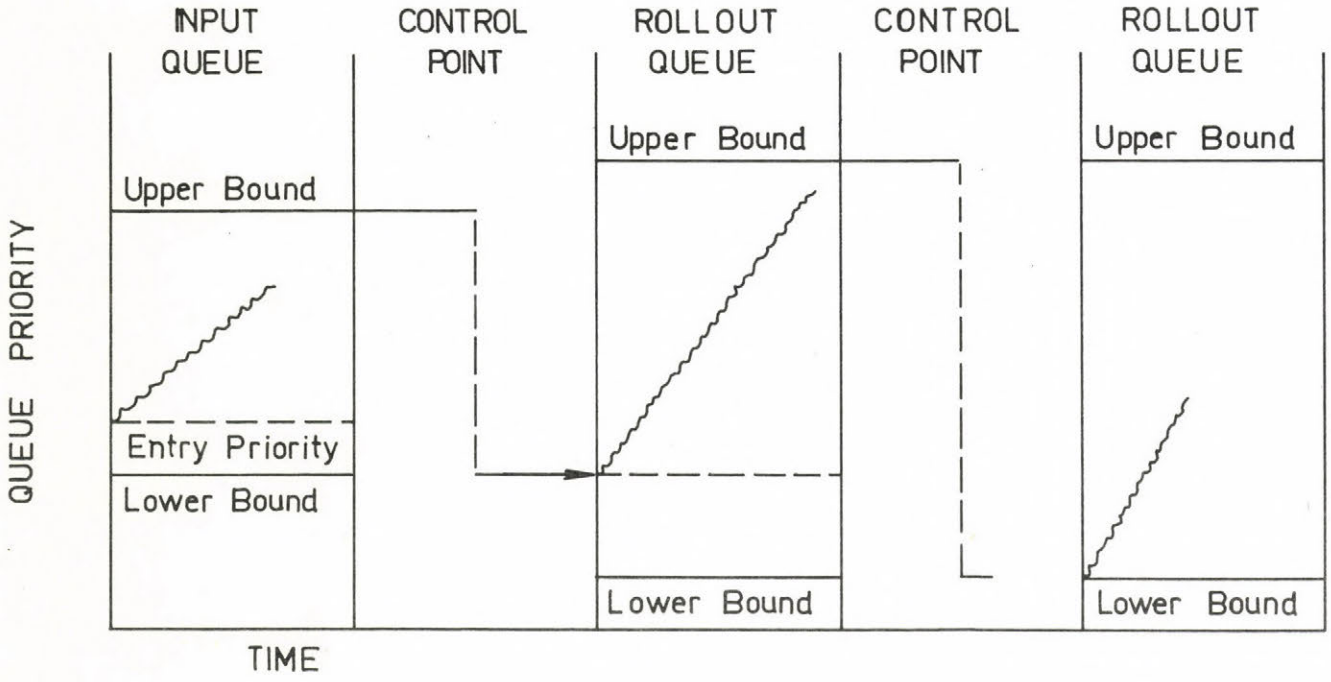


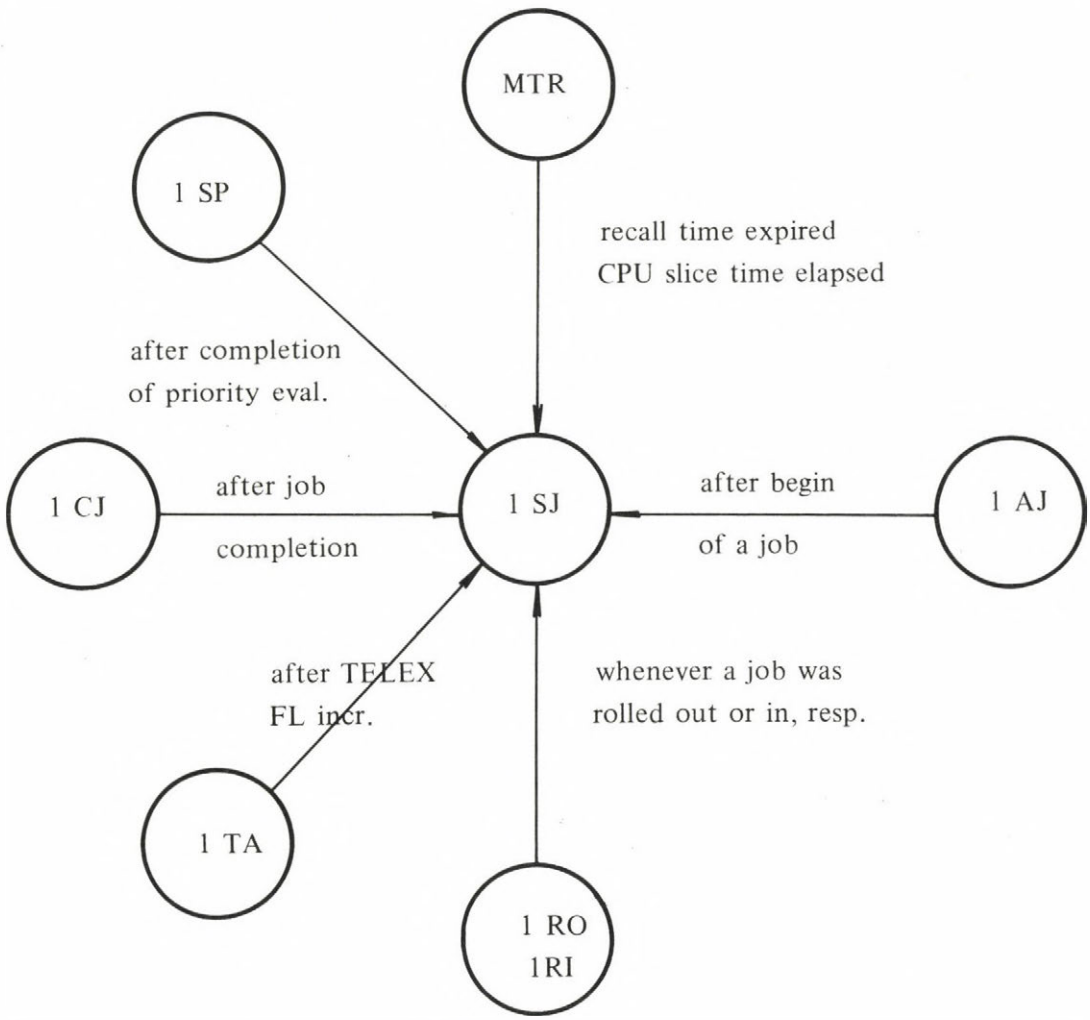
Fig.11.

JOB ORIGIN TYPE	QUEUE TYPE	QUEUE PRIORITY ENTRY PRIORITY	LOLER BOUND PRIORITY	UPPER BOUND PRIORITY	INCREMENT	TIME SLICE CPU	TIME SLICE CM	INITIAL CPU PRIORITY
SYSTEM	INPUT	6600	700	3000	1	100	20	1
	ROLLOUT	6600	100	1000	2			
	OUTPUT	400	100	7700	1			
BATCH	INPUT	2400	2000	4010	1	400	200	30
	ROLLOUT	2400	1010	4004	1	400		
	OUTPUT	200	100	7000	2			
EXPORT/ IMPORT	INPUT	3400	2400	4010	1	400	200	30
	ROLLOUT	3400	1400	4006	1	400		
	OUTPUT	200	100	7600	1			
TELEX	INPUT	4000	3770	7006	1	40	30	30
	ROLLOUT	4004	3740	7000	1			
	OUTPUT	200	100	7000	1			
MULTI-TERMINAL	INPUT	6774	6700	7400	1	400	60	31
	ROLLOUT	6774	4000	7400	1			
	OUTPUT	6000	100	7700	1			

DELAY PARAMETERS

JS	CR	AR	JA	CS
1	10	200	10	10

Fig.12.



Philosophy: 1SJ should be called when the system status has changed

Fig.13.

SCHEDULER FLOW

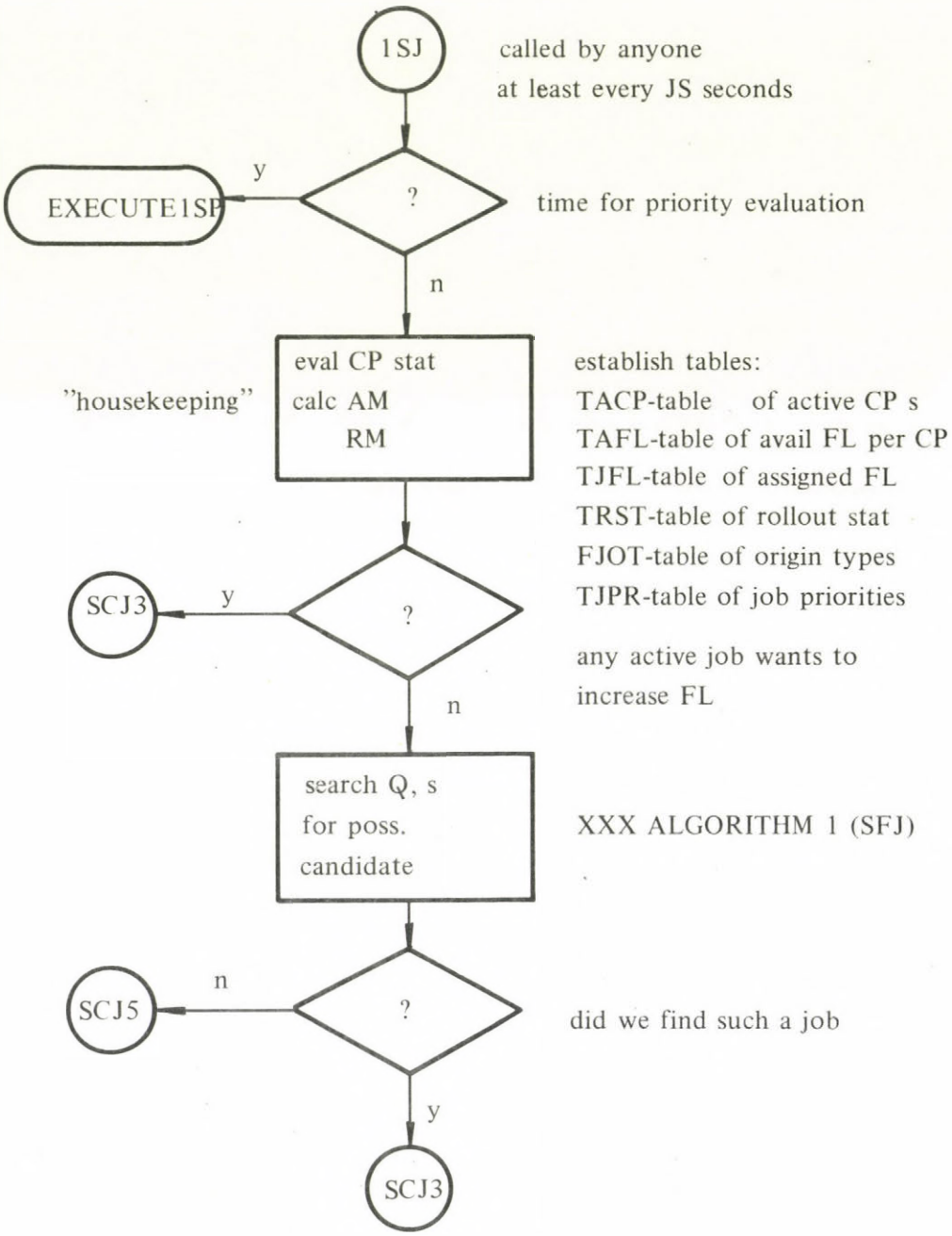


Fig.14.

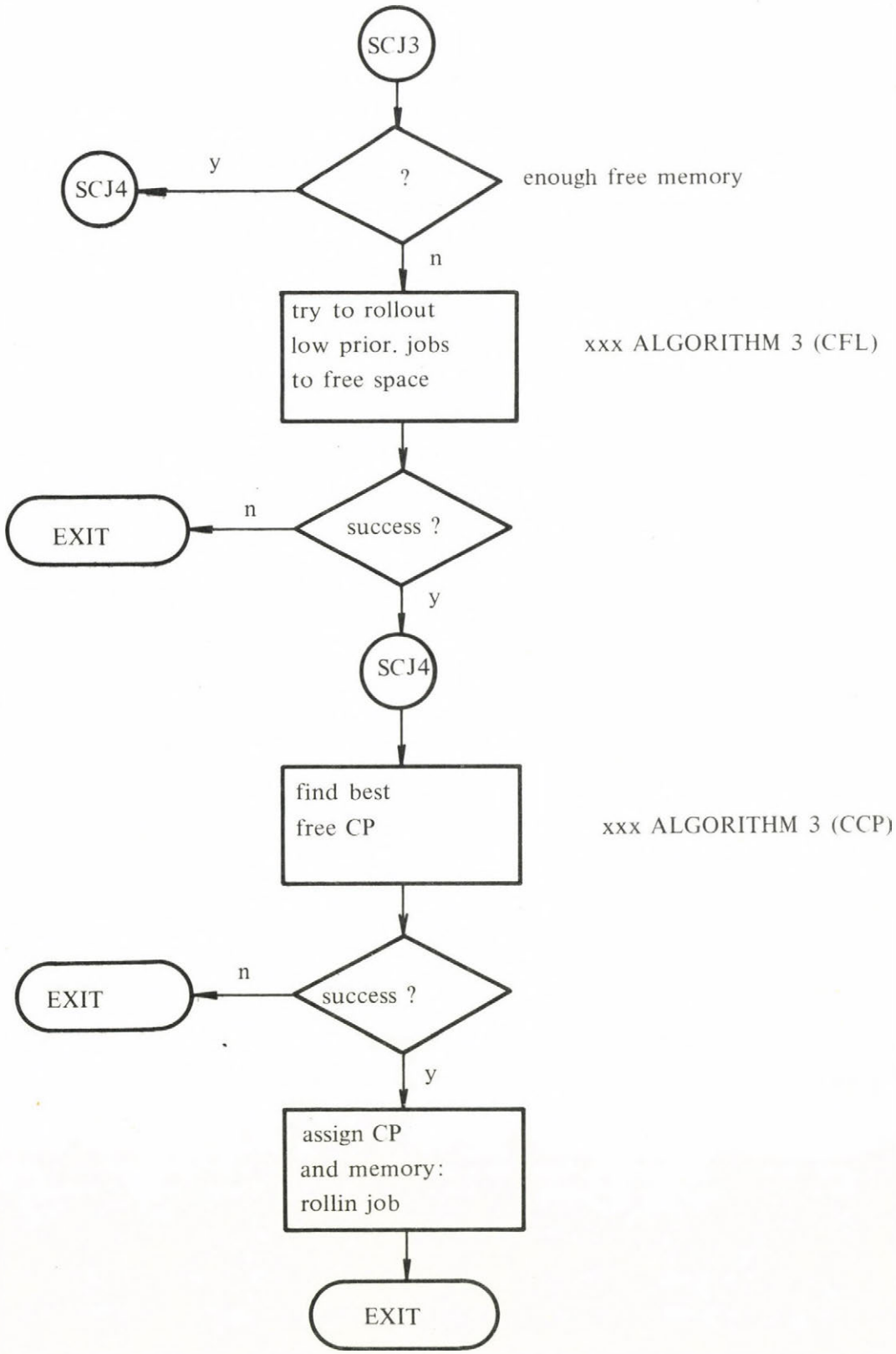

SCJ3
?
y
SCJ4
enough free memory
n
try to rollout
low prior. jobs
to free space
xxx ALGORITHM 3 (CFL)
success ?
n
EXIT
y
SCJ4
find best
free CP
xxx ALGORITHM 3 (CCP)
success ?
n
EXIT
y
assign CP
and memory:
rollin job
EXIT

Fig.15.

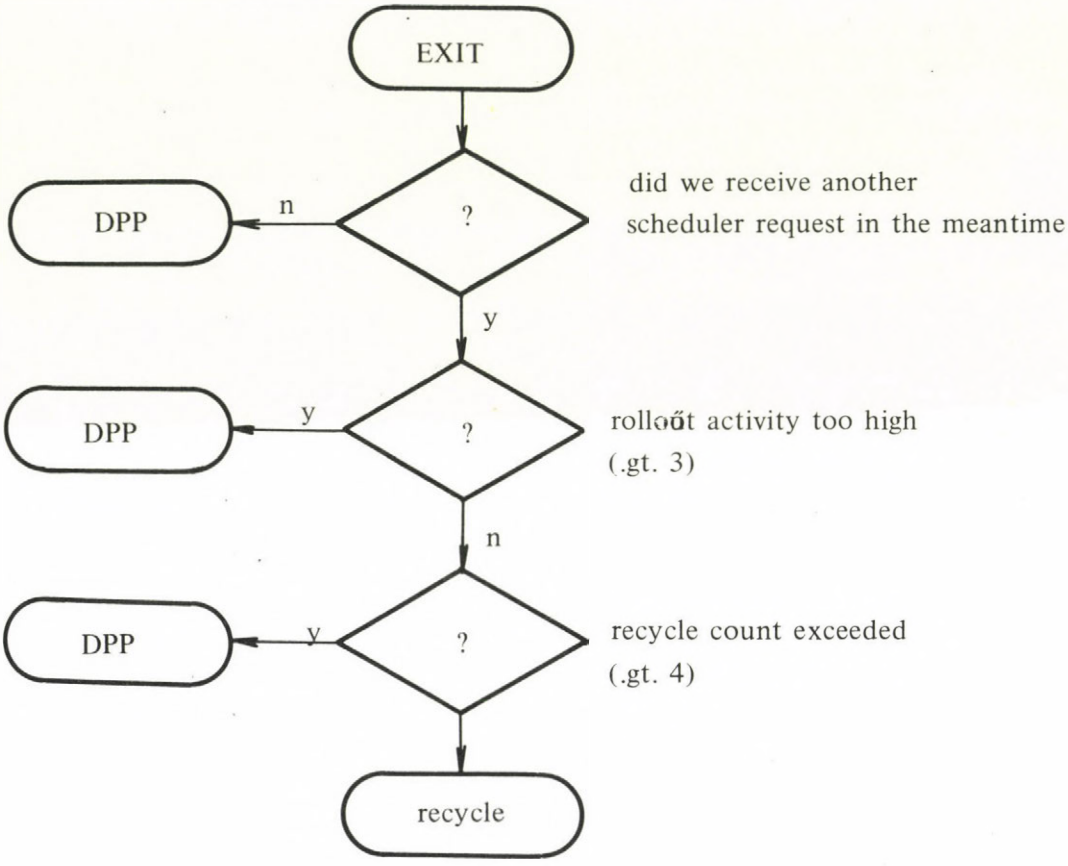

EXIT
DPP
n
?
did we receive another
scheduler request in the meantime
y
DPP
y
?
rollout activity too high
(.gt. 3)
n
DPP
y
?
recycle count exceeded
(.gt. 4)
recycle

Fig.16.

ALGORITHM 1

SFJ – search for job

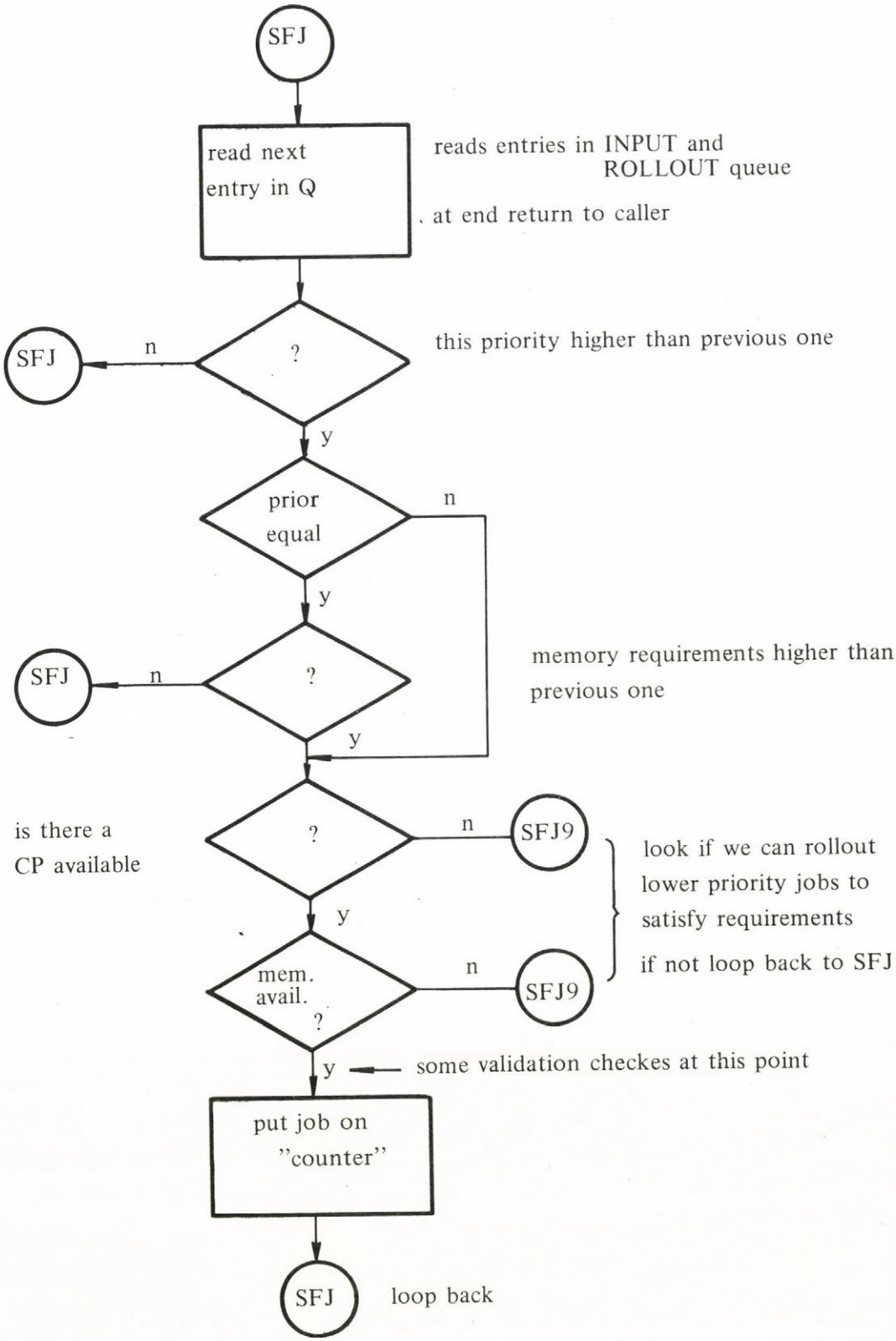


Fig.17.

ALGORITHM 2

CFL – commit field lenght

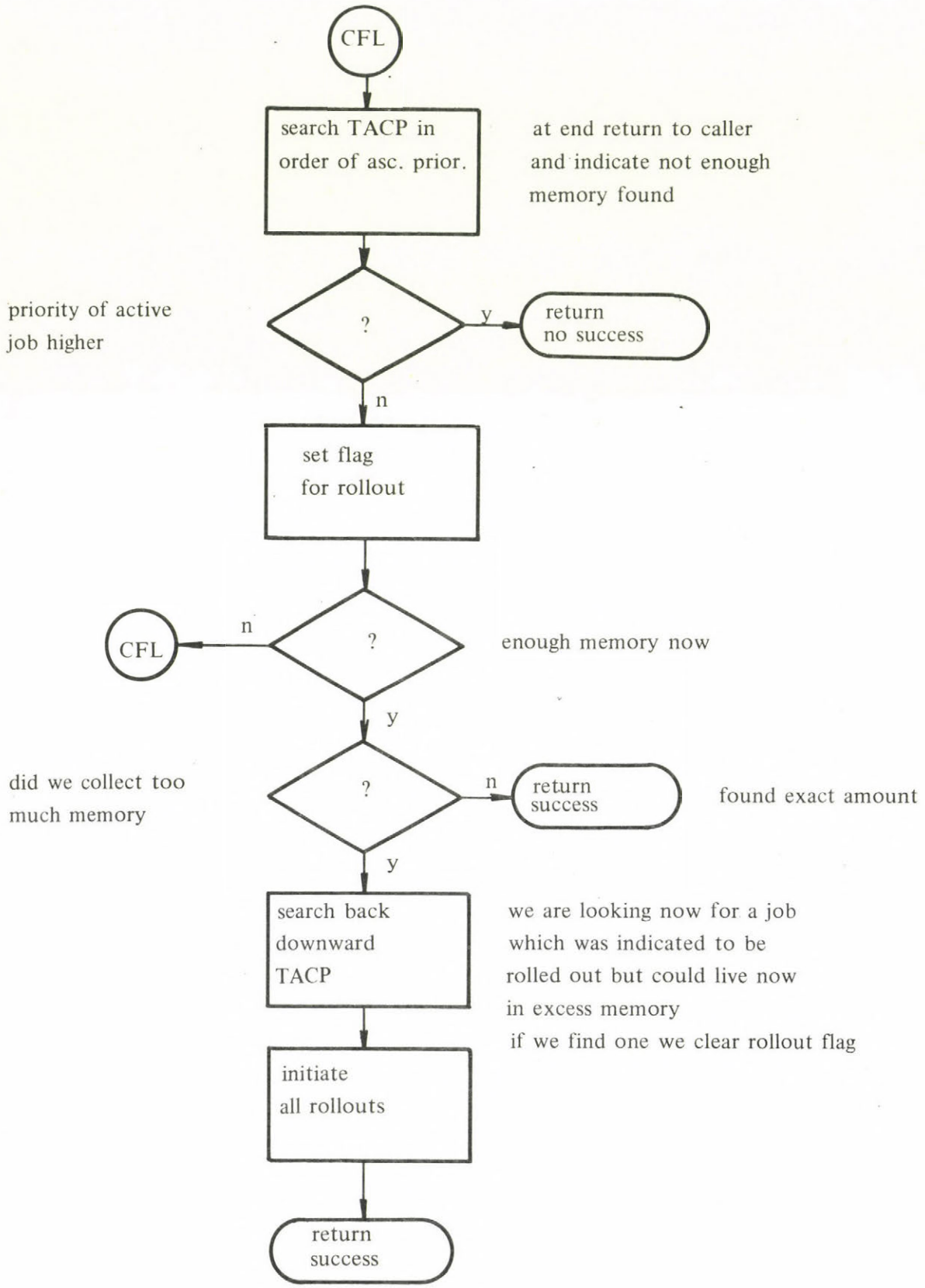
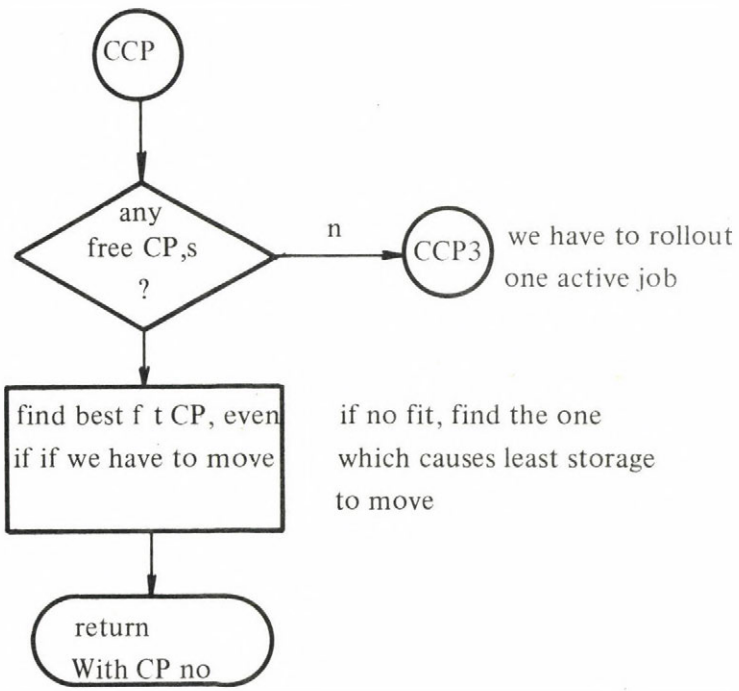


Fig.18.

ALGORITHM 3

CCP – commit control point



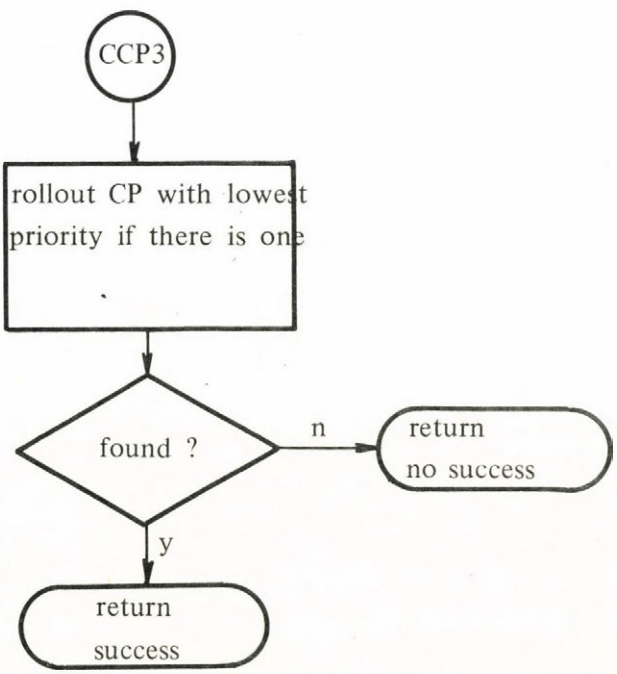


Fig.19.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

PATTHELYZET OPERÁCIÓS RENDSZEREKBEN*

(Összefoglaló)

Quittner Pál

Egyetemi Számitóközpont

A multiprogramozás bevezetése nagymértékben megnövelte a számitógépeken adott idő alatt elvégezhető munka mennyiségét, de az egymással párhuzamosan futó folyamatok (programok) koordinálása uj problémákat vetett fel. Az egyik ilyen probléma a patthelyzet (deadlock) kialakulásának elkerülése.

Patthelyzet akkor következik be, amikor két vagy több folyamat vár olyan lefoglalt erőforrásokra, melyek normális üzemmód mellett soha nem lesznek számunkra elérhetők és ezáltal ugy tartják fel egymást, hogy egyikük sem tud továbblépni. Tipikus példa erre direkt kártyaolvasó és sornyomtató használatánál az, amikor az egyik program lefoglalta a kártyaolvasót és kéri a sornyomtatót, mig a másik már korábban lefoglalta a sornyomtatót és most kéri a kártyaolvasót.

Az ilyen és ehhez hasonló jellegü patthelyzet létrejötte jelentős mértékben megnöveli a feldolgozási költségeket. Az az idő, amig az operátor észreveszi, hogy a gép áll, mindenképpen kárbavész. Ehhez még hozzájárulnak a patthelyzet megszüntetésének, általában valamelyik program ujrainditásának költségei.

Az operációs rendszerek erőforrás allokálását grafikusan általában egy erőforrás, vagy egy erőforrás–folyamat irányitott gráffal szokták ábrázolni (1. ábra) [1,2]. Könnyen belátható, hogy a patthelyzetnek mindkét ábrázolási módnál szükséges feltétele a gráfon belüli kör kialakulása. A számitógépes feldolgozáshoz jól algoritmizálható az erőforrásoknak, az igényeknek és a pillanatnyilag lefoglalt erőforrásoknak mátrixban való ábrázolása [3,4].

Általánosan megfogalmazva a patthelyzet létrejöttének szükséges feltétele [1], hogy a rendszerben legyen legalább két olyan erőforrás, melyeket

(1) több folyamat igényelhet egyidejüleg, de csak egy foglalhat le kizárólagos használati joggal,
(2) ezeket az erőforrásokat más folyamatok ideiglenesen se tudják "kölcsönkérni",
(3) ezeket a lefoglalt erőforrásokat a lefoglaló folyamatok akkor se engedjék el, ha más erőforrások hiánya miatt várakozó helyzetbe kerültek.

A patthelyzet kivédésére szolgáló módszerek ezen szükséges feltételek valamelyikét szüntetik meg állandóan vagy ideiglenesen.

*Az előadás részletes szövege az Automatizálás 1975 juniusi számának 18. oldalán jelent meg.

Az *eleve kizárás* módszere nem engedi meg, hogy több aktiv folyamat igényelhesse ugyanazokat az erőforrásokat. A nagy számitógépgyártó cégek leggyakrabban ezt az algoritmust alkalmazzák. Legegyszerübb formájában az üzemező (job scheduler) addig nem indit el egy folyamatot, amig a futás folyamán szükséges és az (1) – (3) feltételeket kielégitő összes erőforrást hozzá nem rendelte.

Az *elkerülés* módszere elvileg megengedi a folyamatok inditásakor a patthelyzet létrejöttének veszélyét. Az egyes erőforrások tényleges hozzárendelésénél azonban futás közben dinamikusan megvizsgálja, hogy az igy kialakuló erőforrás szétosztásánál kialakulhat-e patthelyzet, és ha igen, akkor a hozzárendelést nem engedélyezi [3,4].

Az *észlelés és megszüntetés* módszerénél a rendszer megengedi a patthelyzet kialakulását, de állandóan figyeli, hogy az bekövetkezett-e. Amennyiben igen, akkor lefoglalt erőforrások felszabaditásával oldja ezt fel.

A modern számitógépeknél a hardware erőforrások legnagyobb részére az (1) vagy (2) feltétel nem teljesül. Pattveszélyt leginkább a mágnesszalag egységek, mágnesszalag file-ok és a több felhasználó által módositható mágneslemez file-ok jelentenek. Az 1. táblázatban összefoglaltuk, hogy a legfontosabb erőforrásoknál a patthelyzet szükséges feltételei közül melyik nem teljesül, illetve ha ezek mindegyike fennáll, akkor a három fő módszer közül melyikkel szüntetik meg a patthelyzet veszélyét. A 2. táblázatban összehasonlitottuk ezen módszerek legfontosabb előnyeit és hátrányait.

Adatbázis kezelő rendszereknél, ahol elvileg több program is módosithatja ugyanazon adatokat uj, a standard operációs rendszerek erőforrás–allokálásához képest lényegesen nagyobb nehézségek jelentkeznek. Ezek közül a legfontosabbak a következők [5]:

– Az erőforrások nincsenek egyértelmüen definiálva (többszörös nevek).
– Az erőforrások dinamikusan változnak és ezért az igényeket rendszerint nem lehet előre teljes mértékben meghatározni.
– Az erőforrások (az önállóan "lezárható" rekordok) száma több nagyságrenddel nagyobb.

Ezen nehézségek miatt adatbázis kezelő rendszereknél még nem alakult ki egységes módszer a pattveszély kiküszöbölésére. Az eddig alkalmazott algoritmusok [5,6] csak ad hoc jellegü megoldások.

Irodalom

[1] E.G. Coffman Jr.: Deadlocks in Computer Systems, in Operating Systems, Infotech State of the Arts Report. 1972, 353.o.

[2] A.J.T. Colin: Introduction to Operating Systems. 14. fejezet, MacDonald (American Elsevier) London, New York, 1971.

[3] A.N. Haberman: Prevention of System Deadlocks. CACM 12 (1969) 373, 385.

[4] R.C. Holt: Comments on Prevention of System Deadlocks. CACM 14 (1971) 36.

[5] D.C. Chamberlain, R.F. Boyce, I.L. Traiger: A Deadlock Free Scheme for Resource Locking in a Data Base. Proc. IFIP 74, 340. o, North Holland, 1974.

[6] A. Shoshani, A.J. Bernstein: Synchronization in a Parallel–Accessed Data Base. CACM 12 CACM 12 (1969) 604.

Summary

Deadlock in operating systems
P. Quittner

The necessary conditions for deadlock are summarized and methods to avoid it (prevention preemption and detection) are compared. Special deadlock problems arising from simultaneos updating in a data base are discussed.

Р е з ю м е

Наличие "пата" в операционных системах
Пал Квиттнер

В статье обобщаются условия необходимые при "пате" и сравниваются различные методы его обхода (а именно исключение, обход, изъятие). В статье также обсуждаются специальные проблемы, возникающие при параллельной модификации базисов данных.

ERŐFORRÁSOK TULAJDONSÁGAI A PATTHELYZET SZEMPONTJÁBÓL

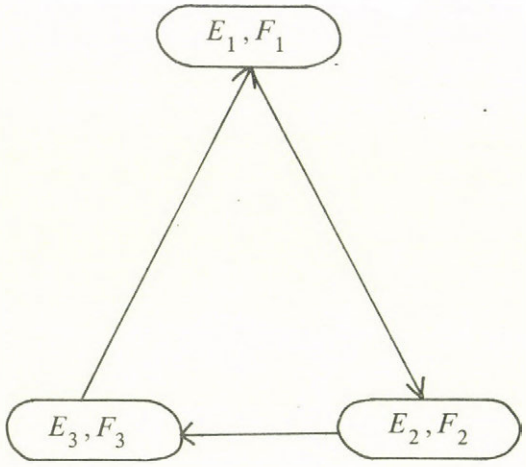
Erőforrás	Patthelyzet szükséges feltételei közül melyik nem teljesül	Patthelyzet megakadályozásának módszere
Központi memória	Kölcsönkérhető, illetve elengedi (csak roll–in roll–out vagy vagy virtuális tárnál)	Eleve kizárás
Processzor (CPU)	Kölcsönkérhető a megszakitási rendszer révén	
Csatornák	Elengedi	
Lassu perifériák	Közös virtuális I/0 egységek nem foglalhatók le kizárólalagos használatra	Megszüntetés (az operátor elveszi)
Mágnesszalag egységek és file-ok	A file megnyitása előtt és lezárása után kölcsönkérhetők, illetve elengedheti a rendszer	Eleve kizárás, elkerülés
Mágneslemez egységek	Közösen használhatók	
Mágneslemez file-ok	Olvasáshoz közösen használhatók	Módositásnál: eleve kizárás vagy elkerülés
Rendszerprogramok és táblázatok	Reentrant programok közösen használhatják	Eleve kizárás

1. Táblázat

PATTHELYZET MEGAKADÁLYOZÁSÁRA SZOLGÁLÓ MÓDSZEREK ÖSSZEHASONLITÁSA

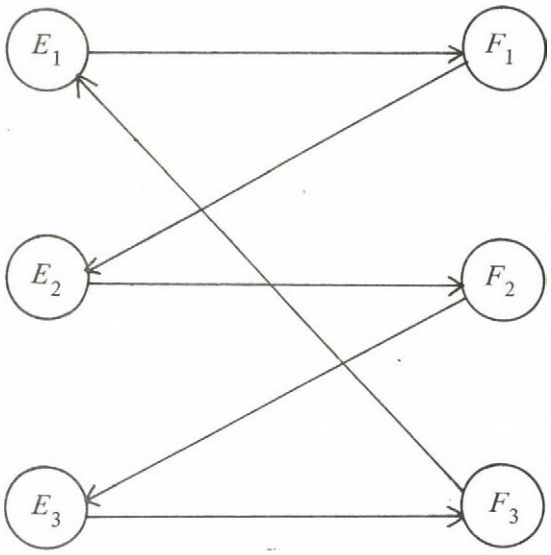
Módszer	Előny	Hátrány
Eleve kizárás	Gyors algoritmus, Soha sem lehet patthelyzet, Eredeti job-prioritás megmarad	Fölöslegesen lefoglal erőforrásokat, ezáltal a multiprogramozás lehetőségeit erősen csökkenti. Előzetes információk kellenek az erőforrás igényekről.
Elkerülés	Jobban gazdálkodik az erőforrásokkal, Soha nem lehet patthelyzet	Rendszeresen megismétlendő algoritmus minden lefoglalásnál. Bonyolultabb algoritmus (különben uj problémák jelentkeznek). Multiprogramozás lehetőségeit csökkenti, mert fölöslegesen nem enged lefoglalni pattveszélyt előidéző erőforrásokat. Előzetes információk kellenek az erőforrás–igényekről.
Észlelés	Egyszerü algoritmus, Összes szabad erőforrás allokálható, Nem kell előzetes információ az erőforrás –igényekről.	Patthelyzet kialakulhat, megszüntetése igen költséges lehet.

2. Tábázat



Folyamat	Erőforrás	
	Kért	Lefoglalt
F_1	E_2	E_1
F_2	E_3	E_2
F_3	E_1	E_3

a.)



b.)

1. ábra

Erőforrás allokálás grafikus ábrázolása

a.) erőforrás gráf

b.) erőforrás – folyamat gráf

A patthelyzet szükséges feltétele a gráfon belüli kör.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

A SCRATCH–POOL KIHASZNÁLTSÁGÁNAK MÉRÉSE

Tóth Beatrix – Tőke Pál

MTA Számitástechnikai és Automatizálási Kutató Intézete

A CDC-3300-as gépek MASTER operációs rendszere a programok input és output adatait egy speciális disk területén az u.n. scratch pool-on tartja. Ennek a disk területnek a dinamikai kezelése lehetővé teszi, hogy a programok össz igénye – melyet jelenleg előre meg kell adni – meghaladja a fizikai scratch pool területet. Ennek oka, hogy a programok futása során a valódi scratch terület igénye időben véletlenszerüen változik és csak ritkán éri el a maximumot. Több program egyidejü futásánál a véletlen ingadozások miatt az együttes scratch pool igény jóval alatta marad az egyes igények maximumai összegének.

A dolgozatban megvizsgáljuk a scratch igény változását, az egyes jobokra vonatkozóan, mint stochasztikus folyamatot. A statisztikai vizsgálatokon kivül megadjuk a scratch pool jelenlegi kezelésének leirását és megvizsgáljuk a dinamikus kezelés megvalósitásának lehetőségét.

1. A scratch pool kezelési mechanizmusa, vezérlési sajátságai

A scratch pool-n az allokálás minden file esetén szegmens méretben történik. (A szegmens installációs paraméter – jelenleg 1 szegmens = 11 200 szó.) A pool nagysága 250 szegmens.

A *scratch pool belső adminisztrációja* a MASTER operációs rendszerben három rekesz segitségével történik:

PSCRATCHF – a pool méretét tartalmazza szegmensekben

PSCRATCHL – megadja. hogy az adott pillanatban a rendszer mennyi logikai pool igényt képes kielégiteni

OC.TSCNT – megadja, hogy az adott pillanatban a rendszer hány szabad szegmenssel rendelkezik

Egy *job* esetén a teljes logikai pool igény:

$$scr + pun + out + abort$$

ahol mindegyik szegmens méretben értendő. Az igény a job teljes igényét jelenti, illetve a scr igény esetén a fellépő maximális igényt *scr*-vel jelöljük és a többiekre vonatkozóan a t időpontig fellépő igények összegét $pun(t)+out(t)+abort(t)$-vel jelöljük. Látható, hogy ezek időbeli változásának vizsgálata több job együttes futása esetén értékes megtakaritást jelentene, ha minden t időpontban a pillanatnyi job igényt kellene csak lefoglalni a pool számára a disken. Az egyes

igények nagysága nem haladja meg a 63-t (out és abort együtt értendő).
(Ennek oka, hogy a File Definition Table-ban a szegmensszámra 6 bitnyi hely van.) Nyilván a teljes pool igény nem áll fent végig a futás alatt.

A job teljes logikai pool igénye nem haladja meg a PSCRATCHF-t, ellenkező esetben a rendszer a jobot nem schedulálja.

A job inditásánál – mint minden igénynek – a logikai pool igénynek is kielégithetőnek kell lenni, tehát értéke ⩽ PSCRATCHL. A jobok inditására logikailag ez nem szükséges általánosságban ld. pl. [4]. A MASTER által alkalmazott ezen inditási módszer egy lehetséges megoldás a " deadlock " állapotok elkerülésére. Ld. pl. [1]. Az inditásnál történik meg a teljes out, pun, abort igény lefoglalása.

Mivel a scratch pool logikai értékét nagyobbra állitják be a fizikainál, elképzelhető, hogy az allokálandó szegmensszám meghaladja a rendelkezésre álló fizikai szegmensszámot. Ebben az esetben az igényt kibocsátó task speciális SEGTAB WAIT állapotba kerül. Mihelyt az OC.TSCNT tartalma nő, a task ujra kibocsátja az allokálási igényét.

A befejezéskor a TRMOVR, az operációs rendszer egy taskja, a felesleges szegmenseket visszaadja a rendszernek, de az out file -t a nyomtatás, a pun file-t a lyukasztás végéig fenntartja.

A fenti kezelési mechanizmus a következő hiányosságokkal rendelkezik:

Az out file teljes nagyságban már az inditáskor létezik – pedig erre csak később van szükség. A felhasználónak a szükséges sorigény többszörösét adják meg. Végül, a rendszer az igényelt sorokat teljesen kitöltötteknek tekinti, és ennek megfelelően foglalja le a helyet.

Az emlitett problémák megoldására Tőke Pál tett javaslatot [2]-cikkében. Ennek lényege a következő:

a.) kezdetben a rendszer csak egy bizonyos nagyságu out file-t foglalna le a job számára, később ezt szükség szerint növelné.

b.) amennyiben az out file elérne egy előre meghatározott nagyságot, a rendszer ezt automatikusan lezárná és nyomtatná. Ennek az optimális méretét kellene meghatározni. Erre vonatkozó elméleti eredmények találhatók Tomkó [3] dolgozatában. Az optimalizáláshoz szükséges alapvető mérések eredményeit a dolgozatban közöljük.

MASTER 4.0 operációs rendszer már tartalmaz egy PPOF makrot és taskot. Ezt a felhasználónak kell hivnia. Hivás esetén a makro az out file-t két részre osztja: a teleirt szegmenseket lezárja és kinyomtatja, az üres szegmenseket tekinti ezután az out file-nak. Hiányossága ennek a megoldásnak, hogy az inditáskor a teljes file lefoglalás megtörténik, a fölösleges szegmensek a terminálásig megmaradnak.

2. Mérési eredmények

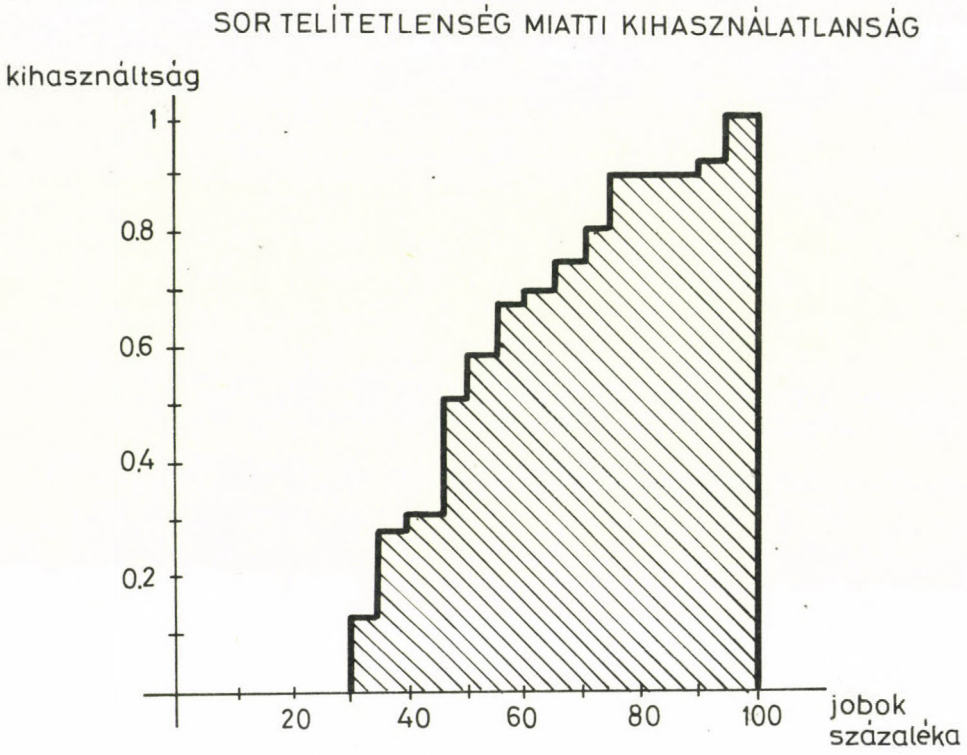
Alapvető célkitüzésünk, hogy az inditáskor az igényeknek megfelelő nagyságu out file-t foglaljunk le, amit szükség esetén bőviteni lehet. A kezdeti szegmensszám meghatározásához mértük a schedulált szegmensek számát, valamint a teleirt blokkok számát. Ugyancsak gyüjti a mérési program a kinyomtatott sorok számát, a kezelési és a terminálási időt, végül a terminálás módját (normál vagy abnormál).

A mérés pontos mechanizmusa. A job terminálásakor a TRMOVR-ben elérhető a job tábla cime. Ennek segitségével végigkerestük a Primary Task List-t (ahol minden program taskhoz és minden nyitott 1/0 file-hoz egy háromszavas bejegyzés tartozik), hogy az illető job file-jának FCI tábla ciméhez hozzájussunk. Majd ebből a megfelelő File Definition Table cimét olvastuk ki. Ez utóbbiból kaptuk adatainkat.

A mérés megvalósitása. A méréseket 1974. októberében normál user futások alatt végeztük kb. 21 óra hosszat (5 alkalommal). Ez idő alatt a lefutott jobok száma 646 volt.
Méréseink két hiányosságát emlitem meg.

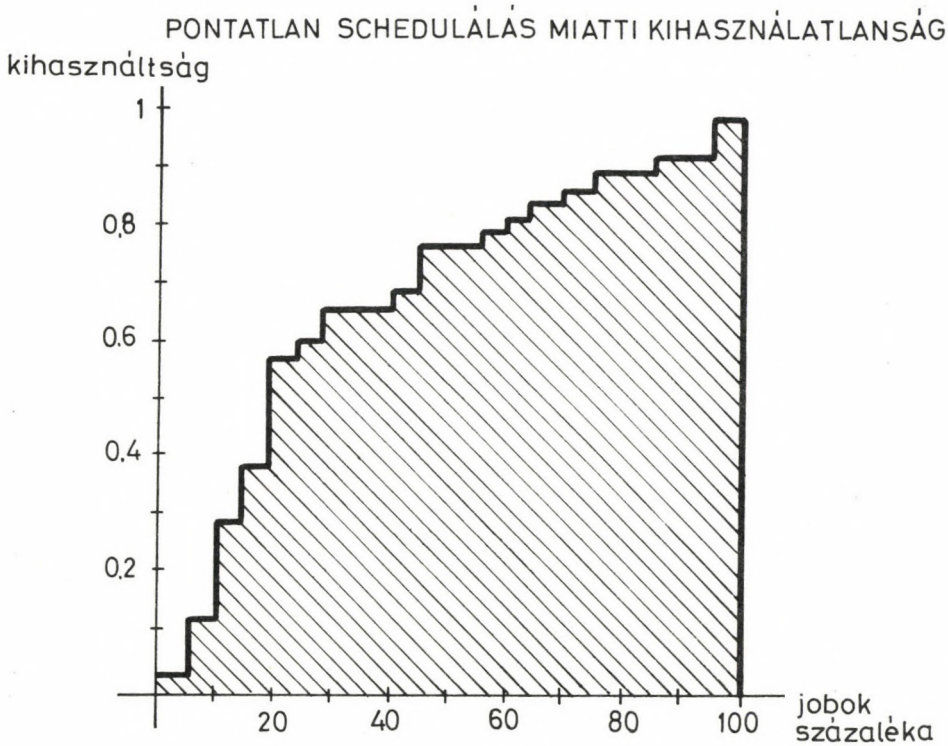
Ugy tekintettük, hogy az out file a terminálásig létezik, pedig a printelés befejezéséig megmarad – de ez utóbbi időpontot nem tudjuk pontosan meghatározni.
Az abnormálisan terminált jobokat figyelmen kivül hagytuk. Ennek oka az volt, hogy az abort paraméter használata lényegesen megváltoztatja az out file kihasználtságot.

Vizsgáltuk a *sorok telitettségét.* Ezen mutató meghatározásánál a következő mérési adatok álltak rendelkezésünkre: Ismerjük a kinyomtatott sorok számát, továbbá azon disk - terület nagyságát, amelyen ezen sorok ténylegesen tárolódnak. Az operációs rendszer maximálisan teleírt sorokat tételez fel, amikor kijelöli az OUT file méretét. E két adat, a ténylegesen mért block-szám továbbá az adott sormennyiségnek megfelelő disk-terület egymáshoz való viszonyitása adja a sortelítetlenségi mutatót. Ennek várható értékére 0,53-at szórására pedig σ^2_T = 0,035-öt kaptunk. Az empirikus eloszlásfüggvényt az alábbiakban szemléltetjük.

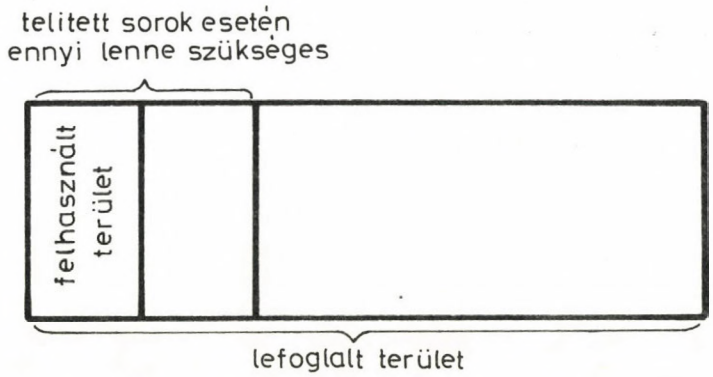


1.ábra

A pontatlan schedulálás mutatóját a kinyomtatott sorokhoz szükséges szegmensek számának, és a ténylegesen lefoglalt szegmensszámnak a hányadosából kapjuk. A sorok számából a szegmensméretre való áttérésnél az operációs rendszer schedulálási algoritmusát vettük alapul, így a sorokat teljesen kitöltöttnek tételeztük fel. Ez a mutató átlagban 0,35 volt, tehát durván háromszor nagyobb terület foglalódik le a diskben indulásnál még ideális esetben is (teljes sortelítettség feltételezése) mint amennyi valójában szükséges. Ha még figyelembe vesszük a sortelitetlenségi defektust is, akkor ez az ellentmondásos helyzet még élesebben rajzolódik ki. Ezen kihasználtsági mutató szórása $\sigma^2_T = 0{,}07$-nek adódik. Az alábbi ábra empirikus eloszlásfüggvényt szemlélteti.

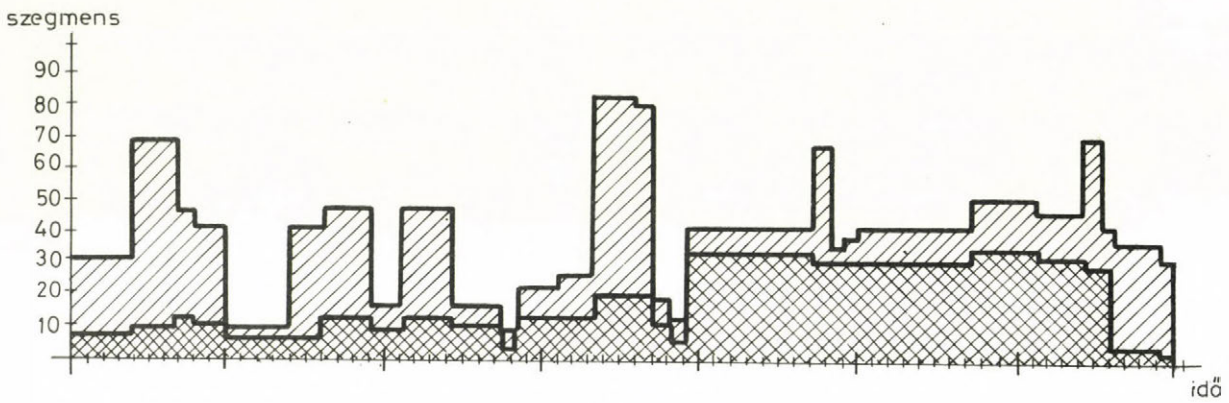

PONTATLAN SCHEDULÁLÁS MIATTI KIHASZNÁLATLANSÁG
kihasználtság
1
0.8
0.6
0.4
0.2
20
40
60
80
100
jobok
százaléka

2.ábra.


telített sorok esetén
ennyi lenne szükséges
felhasznált
terület
lefoglalt terület

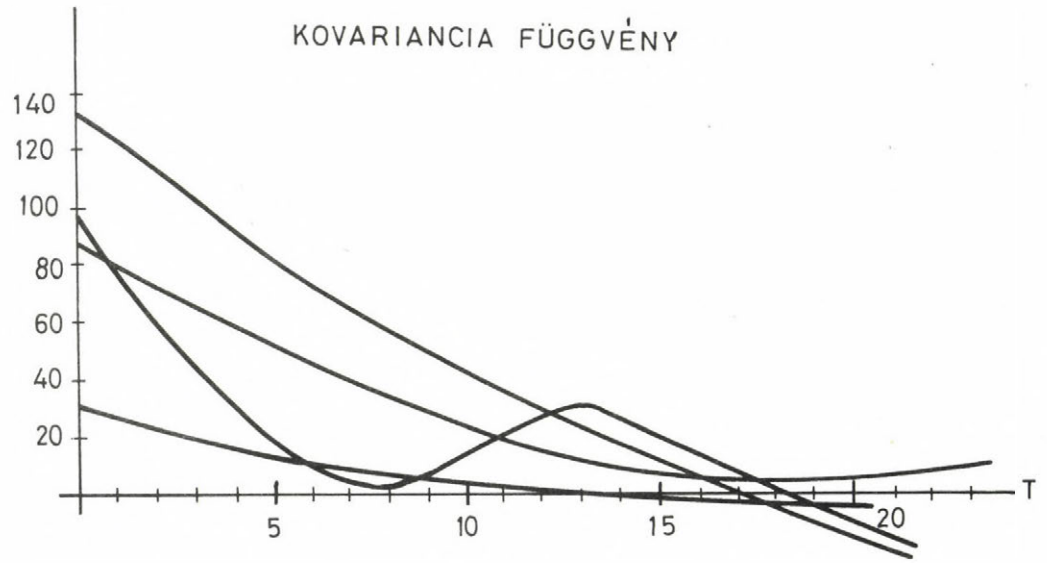
3.ábra.

A következő ábra az out igények alakulását mutatja: a lefoglaltat és a ténylegest. A mintát percenként vettük kb. 5 óra hosszat (– itt 70 percet ábrázoltunk). Hosszabb mérés esetén ez a folyamat sztochasztikus változást mutat és időben folytonosnak vehető. Első közelitésben egy 2 dim-s Gauss–Markov folyamatnak tekinthetjük. Ha a későbbiekben ez jelentős eltérést mutat, akkor korrigáljuk.



4. ábra.

Az empirikus kovarancia függvény azt mutatja, hogy 16-18 percenként gyakorlatilag már nincs korreláció:



5 ábra.

A helyfoglalás, mint emlitettük, szegmensenként történik. Egyszerű mutatóval próbáltuk objektive mérni, hogy a szegmensméret megválasztása (installálási probléma) mennyire durva. Lényegileg a "túlschedulálást", ami a lefoglalás szegmensnyi kvantumos voltából adódott, mértük *sor-egységben* és viszonyitottuk a tényleges sor-igényhez. Nyilván ez a viszonyszám kisebb scr-igényü joboknál nagyobb, mig a nagy nyomtatási igény mellett elenyésző.
A szegmensméret megválasztásának jóságát éppen az adja, hogy ez a mutató mikor válik megnyugtatóan kicsivé. Gyakorlatilag 8 lefoglalt szegmensnél ez a mutató 0,0025% körül van, ami elfogadható. (8 szegmens kb. 2000 sor-nak felel meg).

A fenti mérési eredmények elegendő alapot szolgáltatnak arra, hogy sokkal dinamikusabb SCRATCH–POOL kezelő mechanizmust dolgozzunk ki. A kapott eredmények szimulációs módszerek alapjául szolgálhatnak, mivel a valóságos helyzetet tükrözik.

Elképzeléseink az új kezelési politika gyakorlati megvalósitására már készek. Jelenleg a MASTER 4.0 operációs rendszerhez történő illesztésén fáradozunk.

Irodalom

[1] Colin, A.S.T.; Introduction to operating systems, 14. fejezet, Mac Donald (American Elsevier) London, New York, (1971).

[2] Tőke P., Dynamic use of the scratch-pool, Proceedings of the Computer Science Conference, 117-121 pp. Székesfehérvár (1973).

[3] Tomkó J., Studynig Output Organization with a Single Finite Quence, Proc of. Conputer Science Conference, 113 - 117 pp. Székesfehérvár (1973).

[4] IBM. OS/360 Consepts and Facilites. In Programming Lauguages and Systems. (S. Rosen, Ed.) Mc Graw - Hill, 1967, 598 pp. New York.

Summary

A scratch – pool utilization study

B. Tóth, P. Tőke

The MASTER operating system, for the CDC 3300, keeps the job's input, output and scratch files on a common disk are called SCRATCH-POOL. In our paper we discuss the management of this pool and the statistical investigation about the exploitation of the OUT file (fullness of lines, defekt in consequence of the inaccurate scheduling, examination of the segment size).

On the basis of our results we think possible the elaboration of a more dynamic SCRATCH-POOL management.

Р е з ю м е

Об эффективности дисковой памяти операционной системы MASTER

Б. Тот, П. Токе

Операционная система MASTER для машины CDC-3300 распределяет дисковую память из общего массива ОС так называемого SCRATCH-POOL для вводного массива с перфокарт, для выводного на печать, и для рабочих массивов заданий.

В стате затрагиваются вопросы методического обращения и управления общего массива ОС, а так же произвадится статистический анализ использования выводного массива на печать (уплотнение строк, дефекты из-за неточного задания числа строк, выбор размера сегмента и т.и.).

На основе разработанных данных становится возможным осуществление более динамического механизма управления.

MTA Számítástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

A VIRTUÁLIS MEMÓRIA EGY ALGEBRAI ÉS EGY VALÓSZINÜSÉGELMÉLETI MODELLJE

Gyires Tibor

MTA Számítástechnikai és Automatizálási Kutató Intézete

Az utóbbi időben egyre nagyobb az érdeklődés az eddigiektől eltérő memória szervezés iránt, aminek a neve virtuális memória. Milyen okok tették szükségessé egy uj rendszer megtervezését, ami sok elméleti és gyakorlati probléma kiinduló pontja lett?

A számitógépek elterjedése lehetővé tette, hogy egyre bonyolultabb, összetettebb feladatokat számitógép segitségével oldjanak meg, amelyek megoldásához egyre nagyobb programokat kellett irni. A programok nagyságának a belső memória nagysága szabott felső határt. A belső memória drágasága miatt felvetődött a probléma, hogyan lehet lefuttatni egy olyan programot, ami meghaladja a memória nagyságát? A kérdésre két megoldás született:

overlay technika,
virtuális memória szervezés.

Az overlay technika a következő: a programot a felhasználó tetszés szerinti nagyságu darabokra, logikailag összefüggő u.n. szegmensekre bontja, amelyek közül egynek, a "main"–nak kitüntetett szerepe van. Ez a szegmens állandóan benn van a memóriába és ezen keresztül valósul meg a többi szegmens közötti kommunikáció. A memóriában mindig két szegmens van, a main és az aktiv szegmens. A fennmaradó szegmensek valamilyen külső tárolóra (disk mágnesszalag, stb.) kerülnek, melyek közül mindig azt cseréljük ki a memóriában levővel, amelyikre hivatkozás történik a programból.

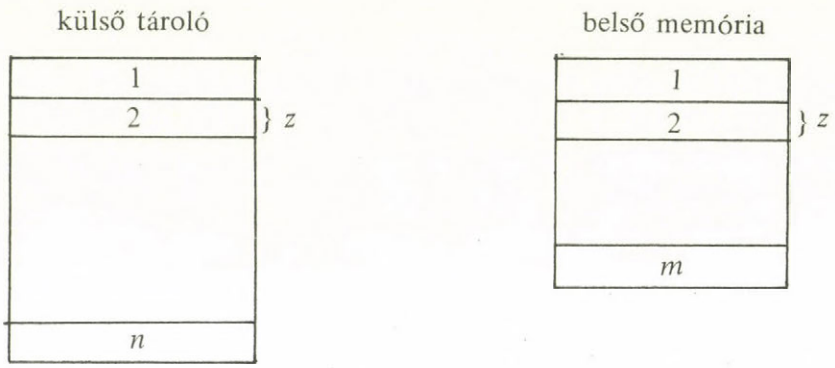
A virtuális memória szervezés esetében a programot és a memóriát változó, vagy egyenlő nagyságu nem szükségképpen logikailag összefüggő blokkokra osztjuk. Változó hosszuságu blokkok esetében szegmentált, egyenlő hosszuságu blokkok esetében page–elt szervezésről beszélünk. A blokkok között nincs kitüntetett. A nem betölthető blokkokat valamilyen külső tárolóra visszük, ahonnan szükség esetében behozzuk a memóriába. A program felosztása és a blokkok cseréje az overlay–jel ellentétben automatikus. A dolgozatban a page–elt szervezéssel fogunk foglalkozni.

Mindkét esetben az a cél, hogy a program futása közben a lehető legkevesebbszer hivatkozzon olyan programrészre, amely külső tárolón helyezkedik el, vagyis a program lehető legkevesebb interrupt–tal fusson le. A dolgozat a virtuális memória egy algebrai és egy valószinüségelméleti modelljét tartalmazza. A modellek célja különböző feltételekkel bizonyitani azt, hogy abban az esetben, ha a blokkok cseréjére az u.n. önjavitó algoritmust alkalmazzuk, a blokkcserék számát tekintve ez az algoritmus optimális.

I. Algebrai közelítés

1.1. **Definició.** Egy page–elt virtuális memória rendszernek nevezünk egy $V = (N,M,f)$ hármast, ahol $N = \{1,2, \ldots, n\}$ a programot alkotó page–eket, $M = \{1,2, \ldots, m\}$ a memória fizikai page-it jelöli, ahol $1 \leqslant m \leqslant n$ és $f: N \to M$ "page függvény" azzal a tulajdonsággal, hogy

$$f(i) = \begin{cases} j, & \text{ha az } i\text{-edik page a } j\text{-edik fizikai page-ben van.} \\ \text{undef.} & \text{egyébként} \end{cases}$$



Ha a page hossza z (pl. z szó), akkor egy α virtuális címre $0 \leqslant \alpha < nz$ hasonlóan egy β virtuális címre $0 \leqslant \beta < mz$. Ha adott egy α virtuális cím, a rendszer software vagy hardware uton meghatároz egy (i,w) párt ugy, hogy $\alpha = (i-1)z + w$ ahol $0 \leqslant w < z$, és generálja az α-hoz tartozó β memória cimet, amelyre

$$\beta = [f(i) - 1]\, z + w, \quad \text{ha } f(i) \text{ definiálva van.}$$

Ha az $f(i)$ page függvény nem definiált, a program végrehajtásában interrupt lép fel "page várakozás" ideig, addig amig a rendszer nem tölti be a hiányzó page-t a külső tároló egy rendelkezésre álló fizikai page-ébe és módositja a page függvényt a memória aktuális tartalmának megfelelően. Interrupt esetén a következő problémák lépnek fel:

a.) helyettesitési probléma: melyik page-t vigyük ki a belső memóriából, hogy be tudjuk tölteni a hivatkozott page-t?

b.) elhelyezési probléma: hová töltsük a bejövő page-t?

c.) betöltési probléma: mikor töltsük be a hiányzó page-t?

Ha egy page-t akkor és csak akkor töltünk be, amikor a programból hivatkozás történik rá "demand page" -elérésről beszélünk, ellenkező esetben "nondemand" vagy "prepage"-elésről.

Demand page-elés esetén az elhelyezési probléma megoldása triviális. Ha a memória még nem telt be, a bejövő page-t tetszőlegesen helyezzük el egy üres fizikai page-be, ha a memóriában már nincs üres page, először a helyettesitési problémát megoldó u.n. helyettesitési algoritmust hajtjuk végre, majd az igy kijelölt page helyébe töltjük a bejövő page-t, vagyis a demand page-elő rendszerek vizsgálatát elegendő a helyettesitési probléma vizsgálatára korlá-

tozni. Mivel a későbbiekben egy feltétel fennállása esetén kimutatjuk, hogy egy demand page-elő rendszer az interruptok számát tekintve hatékonyabb, mint egy prepage-elt rendszer, a továbbiakban demand page-elt rendszereket vizsgálunk.

Mint az előzöekben, legyenek $N = \{1,2, \ldots ,n\}$ egy program page-i, $M = \{1,2, \ldots ,m\}$ a memória fizikai page-i, és $1 \leqslant m \leqslant n$. Jelölje N^k a k hosszuságu sorozatok halmazát N fölött, $k \geqslant 0$.

Jelölje $\omega = r_1, r_2, \ldots , r_T$ N^T-beli elem egy program page hivatkozásait valamely T-re. Ha $r_t = x$, ez azt jelenti, hogy a program a t-edik időpillanatban az x page-ra hivatkozott. Legyen $S \subseteq N$ egy memória allapot és

$$\mathfrak{M}_m = \{ S / S \subseteq N \text{ és } |S| \leqslant m \}$$

ahol $|X|$ jelöli az X halmaz elemeinek a számát.

1.2. **Definició.** Legyen M és N adott. Egy M-re vonatkozó page-elő algoritmusnak nevezzük az $A = (Q, q_0, g)$ hármast, ahol

Q az algoritmus kontroll állapotainak a halmaza,
q_0 eleme Q-nak, kezdeti kontroll állapot,
g $\mathfrak{M}_m \otimes Q \otimes N \longrightarrow \mathfrak{M}_m \otimes Q$ allokáló függvény azzal a tulajdonsággal, hogy amennyiben

$g(S,q,x) = (S', q')$, akkor $x \in S'$

Jelölje $S+x$ az $S \cup \{x\}$ halmazt, $x \notin S$
és $S-x$ az $S - \{x\}$ halmazt, $x \in S$

1.3. **Definició.** Az A page-elő algoritmus demand page-elő algoritmus, ha allokáló függvénye:

$$g(S,q,x) = \begin{cases} (S,q') & \text{ha } x \in S \\ (S + x, q') & \text{ha } x \notin S \text{ és } |S| < m \\ (S + x - y, q') & \text{ha } x \notin S, \quad |S| = m, \ y \in S \end{cases}$$

Jelölje $|X_t|$ a t-edik időpillanatban betöltésre kerülő page-k számát, $|Y_t|$ a kivitt page-k számát. Az algoritmus költségének kiszámitásánál az $|Y_t|$ page kivitelének költségét figyelmen kivül hagyjuk, hiszen feltételezhetjük, hogy a betöltés a kivitellel együtt történik.

1.4. **Definició.** Jelölje $h(k)$ k page betöltésének költségét, $k \geqslant 1$, $h(k) \geqslant h(1) = 1$. Legyen $\omega = r_1, r_2, \ldots , r_T$ a program hivatkozás stringje, S kezdeti memória állapot.

Az A page-elő algoritmus S-re és ω-ra vonatkozó költsége

$$C(A,S,\omega) = \sum_{t=1}^{T} h(|X_t|).$$

Ha A demand-page-elő algoritmus, $|X_t| \leqslant 1$, $1 \leqslant t \leqslant T$ és

$$C(A,S\omega) = \sum_{t=1}^{T} |X_t|$$

Egy algoritmus optimális, ha költsége minimális.
Érvényes a következő, Peter J. Denning-től származó tétel.

1.1. **Tétel.** *Legyen* $A = (Q,q_0,g)$ *egy page-elő algoritmus, és tegyük fel, hogy* $h(k) \geqslant k$, $k \geqslant 1$, $h(1) = 1$, *Akkor létezik egy* A' *demand page-elő algoritmus, ugy, hogy*

$$C(A',S_0,\omega) \leqslant C(A,S_0,\omega)$$

bármely S_0*-ra,* ω*-ra.*

A tétel bizonyitja, hogy a virtuális memória szervezés legalább olyan hatásfoku az interruptok optimális számát tekintve, mint az overlay szervezés, ezenkivül a kűlönböző programrészek cseréje a virtudlis memória szervezés esetében automatikus, az overlay esetében ez felhasználó feladata.

Milyen algoritmus szerint cserélhetjük a page-ket, hogy a program a lehető legkevesebb interrupt-al fusson le?

Ahhoz, hogy ezt elérjük, azt a page-t kellene kicserélni, amelyre a legkésőbb történik a programból hivatkozás. Ez az algoritmus viszont feltételezi a hivatkozás string ismeretét, amely az esetek többségében nem áll rendelkezésünkre, igy az ilyen tipusu algoritmusokkal kapcsolatos eredmények elméleti jelentőségüek, a gyakorlatban nem alkalmazhatók. A gyakorlatban alkalmazott algoritmusok a page cseréig gyüjtöttt információk alapján döntenek. Két hasonló, már alkalmazott algoritmus a FIFO, LRU (Least Recently USED). A FIFO algoritmus a jól ismert first-in first-out alapon cseréli a page-ket. Az LRU azt a page-t cseréli ki, amelyre a program a legrégebben hivatkozott.

Mindkét algoritmus kontroll állapotainak halmaza $Q = N^m$, legyen Q egy eleme $q = (y_1,y_2,\ldots,y_m)$.
$|S| = m$ esetén a két algoritmus allokáló függvénye:

$$g_{FIFO}(S,q,x) = \begin{cases} (S,q) & \text{ha } x \in S \\ (S + x - y_m, q') & \text{ha } x \notin S \end{cases}$$

$$g_{LRU}(S,q,x) = \begin{cases} (S,q'') & \text{ha } x \in S \\ (S + x - y_m, q') & \text{ha } x \notin S \end{cases}$$

ahol

$$q' = (x, y_1, \ldots, y_{m-1}) \qquad \text{és ha } x = y_k$$

akkor

$$q'' = (x, y_1, \ldots, y_{k+1}, \ldots, y_m)$$

Önjavító algoritmus

Célszerünek látszik egy olyan algoritmus bevezetése, amely azt a page-t jelöli ki cserére, amelynek a relativ gyakorisága a legkisebb (egy page relativ gyakorisága alatt értjük a program futás elindulása óta a page-re történt hivatkozások számának és az összhivatkozások számának a hányadosát). Nevezzük ezt önjavító algoritmusnak.

Kontroll állapotának halmaza $Q = N^m$, az algoritmus egy állapota $q = (y_1, y_2, \ldots, y_m)$ és jelölje $\tau(y,t)$ az y page-re a t időpillanatig történt hivatkozások számát, v a t időpillanatig történt összes page hivatkozások számát.

Rendezzük az $y_1, y_2, \ldots, y_m$ page-ket ugy, hogy

$$\frac{\tau(y_i, t)}{v} \leqslant \frac{\tau(y_{i+1}, t)}{v}, \qquad i = 1,2, \ldots, m-1$$

Az algoritmus allokáló függvénye

$$g(S,q,x) = \begin{cases} (S, q') & \text{ha } x \in S \\ (S + x, q') & \text{ha } x \notin S, \ |S| < m \\ (S + x - y_1, q'') & \text{ha } x \notin S \ \ |S| = m, y_1 \in S \end{cases}$$

ahol

$$q'' = (x, y_2, \ldots, y_m)$$

és ha

$$y_k = x, \quad q' = (y_1, \ldots, y_{k+1}, y_k, \ldots, y_m)$$

1.5. **Definició.** Egy program egy $P = (N, U, u_0, f)$ négyes, ahol

N: a program page-k halmaza,
U: a program állapotok halmaza,
u_0: eleme U-nak, kezdeti program állapot,
f: állapot függvény, melyre $f: N \times U \longrightarrow U$

(Egy program állapot lehet például a memória aktuális tartalma.)

Jelölje $p(x,t) = P(r_t = x)$ annak a valószínüségét, hogy a t-edik időpillanatban az x-page-ra történt hivatkozás, $x \epsilon N$ és $\omega = r_1, r_2, \ldots, r_T$.

Egy program l-ed rendü, ha $|U| = l + 1$.

Jelöljön $<$ N fölött egy bináris relációt ugy, hogy ha

$$x < y \Rightarrow p(x,t) \leqslant p(y,t), t > 0.$$

$x \leqslant y$ jelenti, hogy $x < y$ vagy $x = y$.
$s = \min S$ olyan S-beli elem, melyre $s \leqslant x$ bármely $x \epsilon S$.

1.2.**Tétel.** *Ha egy P program 0-ad rendü, és értelmezve van N fölött egy $<$ bináris reláció, akkor az A optimális page-elő algoritmusnak a következő g: $\mathfrak{A}_m \times N \longrightarrow \mathfrak{A}_m$ allókáló függvénye van*

$$g(S,x) = \begin{cases} S & \text{ha } x \epsilon S \\ S + x - s, & \text{ha } x \notin S \end{cases} \quad \text{ahol}$$

$s = \min S$ *és* $|S| = m$.

Bizonyitás [1]-ben.

A tételben szereplő allokáló függvény azonos az önjavitó algoritmus allokáló függvényével, vagyis az önjavitó algoritmus optimális.

II. Az önjavitó algoritmus egy valószinüségelméleti közelitése

Legyen m a belső memória page-inek száma. Osszuk fel valamely adott programot $y_1, y_2, \ldots, y_m, \ldots, y_{m+n}$ page-ekre. Tegyük fel, hogy a page-ekhez rendre az adott programra vonatkozó hivatkozási valószinüségek

$$p_1 > p_2 > \ldots > p_m > \ldots p_{m+n} > 0$$

sorozata tartozik. Tegyük fel továbbá, hogy az egyes page-ekre történő egymásutáni hivatkozások függetlenek.

Legyen v a page–hivatkozások száma, ebből történjen

v_1 - szer az y_1-re
v_2 - szer az y_1-re
.
.
.
v_{m+n} - szer az y_{m+n}-re

hivatkozás, $\sum_{j=1}^{m+n} v_j = v$

Megjegyzés. Amennyiben a valószinüségeknek a sorozata egy rendezést követ, ugy a hozzájuk tartozó relativ gyakoriságoknak a sorozata is elegendő nagy M-re közel 1 valószinüséggel ugyanazt a rendezést követi.

Ezt a következő meggondolások alapján lehet belátni:

Legyen a valószinüségek sorozata

$$p_1 > p_2 > p_m > p_{m+1}, \ldots > p_{m+n}$$

A nagy számok gyenge törvénye értelmében

$$\forall \epsilon > 0, \delta > 0 \; \exists N_0^{(k)}, \text{ hogy } N_0^{(k)} \leqslant N\text{-re}$$

$$P\left(\left| \frac{\nu_k}{\nu} - p_k \right| < \epsilon \right) > 1 - \delta \qquad k = 1, 2, \ldots, m + n$$

Jelölje

$$p_1 - p_2 = \Delta_1$$
$$p_2 - p_3 = \Delta_2$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$p_{m+n-1} - p_{m+n} = \Delta_{m+n-1}$$

$\Delta_i > 0$, $i = 1,2, \ldots, m + n - 1$ a feltétel szerint és jelölje

$$\min_{i=1, \ldots, m+n-1} \Delta_i = \Delta, \qquad \text{és} \qquad \max_{i=1, \ldots, m+n-1} N_0^{(i)} = N_0$$

és válasszuk

$$\epsilon < \frac{\Delta}{2}$$

Igy $\epsilon > 0, \delta > 0 \; \exists N_0$, hogy $N_0 \leqslant N$-re

$$P\left(\left| \frac{\nu_k}{\nu} - p_k \right| < \epsilon, \; k = 1,2, \ldots, m + n \right) > 1 - \delta$$

A $(p_k - \epsilon, p_k + \epsilon)$ $(k = 1, \ldots, m + n)$ diszjunkt intervallumok, ezért

$$\lim_{\nu \to \infty} P\left(\frac{\nu_1}{\nu} > \frac{\nu_2}{\nu} > \ldots > \frac{\nu_{m+n}}{\nu} \right) = 1$$

Feltételezhetjük, hogy page-eknek pozitiv hivatkozási valószinüségük van, ami azt jelenti, hogy az y_i page előbb-utóbb bekerül a memóriába. De a következő cserénél, mivel az algoritmus a legkisebb relativ gyakoriságu page-t viszi ki, a megjegyzés szerint az y_i-nél kisebb valószinü-

ségü page kerül kivitelre, amennyiben van ilyen. Ha a page hivatkozások száma elég nagy, ez azt eredményezi, hogy a memóriában közel 1 valószinüséggel a legnagyobb valószinüségü $y_1, y_2, \ldots, y_{m-1}$ page-k maradnak benn, mig az y_m page bonyolitja le a cseréket, vagyis $p_1 > p_2 \cdots, > p_m > \cdots, p_{m+n}$ valószinüségnek megfelelő $y_1, y_2, \ldots, y_m, \ldots y_{m+n}$ rendezés alakul ki.

1.3.**Tétel.** *Az önjavitó algoritmus alkalmazása esetén az interruptok számának várható értéke aszimptotikusan minimális.*

Bizonyitás. Legyen μ_j egy karakterisztikus valószinüségü valtozó

$$\mu_j = \begin{cases} 1, \text{ ha a } j\text{-edik hivatkozás cserére vezet} \\ 0 \text{ egyébként} \end{cases}$$

Jelölje A azt az eseményt, hogy a rendezésben v_i pozíciója a j-edik lépésben nagyobb mint m. Rögzitsük $N_0 < k$-t. $k < N$-re, az interruptok számának várható értéke

$$\frac{1}{\nu}(E \sum_{j=1}^{n} \mu_j) = \frac{1}{\nu}\left[\sum_{j=1}^{N} \sum_{i=1}^{m+n} p_i P(A) \right] =$$

$$= \frac{1}{\nu}\left[\sum_{j=1}^{k} \sum_{i=1}^{m+n} p_i P(A) + \sum_{i=k}^{N} \sum_{i=1}^{m} p_i P(A) + \right.$$

$$\left. + \sum_{j=k}^{N} \sum_{i=m+1}^{m+n} p_i P(A) \right]$$

A $P(A)$ valószinüségek a megjegyzés szerint $i \leqslant m$ esetén 0-hoz, $i > m$ esetén 1-hez tartanak, igy az interruptok várható száma:

$$\lim_{\nu \to \infty} \frac{\nu - k}{\nu} \sum_{i=m+1}^{m+n} p_i \longrightarrow p_{m+1} + \cdots + p_{m+n}$$

Ez az összeg minimális, mert ha bármely olyan algoritmussal is dolgozunk, amelyik nem a legkisebb relativ gyakoriságu page-t cseréli ki a memóriából, vagyis nem önjavitó algoritmus, ez az érték csak nöhet.

A tétel igaz akkor is, ha nem tételezzük fel, hogy a $p_1, p_2, \ldots, p_{m+n}$ egymástól különbözőek. Ebben az esetben az algoritmus a megegyező valószinüségü page-k közül tetszés szerint jelöl ki egyet cserére.

Az eddigiekben feltételezzük, hogy a page-eknek egymásra való hivatkozásai függetlenek. Természetes módon merül fel a kérdés mit mondhatunk akkor, ha a függetlenséget függőségi megszoritásokkal helyettesitjük. Ilyen kérdésekkel foglalkoznak G. Ingargiola és J.F. Korsch [4] cikkükben.

Az előzőekben tárgyalt önjavitó algoritmus feltételezete azt, hogy a page hivatkozások száma egy nagy N számnál nagyobb. Felvetődik a kérdés, hogy milyen algoritmust alkalmazzunk abban az esetben, ha a page hivatkozások száma véges. Ezzel a kérdéssel nem azonos, de hozzá hasonló a "two-armed bandit" prébléma, amelyre megadott algoritmus véges vagy megszámlálhatóan végtelen nagy N-re is optimális. A "two-armed bandit" problémához hasonlóan a page problémát 3 page-re pl. a következőképpen fogalmazhatjuk meg:

Jelölje $N = \{1,2,3,\}$ a programot alkotó page-eket, $M = \{1,2\}$ a memória fizikai page-eit, és a page-eknek egymásra való hivatkozásainak a száma legyen véges. A feladat olyan algoritmus megadása, amely alkalmazása esetén a program a lehető legkevesebb interrupt-tal fut le.

Az igy megfogalmazott feladat nem minden algoritmus esetén azonos a "több pisztolyos bandita" próblémával sem. A 3-as számú page kint-tartozkodása véletlenszerű és ugyanakkor függ a másik két lehetséges page -párra vonatkozó stratégiától.

Irodalom

[1] Aho, Alfred V. – Denning, Peter J. – Ullman, Jeffery D.: Principles of Optimal Page Replacement. Journal of the Association for Computing Machinery. (1971), Vol. 18, No. 1. 80-93 p.

[2] Denning, P.J.: Virtual Memory. Computing Surveys. Vol. 2. No. 3. (1970), 153-189 p.

[3] Parmelee, R. P. etc.: Virtual Storage and Virtual Machine Concepts. IBM Systems Journal. 1972. No. 2. 99-130 p.

[4] Ingargiola, G. – Korsh, J. F.: Finding Optimal Demand Paging Algorithms. Journal of the Association for Computing Machinery. Vol. 21. No. 1. 1974. 40-53 p.

[5] Joseph, M.: An Analysis of Paging and Program Behaviour. Computer Journal. Vol.13. No.1. 1970. 48-54 p.

[6] F.R. Gantmacher: Matrizen Rechnung. II. Berlin, (1959)., 78 p.

Summary

An algebraic and a probabilistic model of the virtual memory

T. Gyires

In this paper the author gives a survey about new results of theoratical investigations connected with paging virtual memory.

In the probabilistic model the optimality of the self-correcting algorithm is proved in the case of independent reference string.

At last a problem is raised concerning with finite reference string.

Р е з ю м е

Алгебраическая и стохастическая модели виртуальной памяти

Т. Дирош

В данной работе афтор даёт польный обзор о современном состоянии теоретических исследований связанных с системами странимной (раде) виртуальной памяти.

В вероятностной модели в случае независимого ряда ссылок афтор доказивает оптимальность самоусовершенствующегося алгоритма.

В конце афтор указывает на проблему возникающую в случае конечного ряда ссылок.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

NAGY RENDSZEREK SOFTWARE PROBLÉMÁI

Benczúr András

MTA Számitástechnikai és Automatizálási Kutató Intézete

Nem általában nagy rendszerekkel, hanem egy konkrét információs rendszer megvalósitása során szerzett tapasztalatokkal szeretnék foglalkozni. Elsősorban az alapsoftware rendszerrel összefüggő problémákat részletezem.

A rendszer kialakitásának előzményei

A Dunai Vasmű 1974-ban kapott egy UT-200-as terminált (display, kártyaolvasó, sornyomtató és lyukszalagolvasó konfigurációval) a SzTAKI Budapesten, a Várban lévő CDC-3300-as számológépéhez. Ezzel egyidőben a gép konfigurációja is bővült, a felhasználható memória 30K szóval nőtt, a szabad on–line disk kapacitás 2 kis diskről (8M character) 6 kis és 2 nagy (32M character) diskre nőtt.

Azt a feladatot kaptuk, hogy alakitsuk át a Vasmű egyik – a CDC-n lévő – nyilvántartási rendszerét terminálon keresztüli használatra. Az erre a célra kiválasztott rendszer a termelésirányitás alapadatait, a rendelkezéseket, a szerződéseket, a termelési és forgalmi bizonylatokat tartja nyilván.

A teljes rendelésállomány kb. 10 ezer tételt, a szerződésállomány 25-30 ezer tételt jelent, a havi termelési, szállitási adatok száma pedig 30-40 ezer között mozog. Naponta körülbelül kétezer uj adatot kell bevinni a rendszerbe. Az 1971-ben elkészült első változatot 1974-ben átalakitottuk terminálon keresztüli használatra, ennek során természetesen változásokra és uj lehetőségek bevezetésére is sor került. Az uj rendszerben megtartottuk a régi szolgáltatásokat is, amelyek általában az állományok különböző sorrendben és összesitési csoportositásban történő kiiratását jelentik. Ezeket a listákat elszámoláshoz, utólagos ellenőrzéshez, kimutatásokhoz, havi tervek összeállitásához és egyéb adminisztrációs célokra használják. A kialakult szokások és igények szerint ezekkel a listákkal hetente a teljes állományunk 2-3-szor kiiratódik, természetesen különféle bontásokban. Ez a munka terminálon keresztül nem végezhető el, a nagytömegű kiiratások továbbra is a Várban készülnek. A terminálos kiiratáshoz új rendszert dolgotunk ki, indextáblázatok segitségével lehetővé tettük az adatok különböző paraméterek szerinti gyors kiválogatását és igy az adatállomány lekérdezés jellegű kiiratását, másrészt táblázatos és csak összesitéses formájú kiiratásokkal rövid és tömör információ szolgáltatását. Ilyen módon lehetőség van a napi termelés irányitásához operativ adatszolgáltatásra. Ennek feltétele viszont az, hogy az adatok aznapi állapotot tükrözzék, tehát az adatok keletkezése és beillesztése között egy napnál rövidebb idő teljék el. Ezért az adatbevitelt – kártyáról és lyukszalagról – terminálról, naponta két alkalommal célszerű végezni. Egy–egy félnapi adatbevitelt compute időben nem lehet többet fél óránál, de a lassú adatátvitel, sok közvetlen disk–hozzáférés miatt a programok lefutási ideje ennek 2–3-szorosa.

A terminál üzembeállitásától a DV felhasználói többletszolgáltatást vártak. A leglényegesebb elvárást az adatrendszer és a valós adatok közötti időeltolódás minimálisra csökkentése, a napi termelésirányitáshoz operativ adatszolgáltatás lehetővé tétele jelentette. A gépnél kialakult munkarend, valamint a gép jó kihasználása viszont azt igényelte, hogy az új rendszer ne álljon túl hosszú, túl nagy igényü JOB-okból, kizárólagos géphasználatra lassú, a központi egységet rosszul kihasználó adatfeldolgozó rendszert ne tervezzünk.

A meglévő, régi rendszert kiértékelése, statisztikai elemzése hasznos szempontokat adott az új rendszer tervezéséhez. Melyek voltak a régi rendszer fő jellemzői:

1. Nagyméretű, szekvenciálisan kezelt, rendezett alap file-ok
2. Hosszú, kizárólagos géphasználatra épitett, adattipusonkénti felújitó programok.
3. Rendszeres, felújitásonkénti dump, szalagkezelés miatt bizonytalan helyreállitás.
4. Kiiratás, felújitás mindig a teljes állomány kezelését igényelte; újrainditás hosszú futások megismétlését jelentette.

Az alapvető átszervezést az adatstruktúrában a független részekre bontás (a 9 fő üzemegység adatai 9 független rendszerbe kerültek) és a gyors válaszadás biztositásához közvetlen file-szervezési módszerek alkalmazása felújitás közben hozzáférési lehetőségek előkészitése jelentette.

A rendszer részletes tervének kiépitéséhez erősen figyelembe kellett venni a terminál kezelő software lehetőségeit és a gép terheltségét. A tervezés időszakában a CDC által szolgáltatott [3] RESPOND/MASTER operációs rendszerrel kellett számolnunk. Nyilvánvaló volt, hogy a RESPOND-file-okon keresztül történő közvetlen terminálhasználat nem vehető figyelembe, a RESPOND remote–batch job futtatásai lehetőségére kellett a rendszert alapozni. Mivel a RESPOND-nak mind a memória lekötése, mind a saját ideje nagy, RESPOND alatt csak a legminimálisabbat, az adatbeolvasást és lista lekérést végezzük. A felújitás és a válasz listák összeállitása RESPOND nélkül, minimális géptermi beavatkozási igénnyel fusson, és a rendszer, mint nagy csatornaidejü, lassú és 30-50% memórialekötést jelentő háttérprogram dolgozzon két RESPOND idő között. Igy mind a felújitás, mind a lekérdezés három fázisra bomlik:

FELUJITÁSI RENDSZER

1. Adatbevitel.	RESPOND alatt, terminálról, permanens gyüjtő file-ra.
2. Felújitás	RESPOND nélkül, felújitási dokumentum és ellenőrzési lista permanens file-ra.
3. Felújitási lista lekérése:	RESPOND alatt, terminálra a felújitási lista file-ról.

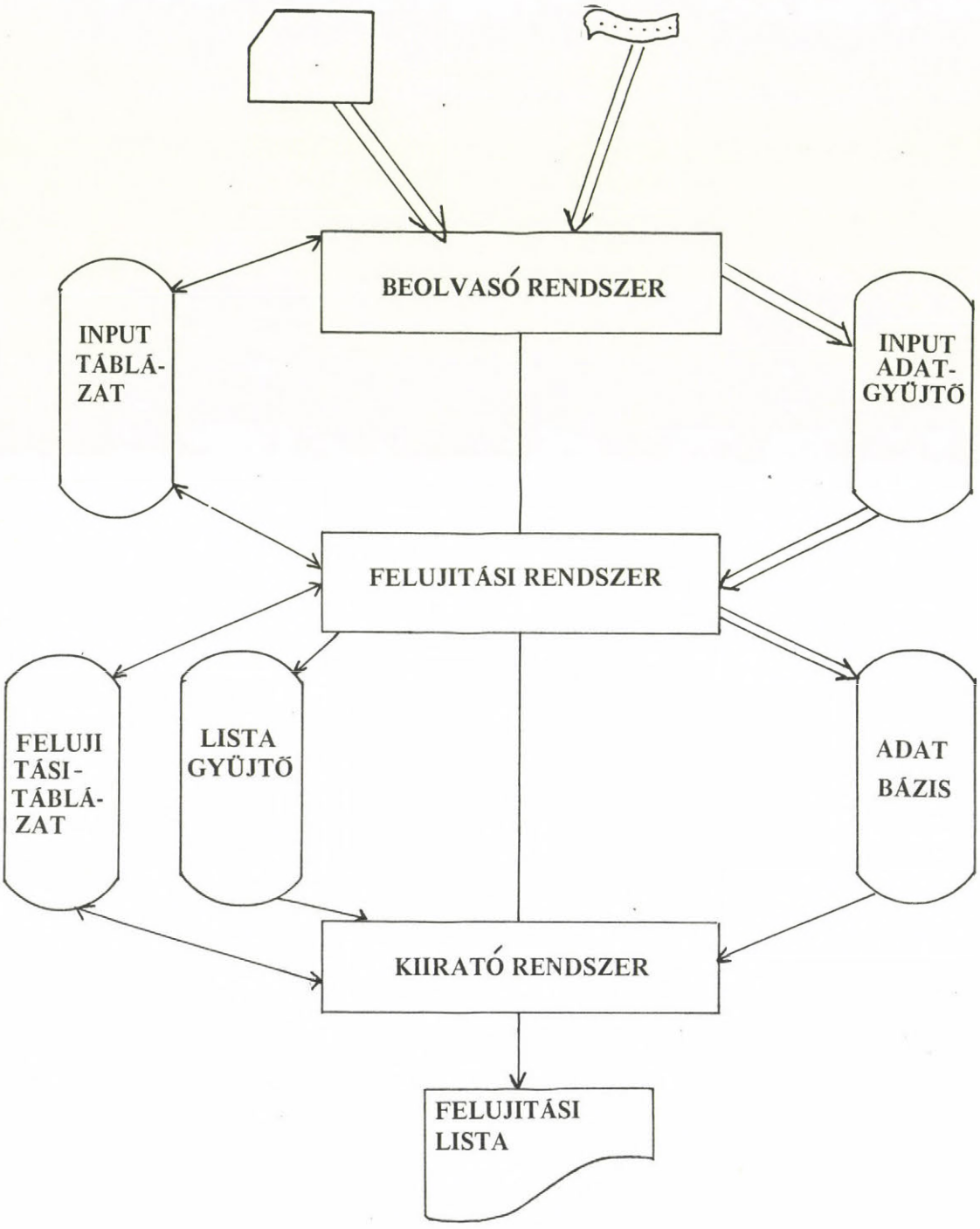
KIIRATÁSI RENDSZER

1. Igény adatkártyák bevitele.	RESPOND alatt, terminálról, permanens gyüjtő file-ra.
2. Lista készités.	RESPOND nélkül, lista permanens file-ra.
3. Lista lekérés.	RESPOND alatt, terminálra, a lista file-ról.

(A listázás a RESPOND által kezelt OUT-file-on keresztül csak a RESPOND alatt futó JOB-okra használható, ezért kellett a lista gyüjtő file-okat közbeiktatni.)

A rendszer kidolgozása közben az intézet Software Osztályán új terminálkezelő software-t fejlesztettek ki, a LOTUS szub-operációs rendszert [4],[5] amely kis memória és időigénnyel megoldja a JOB-ok terminálon való kötegelt futtatását. Ezzel az adatbeolvasás és listázás leválasztása a feldolgozástól szükségtelenné vált, ezt a LOTUS rendszer az IMPORT és EXPORT parancsokkal automatikusan biztositja. A kiiratási rendszer valóban a választ biztositó programok JOB-ként közvetlen futtatását jelenti. A felujitási rendszer szerkezete azonban nem változott. Ez a MASTER operációs rendszer egyéb sajátosságaiból következik, melyeket a felújitás biztonságának, automatikus újraindithatóságának megoldása miatt kellett figyelembe vennünk. Ezen kivül az adatbeolvasás leválasztását a kizárólag közvetlen üzemmódban használható terminál lyukszalagolvasó elkerülhetetlenné tette.

A felújitási rendszer vázlaltát az 1. ábra szemlélteti.


BEOLVASÓ RENDSZER
INPUT TÁBLÁ-ZAT
INPUT ADAT-GYÜJTŐ
FELUJITÁSI RENDSZER
FELUJI TÁSI-TÁBLÁ-ZAT
LISTA GYÜJTŐ
ADAT BÁZIS
KIIRATÓ RENDSZER
FELUJITÁSI LISTA

1. ábra

A BEOLVASÓ RENDSZER a terminálról kártya, vagy lyukszalag formában beolvasott adatokat üzemág és tipus szerint szétválogatja, formailag ellenőrzi és az ADAT GYÜJTŐ file-ok megfelelő szegmenseiben halmozva gyüjti. Az INPUT TÁBLÁZAT őrzi a felújitási rendszer futásaiként az egyes szegmensekből feldolgozott adatok mutatóit, és mutatja a következő felujitás során feldolgozandó adatokat. A rendszer ellenőrzi a szegmensek telitettségét, lehetőséget biztosit túlcsordulásra, de túlcsordulás esetén kezdeményezi az összegyüjtött és feldolgozott adatok mágnesszalagra üritését.

A FEJUITÁSI RENDSZER teljesen zártan, egyetlen JOB futtatásával történik. A nagy számú (40-50) feldolgozó program összefüzéséhez a MASTER operációs rendszernek a COBOL nyelvbe is beépitett TASK-hivási lehetőségét használtuk fel. Az összeállitás közben számos problémát kellett tisztáznunk, amelyek a MASTER és a COBOL file kezelésében lévő eltérések miatt adódtak és csak COBOL taskok hivása közben jelentkeztek.
A rendszer zárttá tételével azt kivántuk elérni, hogy a nagy számú feldolgozó program adminisztrálását, a helyes sorrendek betartását automatizáljuk. Természetesen ilyen – a gép méreteihez képest – nagy rendszer futásánál fel kell készülni a futás megszakadására, és az újrainditás lehetővé tételére. Az újrainditathatóságot úgy kellett megszerveznünk, hogy a megszakadásig végzett munka a lehető legkisebb mértékben vesszen kárba. Ezért a programok hivását vezérlő program (TASK) a FELUJITÁSI TÁBLÁZAT file-on regisztrálja a hivás kezdetét és a futás hibátlan megtörténtét. Igy bármely pillanatban történjen is a megszakadás, a vezérlő program újrainduláskor ellenőrizni tudja, mely programok futottak le és melyik program futása szakadt félbe. Ahhoz, hogy a felújitást a félbeszakadt program folytatásával, újrainditásával folytatni lehessen, minden egyes programhoz ki kellett dolgozni az újrainditás lehetőségét. A sok közvetlen szervezésü és egymással bonyolult kapcsolatban lévő file módositása miatt a felújitó programok nem voltak újrainditathatók, hiszen a logikai és fizikai output szétválása miatt a változások fele részben már kiiródtak a disk-re, fele részben a memóriában voltak a program megszakadásakor. Ugyancsak próblémát okozott újrainditáskor a JOB által használt disk-munkaterületek (SCRATCH file-ok) elvesztése az egyik leggyakoribb hiba, az operációs rendszer fennakadása esetén. Ilyen munkaterületen gyüjti a MASTER operációs rendszer a sornyomtatóra irt adatokat is, és a JOB lefutása után nyomtatja ki onnan. Ennek használata esetén fennakadás előtt hibátlanul lefutott programok listái elvesztek volna, s emiatt az egész futást elölről kellett volna kezdeni. Ez volt az egyik oka annak, hogy a felújitó programok LOTUS alatt is a LISTA-GYÜJTŐ file-ra irják az adatbeillesztés dokumentációját. A másik oka ennek az, hogy ellenőrzéshez ezeket az adatokat a feldolgozásétól eltérő sorrendben, esetleg több rendezésben használják. A programok újrainditásának megoldásához elsősorban a program által módositott file-ok helyreállitási lehetőségét vizsgáltuk meg. A szekvenciálisan, folytatólagosan feltöltődő file-ok esetén elég a kezdő cim megjegyzése. Ezeket a címeket szintén a FELUJITÁSI TÁBLÁZAT tartalmazza.
A közvetlen módon, sokféle hozzáférési módban kezelt file-ok és a hozzájuk segéd file-ok helyreállithatóságát úgy biztositottuk, hogy az alapfile helyreállitása során könnyen konstuálható javitó eljárás.

Megjegyzés: A biztonsági gyüjtő file számos más célt is kielégit. Archiválásra a törölt adatok innen kerülnek, változási statisztikák készülnek róla, a lekérdező rendszer több táblázata ennek alapján követi a változásokat.

A felújitási rendszer vezérlő programja az INPUT TÁBLÁZAT és a FELUJITÁSI TÁBLÁZAT alapján minden tekintetben szervezni tudja a programok futását. Kijelöli számukra a beillesztendő adatokat és a gyüjtő file-ok kezdő címeit, regisztrálja futásukat, félbeszakadás esetén hivja a helyreállitó programot és folytatja feldolgozását. Ezen kivül az allokált file-méretek, a file telitettségének és a feldolgozandó adatok számának ismeretében előre jelzi a várható túlcsordulásokat, s azok igy nem okozzák a futás megszakadását.

A KIIRATÓ RENDSZER feladata, hogy a LISTA GYÜJTŐ file tartalmát megfelelő bontásban és sorrendben kiirassa. Ezen kivül a biztonsági gyüjtő file alapján is készit változási kimutatásokat. A listázó programok a FELUJITÁSI TÁBLÁZAT-ból kapják az egyes programokhoz tartozó lista és biztonsági rekordok kezdő és végcimeit.

Befejezésként röviden érintem a lekérdező rendszert, amely elsősorban file-szervezési és programozási problémák megoldását jelentette, s kevésbé kötödött az operációs rendszerhez. Egyedül azt szeretném hangsulyozni, hogy a LOTUS rendszer nélkül lényegesen körülményesebb lett volna egy jól hozzáférhető rendszert megoldani. A LOTUS-szal tulajdonképpen JOB-szinten interaktiv kérdező rendszert lehetett kialakitani, hiszen a LOTUS a JOB-ok futtatását interaktiv módon végzi. Elvileg semmi különbség nincs a között, hogy egy 30-120 másodperces futást igénylő válaszadás programja JOB vezérkártyával kerül be a rendszerbe, vagy pedig egy program interaktiv módon beolvassa a kérés adatait, s utána hív egy 30-120 másodpercig futó feldolgozó programot.

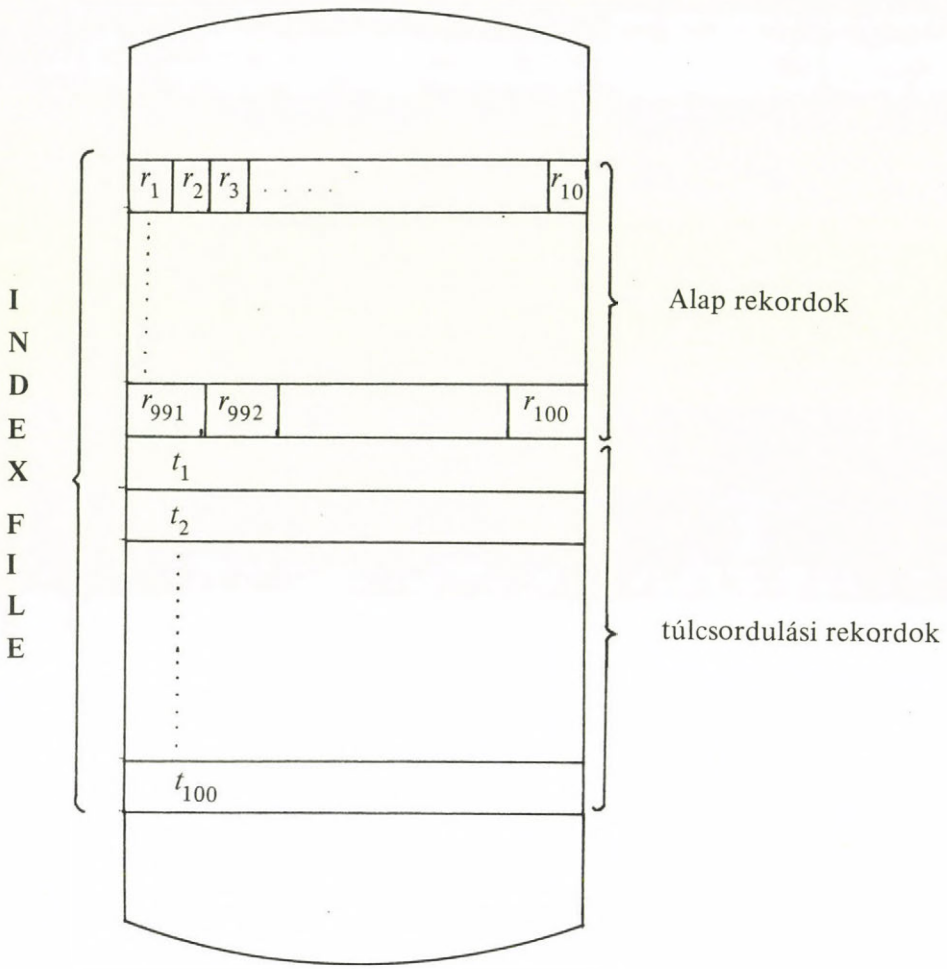
A kérdésre a válaszadás három lépésben történik, mindegyikre külön–külön paraméteres feltétel adható meg. Az első lépés az adatok kiválogatása az előirt paraméterek szerint. A második lépés a kiiratáshoz szükséges rendezés végrehajtása, szintén a kérésnek megfelelően. A harmadik lépés a kivánt formátum szerint az összesitések, táblázatok megszerkesztése és kiiratása.
Az első lépésnek két alaptipusa van:

a) egy–egy üzem teljes adatállományát át kell nézni a kivánt adatok kiválogatása végett
b) indextáblázatokon keresztül közvetlenül elérhetők az elsődleges feltételnek eleget tevő adatok.

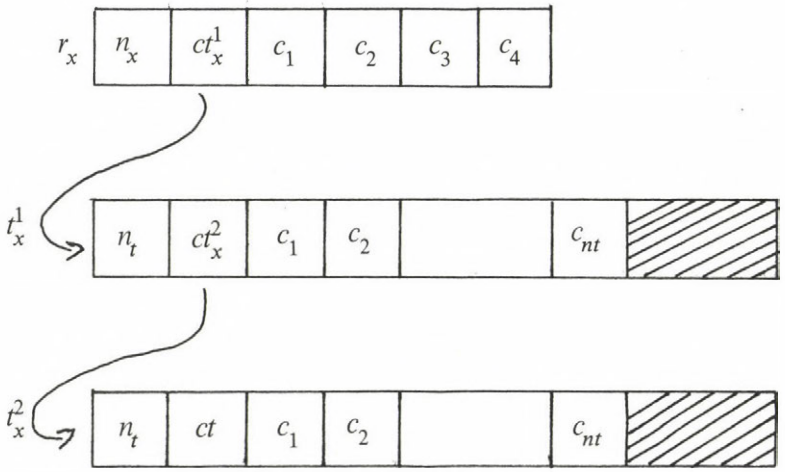
Az a) esetben a válogatás és a rendezés egy lépésben történik. A válógató algoritmus egyenként ellenőrzi a rekordokat, egyszerre akár 5 kivánság szerint, s amennyiben egy rekord valamelyik kivánságnak eleget tesz, átadja a megfelelő rendező eljárásnak. A rendező eljárás leválasztja a rendezési szempontot képező kulcsmezőket az adtrekordokról, kiegészíti az igy kapott adatsort a rekord tárolási cimével, s az igy kapott (cim, kulcsmezők) rekordokat rendezi a kulcsmezők szerint. A rendezés végeredményeként kapott címsorozatot permanens tárolóra irja. A harmadik lépés kiirható eljárásai ennek a cimsorozatnak megfelelő sorrendben olvassák az adatrekor-

dokat és készitik a listát.

A b) esetben a válogatást már az adatrendszer felújitása előkésziti. A leggyakrabban kérdezett paraméterek (pl. rendelő, vállalat, a termék acélminősége, mérete, stb) szerint indextáblázatokatban elérhetők egy adott paraméterértékhez tartozó adatok tárolási cimei. Igy az elsődleges válogatás gyorsan, s teljes állomány átnézése nélkül elvégezhető, a további müveletek más igen kevés adatot mozgatnak. Az indextáblázat szerkezetét a 2. ábra mutatja. Az alaprekordok a paraméterértéknek megfelelő sorszámú rekord poziciót foglalják el az index-file-ben. A paraméterértékek kódoltak, értékkészletük 200-1000 között változik. Az előforduló értékek a teljes készlet 30-50% -át teszik ki, az egy értékhez tartozó adatrekordok számának szórása igen nagy. Ezért választottunk kis alaprekordot és hosszú "tulcsordulási" rekordot a nagy előfordulási értékekhez.



A rekordok szerkezete



2. ábra

ahol

r_x: az x paraméterértékhez tartozó alaprekord, tárolási sorszáma is x.
n_x: az előfordulások száma.
ct_x: az első túlcsordulási blokk cime. Tartalma 0, ha $n_x \leqslant 4$.
$c_1, \ldots, c_4$: előfordulások cimei, csak $n_x \leqslant 4$ esetén bir értékkel az első n_x, egyébként értékük 0.
t_x^i: túlcsordulási blokk.
n_t: a túlcsordulási blokkban található cimek száma, $n_t \leqslant 59$.
ct: a következő túlcsordulási blokk cime. A lánc utolsó blokkjában értéke 0.
c_i: cim–mezők, az első n_t értéke nem nulla, a többié nulla.

Az előfordulási cimek rendezettek, azaz egy blokkon (rekordon) belül $c_i < c_{i+1}$, $1 \leqslant i \leqslant n_t - 1$ esetén, mig blokkok között

$$c_{nt}\left(t_x^i\right) < c_1\left(t_x^{i+1}\right)$$

ha létezik a két túlcsordulási blokk.

A listának is több tipusa van:

A) Előre rögzitett, kontrol fokozatos összesitővel tételes felsorolást készitő lista. Üzemenként 3-5 féle létezik.
B) Paraméterekkel szabályozható, de A-nak megfelelő formáju kiiratás.
C) Paraméterekkel szabályozható, csak összesitő sorokat készitő listázás.
D) Táblázatos formájú, paraméterekkel szabályozott kiiratás.

Mindegyik listázási tipus használható az a) és b) előkészitési tipusok bármelyikével. Terminálos kérdezésre a b) előkészitéssel bármelyik kiiratás alkalmas – feltéve, hogy a kérdés eléggé behatárolt – a C) és D) kiiratások a) előkészitéssel is használhatók.

Irodalom

[1] CDC 3170-3300-3500 Computer Systems MASTER version 4.0. Reference Manual CDC 1973. Pub. no. 60415100

[2] ANSI COBOL version 2 MASTER/MSOS Reference Manual CDC 1969, 1970, 1971. Pub. no. 60281100

[3] RESPOND EXPORT–IMPORT/MASTER Reference Manual CDC 1968, 1969 Pub. no. 6020750

[4] LOTUS – Uj alapsoftware az akadémiai távállomás hálózathaz.Információ és Elektronika (sajtó alatt)

[5] LOTUS MTA SzTAKI CDC-3300 Felhasználói ismertetők 12. 1975.

Summary

Software problems of large systems
A. Benczur

The thema of the presentation is the implementation problems of a large data base system in connection with the basic software. The system belongs to the management information system of Danube Iron and Steel Works. The data base of this system is on the CDC-3300 computer of the HAS in Budapest, and it can be accessed through a UT-200 terminal. The processing of the data base is random batch processing with some query capabilities and the problem of its integrity is very close to that of the on-line processing.

There is described in detail, how a good terminal handling software can help to bring in accordance the user requirements to the data base and the possibilities of the computer system. In this respect we show the difference between the Control Data's RESPOND system and the LOTUS terminal handling sub-operating system developed by the Software Department of the Computing and Automation Institute.

Р е з ю м е

Проблемы больших вычислительных систем по математическим обеспечениям

Бенцур Андраш

Темой настоящего доклада является проблема осуществления большой базы данных с точки зрения использования возможностей данной оперативной системы. Система принадлежит к АСУ Дунайского Металлургического Комбината.

Банк данных хранится на ЭВМ СДС-3300 ВАН в Будапеште и для пользования им можно обращаться через терминал УТ-200.

Обработка данных производится в групповом режиме с некоторыми возможностями запроса и проблема целостности системы очень сходна с проблемой целостности системы, работающей в единовременном режиме.

Дается подробное описание о том, как хорошо организованное МО, управляющее терминалом, помогает согласовать потребности при пользовании и возможности ВС. С этой точки зрения мы сравниваем систему фирмы СДС "РЕСПОНД" и подоперационную систему "ЛОТУС", управляющую терминалом, которая была разработана сотрудниками отдела МО Института Автоматизации и Вычислительной Техники ВАН.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

ON THE "LOAD–STORE" –TYPE SOLUTIONS OF THE "MUTUAL EXCLUSION" –PROBLEM

Előd Knuth
Computer and Automation Institute of Hungarian Academy of Sciences

Since the semaphores had been introduced, we might believe solutions based only on "load"and "store" instructions have lost their importance. It is not quite true. The properties of the elements of these solutions take prominent parts in the general problems according to the executions of concurrent computer programs. Therefore it is reasonable to study these logical elements more.

1. Introduction

1.1. The usual presentation of the mutual exclusion problem

Consider N independet processes (programs) communicating with each other only via shared memory. Each process is a cyclic program with two parts – a *critical section* and a *noncritical section.* The problem is to write the programs so that the following conditions be satisfied:

1) At any time, at most one process may be in its critical section.
2) If there exist processes which have finished their noncritical section, one of them must be able to enter its critical section.
3) Each process must eventually be able to enter its critical section.

1.2 Assumptions on simultaneous operations

a) The effect of "simultaneous" assignement to a data variable d of the values d_1 and d_2 is to be *either* d_1 *or* d_2, but *not* a "mixture" of both.
b) If d is tested when $d = d_1$ and "simultaneously" d is set to d_2, the effect of the test is to be *either* the effect attributable to $d = d_1$, *or* the effect attributable to $d = d_2$, but *not both.*

2. Properties of some basic algorithms

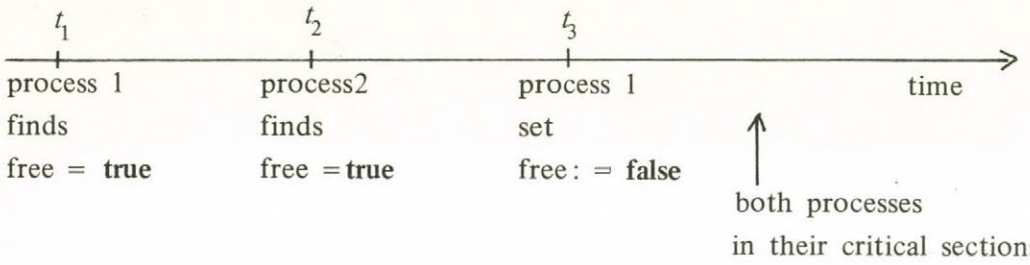
First consider the following trivial algorithm:

Common store: **boolean** free: (initially free = **true**)

There are N processes each of the following form and started in its noncritical section.

```
begin
L:  if not free then goto L;
    free: = false;
    critical section;
    free: = true;
    noncritical section;
    goto L;
end
```

It is easy to see that this system works in a quite unpredictable way. One possible realization may be the following:



We may realize this algorithm is one of the extremities:
It contradicts to conditions 1) and 3) but satisfies condition 2).

The following construction guarentees condition 1).
We use the SIMULA notations:

```
class process;
  begin boolean in critical;
      integer;
    L: in critical: = true;
      for i:=1 step 1until i − 1, i + 1 step 1 until N do
          if proc [i]. in critical then
            begin
              in critical: = false;
              goto L;
            end;
      critical section;
      in critical: = false;
      noncritical section;
      goto L;
  end;

ref (process) array proc [1 : N];
```

The starts of the processes are assumed in their noncritical sections. The compound tests of the processes examine weather there is another process for wich *in critical* = **true** too.

Now we prove that condition 1) holds for this system. Suppose that there are two processes in their critical sections at the same moment t. Let us denote them by $P1$ and $P2$. Let be t_1 and t_2 the moments when $P1$ and $P2$ last time finished their assignment at the label L respectively. Suppose for example $t_1 \leqslant t_2$. Then during the time interval $[t_2,t]$ the boolean flags of both processes have been true values, therefore both processes had to go into their critical sections before t_2. Let us denote this moment for $P2$ by t^*. Hence $t^* < t_2$, but this contradicts to the sequential nature of the process.

This algorithm is another extremity. It satisfies 1), but neither 2) nor 3). To demonstrate it consider a typical blocking situation:

t'	t''	t''_1	t''_2	t''_3	t'_1	t'_2	t'_3
P1 sets its flag true	P2 sets its flag true	P2 refused	P2 resets its flag	P2 serts its flag	P1 refused	P1 resets its flag	P1 sets its flag

forever

3. The so–called "after you blocking" of symmetric systems

Consider a system of N processes. Denote its common data by X and the local data of process P by $P.Y$.

Definition. We call a system of N processes *symmetric* if the local data of each of process has the same structure, each process has the same algorithm and every predicate of process P can be given in the form $F(P.Y,X)$. (i.e. it depends only on the state of the common and on the own local data.)

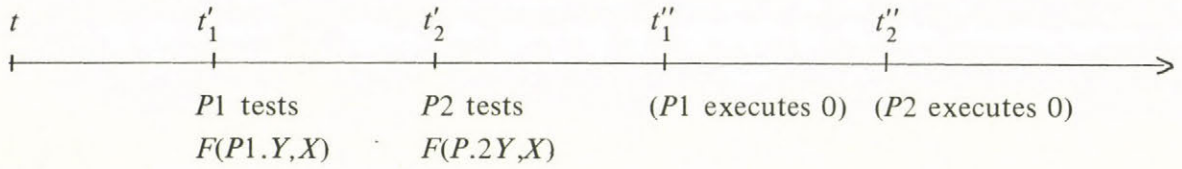
Now we give a simple proof to show that in any symmetric system the processes always can block each other. In other words, if one process can enter its critical section, other processes also will be able to enter at the same time.

Let us concentrate on the conditional statements of the processes. Suppose that the processes $P1$ and $P2$ left the first k conditional statements in the same way, i.e. to each predicate of $P1$ corresponds the same value in $P2$. Let t be a moment when both processes have been executed their k-th statement. According to our assumption $P1.Y = P2.Y$ holds at t.

Consider the $k + 1$ -th statement:

if $F(Y,X)$ **then** 0;

where 0 denotes some kind of operations. Now the executions may well be in the following arrangement:



Our assumption $P1.Y = P2.Y$ and the fact that X is also unchanged during the interval (t,t_1'') imply $F(P1.Y,X) = F(P2.Y,X)$, therefore processes $P1$ and $P2$ will continue again the same way.

Remark 1. The second example of the previous paragraph is also a symmetric one because we can express the compound condition by "there are at least two processes with flag = true".

Remark 2. If the processes cannot identify themselves in any way (e.g. using numbering), the system will necessarily be symmetric, therefore the problem of the mutual exclusion will be unsolvable.

4. Simplest algoritms satisfying only axioms 1) and 2)

One possibility is to solve the blocking situation of our second extreme algorithm in paragraph 2. E.W. Dijkstra found such a solution in his paper [1]. His proof of axiom 1) is rather informal but we can supplement it by the proof described in paragraph 2. to complete it.

To find a second possibility, first consider the following

4.1.**Definition.** Let **D** be a finite set of concurrent instances of the same cylic process P. We will say that L is a *closed program point* of P, if

a) Let t be a moment when an element of **D** reaches its own program point L. Then there exists a moment $t^* < t$ from which the set of processes $C(t^*)$ which are still able to reach their own points L too, already cannot increase.

b) At the moment t the elements of $\mathbf{C}(t^*)$ are explicitly known.

Instead of examining the difficulties of the construction we give an example.

common: **boolean** free; (initially: free = **true**)

process:

```
begin
    boolean flag;
    M:
        flag: = false;
        if not free then goto M;
        flag: = true;
        if not free then goto M;
        free: = false;
```

```
L:
        free: = true;
        flag: = false;
        goto M;
end
```

Now we prove that L is a closed point of the processes. Suppose that one process attained its program point L at the moment t. Then there exists a moment t^* when some process first finished the execution of the statement *free* : = **false.** Obviously $t^* < t$. Since the moment t^* the tests of the processes have been closed. After the moment t only those processes are able to reach their points L which have **true** value of their own flag.*

Remark. This system is a symmetric one, i.e. we have seen that symmetric systems may have closed points.

If we have a closed point it will be very simple to give an algorithm statisfying axioms 1) and 2). We can for example serve the processes according their orders. Suppose that L is a closed point of the processes, then

```
L:
  for i: = 1 step 1 until no – 1 do
     if process (i). flag then goto L;
  critical section;
```

where i is a local integer variable for each process and *no* is the order of the process. Now the closedness guarantees that no more than one process can enter its critical section because the test allowes only the minimal ordered process of *flag* = **true.** It is also easy to prove that one of the processes will surely enter its critical section.

Using these ideas we can produce the following complete algorithm:

* And of course which are executed their second test, but it is no matter for us.

```
common : integer free ; (initially: free = 0)
process (no); integer no;
    begin
        boolean flag;
        integer i;
        M:
                flag: = false;
                if free < 0 then goto M;
                flag: = true;
                if free < 0 then goto M;
                free: = free – 1;
        L:
                for i: = 1 step 1 until no –1 do
                    if process (i). flag then goto L;
        critical section;
            free: = free + 1;
            flag: = false;
        noncritical section;
        goto M;
    end;
```

(We remark that the boolean form of the variable *free* is not sufficient because of the necessity of the correct regeneration of its value in the right time.)

5. Further remarks and problems

The general case i.e. when axiom 3) is satisfied too is not very interesting from the point of view of our investigations because these algorithms may have been built on the same bases The first solution is made by Knuth [2] and one can find some more efficient versions of it in the works of Bruijn [3] Finally I refer to Lamport [5] who found a very interesting and elegant new approach but theoretically it has an unsolved point.

We can say that axioms 1) and 2) are related to the logical possibilities and axiom 3) to some kind of scheduling disciplince. Let us concentrate now only on axioms 1) and 2).

I propose the following problem: what are the common properties of all the possible solutions of the mutual exclusion problem. Is it true that each solution has a closed point?

I believe the answeres to these questions would be useful not only for the mutual exclusion problem but even for understanding the general structures of concurrent processes.

References

[1] Dijkstra,E.W. Solution of a problem in concurent programming control. *Comm.* ACM 8,9 (1965), 569.

[2] Knuth, D.E. Additional comments on a problem in concurrent programming control. *Comm.* ACM 9,5 (1966), 321–322.

[3] de Bruijn, N.G. Additional comments on a problem in concurrent programming control. *Comm.* ACM 10, 3 (1967), 137–138.

[4] Eisenberg, M.A., McGuire, M.R. Further comments on Dijkstra's concurrent programming control problem. *Comm.* ACM 15,11 (1972), 999.

[5] Lamport, L. A new solution of Dijkstra's concurrent programming problem. *Comm.* ACM 17,8 (1974), 453–455.

Összefoglaló

A kölcsönös kizárási probléma elemi operációkkal történő megoldásairól

Knuth E.

A cikkben a kölcsönös kizárási próbléma u.n. "software" – megoldásainak tulajdonságairól van szó. Néhány új egyszerű összefüggést bizonyitunk és bevezetést nyújtunk a próbléma-kör általános tulajdonságainak felderitéséhez.

Р е з ю м е

О проблеме взаимного исключения

Э. Кнут

В работе рассматриваются характеристики решений типа "load-store" проблемы взаимного исключения на основе трех аксиом Dijkstra.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

KÖZPONTI EGYSÉG KIHASZNÁLTSÁGÁNAK SZIMULÁCIOS VIZSGÁLATA MÁSODRENDŰ AUTOREGRESSZIÓS FOLYAMATOK ALKALMAZÁSÁVAL

Knuth Előd

MTA Számitástechnikai és Automatizálási Kutató Intézete

1. A központi egység ütemezésének problémája (CPU scheduling)

Az alábbiakban a feladatot csak főbb vonásaiban mutatom be, tekintettel arra, hogy a részletes leirás magyar nyelven is több helyen megtalálható, ld. pl. Arató [1], Tomkó [2].

Feltesszük, hogy egy multiprogramozott számitógép egyetlen központi egységgel rendelkezik és egyidejüleg N számú *"job"* lehet aktiv állapotban. (Az *egyszerűség kedvéért* ez utóbbi alatt azt értjük, hogy egyidejüleg teljes egészükben be vannak töltve a központi memóriába.)

Feltesszük, hogy egy job u.n. "CPU–igények"és "I/0 *igények*" váltakozó sorozata. A központi egységet igénylő job-ok a *"CPU–sor"*-ban várakoznak kiszolgálásukban az I/0-igényekről azonban most feltételezni fogjuk, hogy kiszolgálásuk soha sem ütközik akadályba.

Az u.n. *"CPU–scheduling"* – probléma mármost a következőt jelenti: Ismerve a sorbanálló igények CPU – igényeinek időtartamát, vagy korábbi igényei időtartamainak statisztikai jellemzőit, eldöntendő, hogy adott pillanatban melyik job-ot szolgáljuk ki úgy, hogy a központi egység kihasználtsága maximális legyen.

A feladat analitikus úton történő megközelitése csak akkor lehetséges, ha a folyamatok leirására egyszerübb valószinüségi feltételezésekkel élünk. Fontos eredményeket találunk erre vonatkozóan Arató [1] és Tomkó [2] dolgozataiban.

A következőkben egy olyan valószinűségi modellről lesz szó, mely a valóságos helyzetet az eddigieknél jobban megközeliti.

2. A folyamatok valószinüségi leirása

Jelölje t egy adott job belső (elapsed) idejét.
Legyen

$$p(t) = \left\{ \begin{matrix} 1 \text{ ha a job a CPU-t} \\ 0 \text{ ha a job I/0-t} \end{matrix} \right\} \text{ igényel a } t \text{ pillanatban.}$$

Legyen az S konstans olyan nagy, hogy a

$$\overline{p}\,(kS) = \int_{(k-1)S}^{kS} p(\tau)d\tau$$

függvény közelitőleg normális eloszlású.

A CPU és I/0 periodusok váltakozásának jellegét a $\overline{p}\ (kS)$ függvény segítségével fogjuk leirni. Ennek alapgondolata az, hogy az intervallumok hosszának eloszlása a *valóságban nem homogén*, hanem periodikusan változó tendenciák figyelhetők meg bennük. Mint ismeretes, ilyen változások valószinüségi leirásának legkézenfekvőbb módja a másodrendű autóregressziós folyamatokkal történő közelités. Ezt a következőképpen adhatjuk meg:

Legyen $p^* = \overline{p} - E\overline{p}$, és tegyük fel, hogy p^* kielégiti az alábbi differencia egyenletet:

$$p^*(kS) + \xi p^*((k-1)S) + \eta p^*((k-2)S) = \epsilon(k)$$

ahol $\epsilon(k)$ független Gauss sorozat 0 várható értékkel és σ szórással.*
Ez azt jelenti, hogy a p folyamat staciónárius és van egy periódusa, ha a

$$z^2 + \xi z + \eta = 0$$

egyenlet gyökei komplexek.

Ez a modell lehetővé teszi, hogy a CPU ütemezési problémában *dinamikus* belső prioritásokat alkalmazhassunk. Ezt a kérdést vizsgálja az ismertetendő szimulációs modell, melyben feltételezzük, hogy a folyamatok (ξ,η,σ) paraméter vektorai közeliek, igy a folyamatok pillanatnyi jellemzői közti viszonyok bizonyos szakaszokon bármilyen irányba eltólódhatnak.

3. Szimulációs modell

Legyen a job-ok CPU és I/0 periódusainak hossza exponenciális eloszlású $\lambda(\tau)$ és $\mu(\tau)$ paraméterekkel, ahol τ a job saját idejét jelöli. Legyen

$$\lambda(\tau) = \frac{1}{E} + \lambda'(\tau)$$

$$\mu(\tau) = \frac{1}{I} + \mu'(\tau)$$

ahol $\lambda'(\mu)$ és $\mu'(\tau)$ másodrendű autoregrssziós folyamatok, E és I pedig konstansok.

A multiprogramozásban futó job-ok számát jelölje N, a forgalom intenzitását pedig ρ:

$$\rho = \frac{NE}{E+I}$$

* A (ξ,η,δ) vektor egy job számára konstans, egy job folyamat számára azonban valószinűségi változó. Erről a kérdésről Arató [5] és Knuth [4] cikkekben már részletesen szóltunk.

Legyen $\tau_k(t)$ a k-adik job belső ideje a t pillanatban. Ekkor a k-adik job CPU, illetve I/0 periodusának várható hossza. a t pillanatban:

$$E_k(t) = \frac{1}{\lambda(\tau_k(t))}$$

illetve

$$I_k(t) = \frac{1}{\mu(\tau_k(t))}$$

A CPU kiszolgálásra vonatkozóan a döntést az $\{E_1, \ldots, E_n\} = E(t)$ és $\{I_1, \ldots, I_n\} = I(t)$ vektorok alapján kell elvégezni. Ezek az értékek azonban valójában ismeretlenek igy értéküket diszkrét pontokban vett minták alapján kell becsülnünk.*

Legyen most

$$\pi_k(t) = \frac{\overline{I}_k(t)}{\overline{E}_k(t) + \overline{I}_k(t)}$$

ahol $\overline{I}_k(t)$ illetve $\overline{E}_k(t)$, $I_k(t)$ és $E_k(t)$ becslései.

Speciális esetekben megmutatható, hogy a legjobb stratégia annak adni a CPU-t, amelyre $\pi_k(t)$ maximális. Nevzzük ezt a továbbiakban "legjobb", a minimális $\pi_k(t)$ ütemezését pedig "legrosszabb" stratégiának.

4. Eredmények, következtetések

A szimulációs modellben három stratégiát hasonlitottunk össze:

1) FIFO (FCFS),
2) "legjobb"
3)" legrosszabb"

A mellékelt táblázatokból kiolvasható, hogy az egyes esetekben mennyit javit a központi egység kihasználtságán az u.n. "legjobb" stratégia alkalmazása.

A stratégiák jelentősége szempontjából döntő szerepe van a forgalom intenzitás, vagyis a terheltség mértékének. Ha ez kicsi ($\rho < 1/2$) ritkán kell várakozni, így nem sok múlik a kiszolgálás módján. A túlterhelt rendszerekben ($\rho > 1$) a legjobb és legrosszabb stratégiák közötti kihasználtsági különbség 5-8% körüli. A tapasztalat azt mutatja, hogy a legnagyobb különbség $\rho \approx 0.7$ környezetében adódik. Ezt fontos tudni, mivel a tényleges rendszerekben a forgalom intenzitás éppen e körül mozog.

* Itt valójában a folyamatok paramétereinek becsléséről van szó. Ezt most nem vizsgáljuk, helyette Gy. Németh [6]-ra utalunk.

Kézenfekvő az is, hogy annál fontosabb a jobb stratégia alkalmazása minél nagyobb N értéke. A numerikus eredmények azt mutatják, hogy két job esetén ($N = 2$) a legnagyobb elérhető javulás kb. 8%, tíz job esetén ez eléri a 14%-ot, és ezután már csak nagyon lassan növekszik.

A most következő táblázatok a folyamat csillapodási paraméterének 0.1 értékére vonatkoznak. (A periodus hosszától csak az eredmények pontossága függ, az E és I konstansok pedig ρ-hoz N-hez vannak megfelelően megválasztva.)

Központi egység állásideje %-ban:

FIFO

$\rho =$	0.5	0.7	0.9	1	1.1	1.3
N						
2	52.5	43.2	25.9	21.8	16.8	9.4
3	51.8	41.0	24.8	20.1	16.2	9.1
5	51.2	38.8	22.1	19.1	15.3	8.9
8	51.0	37.1	21.0	17.7	14.5	8.6

legjobb

$\rho =$	0.5	0.7	0.9	1	1.1	1.3
N						
2	51.7	38.9	22.6	18.8	15.2	8.9
3	50.9	36.3	21.1	18.0	14.7	8.4
5	50.4	32.9	19.1	16.4	13.6	8.1
8	50.3	31.2	18.3	15.9	13.2	7.9

legrosszabb

$\rho =$	0.5	0.7	0.9
N	53.9	48.1	29.3
2	53.9	48.1	29.3
3	53.4	46.2	27.8
5	52.8	43.6	25.4
8	52.3	42.3	24.1

Befejezésül megjegyezzük, hogy a szimulációs program SIMULA 67 nyelven készült, terjedelme kb. 500 kártya, és egy táblázatbeli érték meghatározásakor kb. 1 perc gépidő szükséges. Ez 90%-os szinten az eredmények pontosságára 3% körüli átmérőjü konfidencia intervallumokat jelent.

Irodalom

[1] Arató , M. : On the diffusion approximation of operating systems.

[2] Tomkó, J.: Tömegkiszolgálás elméleti modellek számológép rendszerek matematikai leirásában.

[3] Marschall, B.J.: Dynamic calculation of dispatching priorities under OS/360 MVT, Datamation 9, 93-97 (1969).

[4] Knuth, E.: A structurel computer system model. Computers and Mathematics with Applications (nyomtatás alatt).

[5] Arató,M. Diffusion approximation for multiprogrammed computer systems. Computers and Mathematics with Applications (nyomtatás alatt).

[6] Gy. Németh. T.: A folytonos idejű másodrendű autóregressziós folyamat paraméterbecsléséről. MTA SzTAKI Közl. 10. 33-43. (1973).

Summary

A simulation study of the CPU scheduling problem
based on autoregressive processes
E. Knuth

The paper contains a new approach for the CPU – scheduling problem. The nature of the processes are assumed periodic and therefore an approximation of autoregressive processes is used. Simulation results show the possible improvement ensured by dynamic priorities.

Р е з ю м е

Имитация расписании центральных процессоров

Э. Кнут

Работа посвящена приближению процессоров расписания центральных процессов авторегрессивными процессами. Показываются результаты имитации сравнения разных стратегий.

MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

SZEMELVÉNYEK A KEREKASZTAL MEGBESZÉLÉSEKEN ELHANGZOTT GONDOLATOKBÓL

(összeállitotta: Knuth Előd)

A Téli Iskola alkalmából három kerekasztal megbeszélést tartottunk. Ezek a következő témákat ölelték fel: az operációs rendszerekkel kapcsolatos kutatási témák és ezek állása, valószinüségszámitási módszerek helye a számitástudományban, számitógép hálózatok problémái, a számitógéptudomány hazai helyzete, a kutatások módszerei és kilátásai. Ezek közül a témák közül most kettőt ragadunk ki.

1. Operációs rendszerek elméletének általános problémái

Arató Mátyás részletesen elemezte azokat a nehézségeket melyek e kutatásokat gátolják: "Az operációs rendszerek hatékonyságával kapcsolatban rendszeres méréseket szokás végezni gépek müködése folyamán. Ezek a mérési eredmények nemcsak a statisztikai kiértékelés tárgyát kell, hogy képezzék, hanem lehetőséget kell nyujtsanak a felhasználónak ahhoz is, hogy lehetőségeihez mérten jól használhassa a gépet. A mérési eredmények kiértékelése általában kétfajta következtetéshez vezet. A felesleges adatok száma, melyek nem nyujtanak kellő felvilágositást a problémák megoldásához, igen nagy. Ezt mutatják azoknak a történelmi file-oknak sokaságai, melyek feldolgozatlanul hevernek egy-egy számitógép központi polcain. Azokra az adatokra pedig melyek felvilágositást nyujtanak a gép müködésével kapcsolatban nincsenek mérési eredmények. Ez a kép is mutatja, hogy a mérések megtervezése ma még nem áll olyan fokon, mely hathatósan segitené a tervezőket és a felhasználókat. Ezeknek a méréseknek a programozása, az operációs rendszerben való elhelyezése igen fáradságos munka és éppen a programozásos szemlélet, a minden lehetőséget egyformán mérni akaró gondolkodásmód a gátja a dinamikusan kezelhető statisztikai kiértékelésnek. A programozással kapcsolatos munka igen jelentős és hosszú tapasztalatokat is igényel. Az előbb felsorolt hiányosságok oka az, hogy a számitógépek mint rendszerek leirása igen általános keretek között mozog és még ma sincs egységesen kialakult szemlélet. A gép müködése, matematikai leirása ennél még rosszabb képet mutat,mivel általánosan elfogadott elvekről nem is beszélhetünk. Egyes részek és részfeladatok, valamint részfolyamatok leirása megtalálható. A felhasznált eszközök elsősorban matematikai logikai és tömegkiszolgálási eszközök. Ezeknek a részfolyamatoknak a leirása és tanulmányozása, annak ellenére, hogy egészében nem oldják meg a problémát, igen jelentős, mivel lehetőséget nyujt a müködések lényegi része megértéséhez. Általában a megfelelő matematikai modellek a gyakorlattól messze állnak, mégis fontosabbnak tekinthetők, hiszen lényeges komponensek leirására szolgálnak. Ezek a részleirások szolgálnak a géprendszerek általános müködési képére vonatkozó elképzelések alapjául. Alapvető az a törekvés, hogy olyan modelleket és leirásokat alkalmazzunk, melyek a gépektől függetlenek, általános jellemzést adnak és felhasználhatók az egyes géptipusok értékeléséhez. Ezeknek az általános törekvéseknek az előnye, az is, hogy egyforma nyelven fognak beszélni a különböző országokból és központokból összejö-

vő szakemberek. Mint ahogyan a differenciálegyenletek nyelvén ugyanúgy hangzik a fizikai folyamatok vagy a kémiai folyamatok, esetleg bizonyos közgazdasági folyamatok leirása. Erről a nyelvről ma még csak reményként beszélhetünk, semmiképpen sem a közeljövő reális feledata. Az operációs rendszerek értékelésével kapcsolatban figyelembe kell venni a gépeken futtatható különböző tipusú feladatok sokféleségét melyeknek éppen az operációs rendszer müködése kapcsán szorosan kapcsolódniuk kell egymáshoz.

Az előadások során hallhattuk, hogy az egyes cégek milyen formában és milyen matematikai eszközökkel próbálják rendszereiket világossá és az eszközök szempontjából is közérthetővé tenni. Az IBM cég hálózatokkal kapcsolatban olyan várakozási idő és tömegkiszolgálási tanpéldát tud bemutatni, mely érzékelhetővé teszi a matematikai módszerek közvetlen felhasználását. Sajnálatos, hogy az ilyen tanpéldák még nem szerepelnek egyetemi oktatási anyagként a valószinüségelmélet és statisztika egyetemi anyagaiban. Az ilyenirányú törekvéseket szükséges idehaza támogatni.

Történtek törekvések géptől független operációs rendszerek megvalósitására és formába öntésére (erről tanuskodnak az AFIPS kiadványai). Ezek a kisérletek általában nem mutatnak még jelentős előrehaladást és egyelőre nem is látható, hogy ez a teljesen felülről jövő általános megfogalmazás mennyire vezet sikerre.

Ma már látható és könnyen le is mérhető, hogy az operációs rendszerek nem lehetnek mindenre alkalmasak és egyben hatékonyak is. Ezért van szükség különböző tipusu feladatok megoldására más és más operációs rendszerekre. Az a követelmény, hogy mindenre alkalmas és hatékony operációs rendszert hozzunk létre, durván fogalmazva, ekvivalens azzal a törekvéssel, hogy statisztikai kisérletnél az első és másodfajú hibát egyszerre tegyük zérussá, ennek lehetetlenségét minden statisztikus jól ismeri. A legfontosabb feladat az üzemi, vállalati gépek lehetőségeinek meghatározása. Ma még igen kevés eredményt ismerünk ezen a téren. Hiszen nem ismeretesek azok a nagyságrendek, amelyek gazdaságosan valósithatók meg egy egy gépen. Nem dolgoztuk ki azokat a standard módszereket, melyek megoldják, hogy egy 10 ezer munkást foglalkoztató nagyüzem a maga termelési terveivel és készleteivel milyen nagyságrendü gépet igényel. Hasonló módon kell ezeket a feladatokat megoldani mint ahogyan az ipar egyéb problémáit vizsgáljuk.

Az egyes cégek az operációs rendszerek kialakitásánál üzleti szempontokból indultak ki, azonban ez teszi lehetővé a tömeggyártás bevezetését is. Valószinüleg ez biztositja azt is, hogy egységes statisztikus szemlélettel lesz megoldható sok kérdés.

Az operációs rendszerek hatékonysága vizsgálatához szükséges mérések általában befolyásolják magukat a mérendő folyamatokat. Ennek hatását megvizsgálni és ügyelni az eredeti folyamat értékelésére, jelenleg igen nehéz statisztikai problémát jelent."

Gehér István más oldalról közelitve a problémához, a következőket fűzte Arató Mátyás megjegyzéseihez:

"A tudomány feladata mindig az, hogy absztrakció útján fogalmakat alkosson, egyszerűsitsen, és ily módon engedjen mélyebb belátást a dolgokba. A tudomány fejlődése mindig az egyszerütől halad a bonyolúltabb felé. Ezzel szemben a jelenlegi operációs rendszerek annyira bonyolultak, hogy nem teszik lehetővé matematikai modellek közvetlen alkalmazását a rendszer egészére. Mi is az operációs rendszer? Kielégitő definició, absztrakt modell nincs. Géptől független operációs rendszer ma még álom."

Ouittner Pál utalt a rendszerekkel szembeni követelmények ellentmondásos voltára. Ha hajlékonyan, sokcélúan alkalmazható rendszert készitünk, súlyos tanúlnivalókat akasztunk a felhasználó nyakába. Nem lehet olyan rendszert sem készíteni, mely hatékony is és egyúttal mindenre alkalmas is.

Vámos Tibor egy kérdésre válaszolva a következőket mondotta:
"Érdemes-e olyan kis országbam mint Magyarország új gondolatok ambiciójával fellépni? Szerintem igen. Látjuk, hogy a világon mindenhol az igazán új értékes gondolatokat 5–20 fős csoportok vetik fel.

Sajnos nálunk a próblémák mindig kicsit később vetődnek fel mint a vezető országokban, és általában az győz, aki először találkozik a problémával. Bizonyos értelemben azonban első vesztésre is van itélve. Ezt, vagyis mások negativ tapasztalatait, fel lehet használni."

2. A számitógéptudomány helyzete

Kalmár László bevezetőjében néhány alapvető fogalom tisztázására hivta fel a figyelmet. Többek között a következőket mondotta:

"A számitógéptudomány önálló tudomány, és a szemléletbeli különbség az ami a matematikától elválasztja. (Nem célunk most arról beszélni, hogy mi az ami a műszaki tudományoktól elválasztja.) Ez a különbség az algoritmikus gondolkozás. Matematikában a legbonyolultabb algoritmus a Galois–féle volt, mely eldönti, hogy egy racionális együtthatós egyenlet megoldható–e a négy alapművelet és gyökvonás segitségével. Hasonlitsuk azonban ezt össze pl. egy assemblerrel!"

Arató Mátyás részletesen elemezte, mit kell tekintenünk tudományos tevékenységnek, mi a tudományos eredmény kritériuma. A következőket mondotta:

"A hazai számitástudomány kialakitásában és a kutatások megszervezésében fontos szerepet játszanak a kutató intézmények és az akadémiai bizottságok is, elsősorban azok amelyek támogatják az ilyenirányú kutatásokat. Ezt a konferenciát is az Akadémia Számitástudományi Bizottsága szervezte. A magyarországi kutatások elősegitése nagymértékben fellendülne ha a fiatal tudósok képzése rendszeres formában történne, például az aspirantúra lehetőségeit jobban kihasználnánk. A legujabb kutatások irányitásához azonban megfelelő szintű vezetőkre is szükség lenne, jelenleg azonban az a helyzet, hogy a kutatási irányok meghatározását és a kutatá-

sok végzését önképzéses alapon végezzük. Tudományos vezetőink a már megállapodott kutatási irányokban és a klasszikus próbléma körökben, elsősorban matematikában vannak. Tőlük csak támogatást lehet kapni. Itt felvetődik az a kérdés is, hogy a számítástudományi eredményeknél mit nevezünk tudományos értékű münek? Nemcsak uj elvek és matematikai modellek, valamint a szükséges bizonyitások elvégzése az egyedüli kritériuma a tudományos eredménynek. Ha ezeket nem kisérik megfelelő szervező programok és müködő rendszerek az elért eredményeket nem tekinthetjük számítástudományi értelemben uj felfedezésnek. Az a hármas követelmény, mely az uj elv és a bizonyitás, a modell, végül megfelelő program jelenlétét teszi szükségessé nemzetközi szinten is elfogadott. Hangsulyozni kell azt is, hogy az egyes részek munkaigényessége jelentősen különbözik egymástól. A munkaigényes feladatok elvégzése szerves része a tudományos eredményeknek.

Ismeretes, hogy a Szovjetunióban a megfelelően dokumentált programrendszerek, ha azok kipróbálásra és felhasználásra is kerültek publikációként kerülnek nyilvántartásba és ilyen publikációk alapján lehetséges tudományos fokozat elérése is. A hazai tudományos közvélemény erre a formára nincsen felkészitve és nem is fogadja még el. Természetes az a követelmény is, hogy fiatal, tehetséges kutatóink ezekből a témákból jelentkezzenek aspirantúrára, lehetőleg komoly tapasztalattal és megfelelő elképzelésekkel is. A számítástudományi eredmények értékelésénél figyelembe kell venni azt a nem lebecsülendő körülményt, hogy bizonyos feladatok megoldását (sőt majdnem minden bonyolultabb feladat megoldását) csoportosan és jól szervezetten kell végezni. Az ilyen csoportokon belüli munkák megszervezése, a feladatok jó kijelölése és összehangolása alapvető jelentőségü és csak úgy érhető el megfelelő szintü értékelés, ha ezek bonyolultságát is figyelembe vesszük. Ezek nemcsak hazai tapasztalatok, mind az operációs rendszer tipusú feladatok megoldásában, mind az alkalmazási software készitésében, hanem nemzetközi fórumokon is vizsgált kérdések. Itt külön hangsúlyozni kell, hogy a megfelelően szervezett bonyolult programozási munkák alapvető részei a tudományos eredményeknek, még akkor is ha az algoritmus készitésen túlmenően nem történt meg ezek matematikai leirása.

Joggal merül fel, hogy mit tekintünk számítástudománynak. Erre a kérdésre az előbbiekben már részletes választ adtunk. Nincs szükség filozófiai vitára. Ez csak hátrányos helyzetbe hozna bennünket. Azok a problémakörök, melyek alapvetően mozgatják a jelenlegi kutatási irányokat a következőek. Az operációs rendszerek kérdései, a programozási nyelvek és rendszerek vizsgálatai, az alkalmazási software kérdései, melyek közül különösen kiemelendő a tudományosan alig vizsgált információs rendszerek, adatkezelő rendszerek készitése.

Természetesen ezeken a kérdéseken túlmenően is vetődnek fel számítástudományi kutatási problémák. Jelenleg a magyarországi kutatások szintje nem olyan, hogy egyetemeinken számítástudományi kart lehetne szervezni. A jövő azonban mindenképpen ilyen irányba mutat."

Bach Iván felhivta a figyelmet arra, hogy egy programrendszer használóját, a program belső trükkjei nem érdeklik. Nyilvánvaló, hogy egy compiler-hez user-manual-t és nem a megoldások nehézségét ismertető közleményt kell mellékelni.

Ez kétségtelen, mondta **Frey Tamás**, a felhasználótól azonban nem is azt várjuk, hogy tudományos értékelést adjon. Frey Tamás a továbbiakban felhivta a figyelmet arra, hogy miközben a számitástechnikai publikációk polgárjogaiért harcolunk, megfeledkezünk belső teendőinkről. Meg kell teremtenünk a formákat. Meg kell szüntetni azt a mély szakadékot, mely az elméleti munkák a konkrét gyakorlat között van.