# Tárolt programvezérlésű telefonközpontok operációs rendszere

DR. KÓCZY T. LÁSZLÓ
BME Híradástechnikai Elektronika Intézet

ÖSSZEFOGLALÁS

A tanulmány a mintegy 20 éves múltra visszatekintő tárolt programvezérlésű telefonközpontok operációs rendszereinek, ezen belül elsősorban az ütemezési megoldásoknak adja áttekintését. A történeti fejlődéssel összhangban először az egyszerű és összetettebb időosztásos, majd a fejlettebb funkcióosztásos megoldásokat ismerteti, melyeket összefoglalva funkcióorientált ütemezéseknek nevezhetünk. A második rész a különböző hívásosztásos, majd a virtuális processzoros ütemezési algoritmusokra mutat példákat — ez utóbbiak a folyamatorientált eljárások. A befejezésben az ütemezési rendszereknek a modern univerzális célú konkurrens operációs rendszerekben való elhelyezkedésére mutat rá.

## 1. Bevezetés

Az utóbbi évtizedben a hazai kapcsolástechnikai szakemberek érdeklődésének is középpontjába kerültek a tárolt program vezérlésű központok. Ez különösen igaz, amióta a BHG-ban megindult a tpv — először kvázielektronikus, majd teljesen elektronikus — központok gyártása, illetve a jelenleg is intenzíven folyó fejlesztési munka, melynek eredményeiről és jövőbeli útjairól a Híradástechnikában is számos tanulmány számolt be — itt csak az utóbbi évek néhány cikkére utalunk: [1], [2], ill. [3], [4].

A konkrét rendszerek, központcsaládok ismertetése mellett azonban — úgy véljük — nem érdektelen az általánosabb, a nemzetközi eredményeket is feldolgozó áttekintés sem; célunk a jelen tanulmányban a különböző, ismertebb kapcsolástechnikai gyártó vállalatok néhány fontos terméke alapján kiemelni a közös vagy éppen eltérő jellegzetességeket. Természetesen nincs mód a korlátozott terjedelem keretei között teljes képet adni, e helyett csupán a tpv központok vezérlő szoftverjének egyik leglényegesebb komponensét, az operációs rendszert, s ezen belül is a feladatok ütemezésének megoldását vizsgáljuk. Célunk olyan áttekintés nyújtása, amely a konkrét szakirodalmat tanulmányozók számára egyfajta globális szemléletet ad, s ezáltal az ismertetett rendszerek működésének megértését könnyíti, de esetleg a fejlesztők számára is tartalmaz néhány ötletet.

Mielőtt rátérnénk a fentiekben körülhatárolt téma elemzésére, előrebocsátunk néhány általánosabb gondolatot a tpv szoftver rendszerekkel kapcsolatban. E rendszerekkel szemben támasztott funkcionális és gazdasági jellegű követelmények nagy mértékben meghatározzák a struktúrát és az általános jellemzőket. E követelmények közül a legfontosabbakat az alábbiakban foglaljuk össze:

— *valós idejű vezérlési sebesség*, gyors reakcióidő a beavatkozást igénylő változások esetén;
— igen nagy *megbízhatóság;*
— *változtathatóság* mind egy terméken, termékcsaládon belül a konkrét helyszínfüggő igényeknek megfelelően; mind pedig az új technológiák, új szolgáltatások bevezetését lehetővé téve üzem közben;
— *a karbantartás és üzemeltetés automatizálása* a költségek csökkentése érdekében.

E követelmények közül az első meghatározza az egyes szoftver funkciók végrehajtásának módját, mely éppen az operációs rendszer ütemezési feladatköréhez kapcsolódik. Az adott környezeti ponton jelentkező információ (pl. ívponti állapotváltozás) meghatározott időn belüli feldolgozást, beavatkozást igényel (pl. tárcsahang indítás). Ez azonban feltételezi az észlelés gyors bekövetkezését is. Az észlelés — feldolgozás — beavatkozás funkció hármas elvégzésére rendelkezésre álló idő meghatározza azt a maximális periódushosszat, amelyen belül meg kell teremteni az adott feladatokat elvégző összes program aktivizálódásának és lefutásának a lehetőségét. Természetesen, ha az adott környezeti ponton az adott perióduson belül nem érkezett információ, a tényleges aktivizálás nem következik be. A programegységek ilyen periodikus vagy legalábbis ciklikus indítása, ill. az indítás lehetőségének megadása jelenti az operációs rendszer ütemezési feladatkörét. A fentiek alapján megállapíthatjuk, hogy az ilyen ütemezésre feltétlenül jellemző lesz a ciklikus szervezés — amely valamilyen értelemben minden tpv központban megtalálható.

*DR. KÓCZY T. LÁSZLÓ*

*1975-ben szerzett a Budapesti Műszaki Egyetemen a Villamosmérnöki Kar műszer- és irányítástechnika szakán oklevelet, ugyanitt 1976-ban kutató és fejlesztő irányú szakmérnöki oklevelet. 1977-ben a fuzzy matematikai módszerek és alkalmazásuk témakörében a BME-n kapott egyetemi doktori fokozatot. 1976-tól a BME Híradástechnikai Elektronika Intézet tud. segédmunkatársa, majd tanársegéde, 1983-tól adjunktusa. Közben 1982/83-ban egy tanévet töltött a BHG Fejlesztési Intézetében ipari tapasztalatszerzésen. Részt vett több a BHG-val, ill. a TKI-val közös fejlesztési munkában. Számos publikációja jelent meg fuzzy matematika, útkeresési eljárások és tpv vezérlési kérdések témakörben. A Mathematical Reviews recenzense, tagja az Amerikai és a Lengyel Matematikai Társulatnak, a HTE IB vezetőségének és több MTESZ egyesületnek, továbbá a Karközi Alkalmazott Matematikai Munkaközösség Szervező Bizottságának, ill. a KAMM Füzetek és a KAMM Bulletins szerkesztőségének tagja.*

Beérkezett: 1985. május 20-án (#)

A megbízhatóság követelménye szintén erősen érinti az ütemezés kérdését, a konkrét ütemezési alaptechnikák ugyanis külső vagy belső zavarok, információtorzulások hatására a központ működésének bizonyos funkciókieséseit, vagy akár maradó károsodását (természetesen szoftver károsodását) idézhetik elő. Ezeknek a problémáknak a megoldására természetesen léteznek olyan eljárások, amelyekkel az ütemezési alapmódszert kiegészítve, megbízható rendszert lehet létrehozni.

Nagyon lényeges a változtathatóság kritériuma, amelyet mind funkcionális, mind pedig piaci tényezők erősen motiválnak. Ez a kritérium az egyik előidézője a tpv szoftverek szigorú *moduláris felépítésének*, mely a teljes rendszer funkcionális modularitását követi. (Egy további, hasonlóan erős ok a fejlesztési „technológia", amely a világ valószínűleg ez idáig létrehozott legnagyobb méretű szoftver rendszereinél csak kvázifüggetlen modulok előállítása formájában képzelhető el.) A programok moduláris megoldása viszont kihat az egyes programblokkok aktivizálásának, „felütemezésének" az egységes módjára is.

A karbantartás, üzemeltetés összefügg a fentebb már említett kérdésekkel, elsősorban a megbízhatósággal és a modularitással. Külön probléma az automatikus karbantartást, adminisztratív jellegű tevékenységet végző programok ütemezése. Ellentétben a kapcsolástechnikai funkciókkal, ezek a modulok többnyire nem igényelnek szigorú időhöz kötött futást, így aktivizálásuk történhet kevésbé merev rendszerben: az ilyen feladatok elnevezése háttérjob.

E bevezetés végén megadjuk a tpv központokban található vezérlő szoftver funkciócsoportjainak áttekintését (l. *1. ábra*). A szoftver rendszer 3 fő egységből tevődik össze:

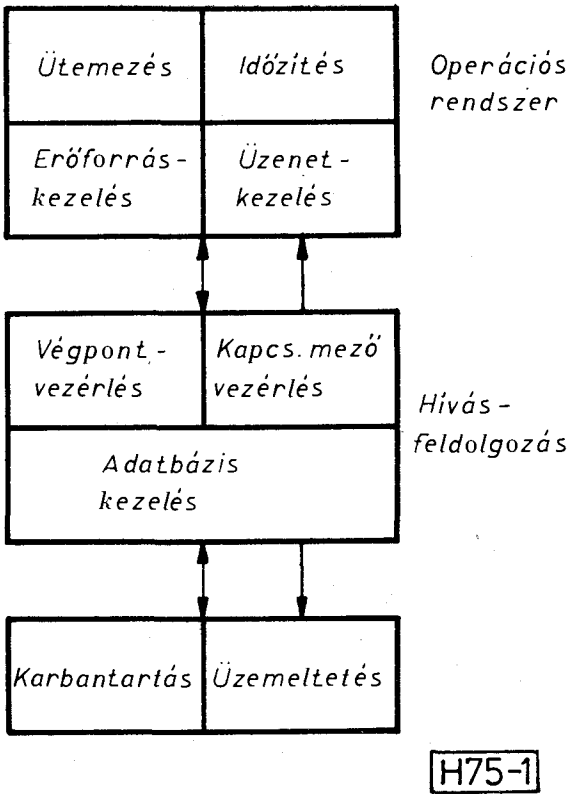
- *Operációs rendszer;*
- *Hívásfeldolgozó rendszer;*
- *Karbantartó és üzemeltető rendszer.*

Az operációs rendszer feladatait négy csoportba oszthatjuk.

- A ciklikusan végrehajtandó feladatok és a háttérjobok *ütemezése;*
- Mindenfajta tevékenység *időzítése;*
- A rendszerben található különböző *erőforrások* (pufferek, várakozó sorok) *kezelése;*
- A vezérlő rendszert alkotó processzorok, ill. a programmodulok közötti *üzenetek kezelése.*

A tényleges kapcsolástechnikai funkciókat a következő csoportokba szokás sorolni:

- *Végpont* (terminális pont) *vezérlés*, amelybe az előfizetői vonalak, trönkök, speciális feladatú ívponti áramkörök, jelgenerátorok, jelvevők, vizsgálóáramkörök stb. letapogatása és működtetése tartozik;
- *Kapcsolómező vezérlés*, amelybe a keresztpontok, összekötő áramkörök stb. állapotainak nyilvántartása, lefoglalásuk, működtetésük stb. tartozik;
- A fenti két csoporthoz kapcsolódó nagy mennyiségű *adat kezelése*, az *adatmező karbantartása.*



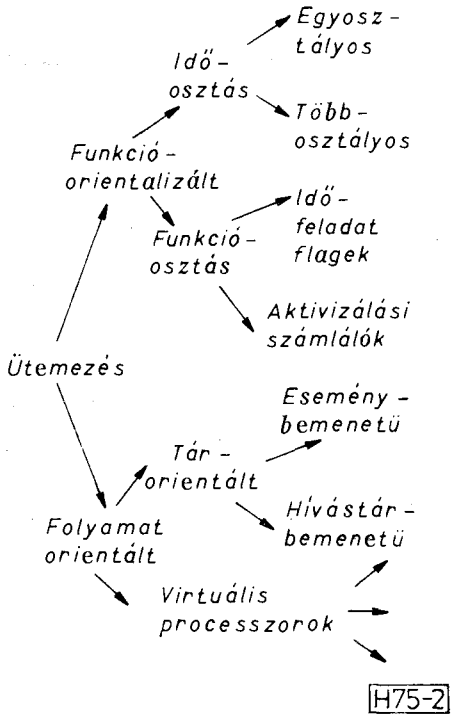

*1. ábra.* Tpv szoftver alkotóelemei

Végül a karbantartás és üzemeltetés csoportba összefoglalható számtalan feladat, amely programterjedelemben, a teljes rendszer nagyobbik felét képezi. Ennek részletesebb elemzésével itt nem foglalkozunk, mivel az ütemezés kérdéséhez kevésbé közvetlenül kapcsolódik.

Megemlítjük, hogy a tpv központok környezetében még számos további szoftver egység is található: a fejlesztést, gyártást és üzembe helyezést, valamint a karbantartást és üzemvitelt támogató számítógépes ill. tpv rendszerek programjai. Mivel azonban ezek magával a központtal csak áttételes, off-line kapcsolatban állnak, a szoros értelemben vett tpv központ szoftverhez nem sorolhatók.

## 2. Az ütemezési elvek osztályozása

Vizsgálatainkat a továbbiakban az operációs rendszerre, azon belül is a feladatok ütemezésének kérdésére szűkítjük. A téma klasszikus szakirodalma a különböző központokba alkalmazott ütemezési módszereket három típusba sorolja: egyszerű időosztásos, funkcióosztásos és hívásosztásos elvre [5]. Hasonló osztályozást alkalmazott korábban e tanulmány szerzője is — az utóbbi típus élesebb kettéosztásával kiegészítve [6]. Megállapíthatjuk azonban, hogy az előbbi két típus — további rokon eljárásokat is idesorolva — lényegi vonásaiban erősen hasonló. Az úu. hívásosztásos módszer viszont — az eseményorientált ütemezést is ideszámítva — az újabb rendszerekben alkalmazott folyamatszemléletű, virtuális pro-




2. *ábra.* Ütemezési elvek családfája

cesszoros feldolgozási elv alapjait tartalmazza, ezek az eljárások tehát egy második fő csoportot alkotnak. A két alaptípus elnevezésére javasoljuk a következő kifejezéseket: *funkcióorientált,* ill. *folyamatorientált ütemezés.*

A funkcióorientált ütemezés alaptípusai közé a következő — egyre fejlettebb változatok — sorolhatók:

— *egyszerű időosztás;*
— *többosztályos időosztás;*
— *idő-feladat flages aktivizálás;*
— *aktivizálási számlálók módszere.*

Az utóbbi kettő a „funkcióosztásos" megoldás merevebb és rugalmasabb változata.

A folyamatorientált ütemezés *tárorientált* és *virtuális processzoros* megoldásokat foglal össze. A tárorientált megoldás két alaptípusa az *eseménybemenetű* és a *hívástárbemenetű* ütemezés. A *2. ábrán* a fenti típusok „családfáját" látjuk.

A *funkcióorientált ütemezés* általános jellegzetessége az, hogy a végrehajtandó feladatok tényleges elosztásától függetlenül az ütemezés mindig a funkciók merev sorrendjében történik. Kissé leegyszerűsítve: ha pl. egy központban egy adott időszakban egyáltalán nincs olyan épülő beszédkapcsolat, amely egy előfizetői készülékre tárcsahang kiadását igényelné (mert az adott időszakban, ill. azt közvetlenül megelőzően nem kezdeményezett egyetlen előfizető sem hívást), a megfelelő időpontokban mégis aktivizálódik a tárcsahang kiadását végző program, majd valamilyen munkalista alapján önállóan dönt arról, hogy tényleges beavatkozást az adott ciklusban nem kell végeznie. Az egyes módszerek között a lényegi különbség abban rejlik, hogy az egyre fejlettebb változatokban az aktivizálás időpontjai, illetve a periodicitások egyre jobban követik az adott feladat ténylegesen szükséges aktivizálási gyakoriságát s így a processzoridő egyre jobb kihasználását jelentik.

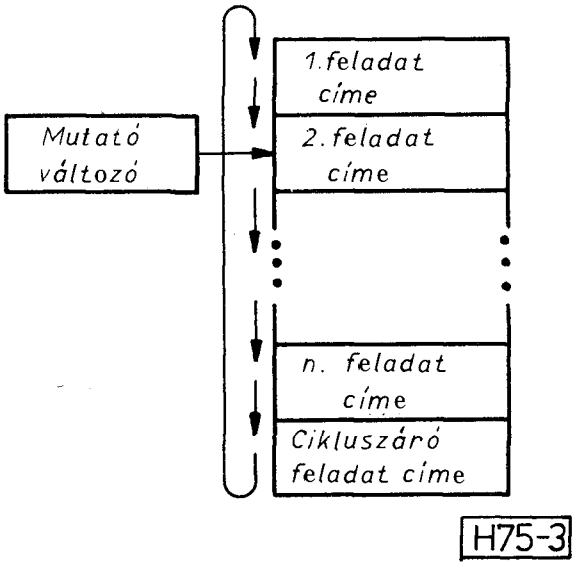
Az időosztásos változatok hátránya, hogy az aktivizálás tényleges konkrét időpontja sem mindig definiált, a funkcióosztásos módszerek, ezt a problémát lényegében kiküszöbölték. Az összes idesorolható ütemezési módszer közös hátránya azonban, hogy funkciótól függő, nagy méretű munkalistákat feltételeznek, melyek között az információáramlást kizárólag maguk az egyes funkciókat ellátó programok biztosítják — ez az egyes programok felelősségét megnöveli, hiszen a tranziens jellegű hibák is katasztrofális követelménnyel járhatnak — egy vagy több hívás felépítése, illetve felügyelete szempontjából. Végül azt is megemlítjük, hogy a funkcióorientált ütemezéshez olyan funkcionális programblokkok illeszkednek, melyek — a szigorú feladatszétválasztás miatt — minden esetben egyedi döntési mechanizmust igényelnek, vagyis a funkcionális programblokk és az adatbázis közötti szoftver interfész semmilyen szempontból nem szabványosítható.

A fenti csoporttal ellentétben, a folyamatorientált ütemezésnél mindig a tényleges feldolgozási igényeknek megfelelő programok aktivizálódnak, így ez az elv optimálishoz közeli időkihasználást jelent. Természetesen ennek a kedvező tulajdonságnak ára is van: az aktivizálás mechanizmusa körülményesebb, illetve időigényesebb. Előnye azonban, hogy homogén jellegű adatbázist használt, ami a különböző funkciók egységes rendszerbe történő összeépítését és a programadatbázis interfész egységesedését eredményezi. Említésre méltó, hogy itt az információk átadása az egyes funkciók között nem dimamikus módon, a programfunkciók közbejöttével történik, hanem az adatbázis lényegében statikus módon hordozza a programok által folyamatosan kezelt adatokat, így a tranziens funkciókiesések csupán az állapotátmenetek késleltetését idézhetik elő, s nem kerülhet sor a teljes információ vagy információcsoport végleges elvesztésére. A súlyos működési hibák természetesen itt is vezethetnek katasztrofális következményekre.

A következőkben részletesen ismertetjük az egyes ütemezési elveket.

## 3. Az időosztásos ütemezés

Az ütemezés legkezdetlegesebb megoldása az *egyszerű időosztásos ütemezés.* Ennek lényege az, hogy az egyes funkcionális modulok közül kiválasztják a leggyakoribb aktivizálást (felütemezést) igénylőt, és ehhez alkalmazkodik az összes többi programblokk aktivizálása is. Az így alkalmazott periodicitás, pl. 25 ms, nyilvánvaló, hogy számos program esetén felesleges gyakoriságot jelent, hiszen többnyire elegendő a néhányszor 100 ms-os reakcióidő, amely természetesen az információészlelés sebességére is megenged pl. 100 ms-ot. A programok rögzített sorrendjét a feladatcímtábla adja (*3. ábra*). Ebben a táblázatban sorban el vannak helyezve a végrehajtandó feladatoknak megfelelő aktivizálandó programblokkok kezdőcímei. Az ütemező feladata csupán




*3. ábra.* Feladatcímtábla szerkezete egyszerű időosztásnál

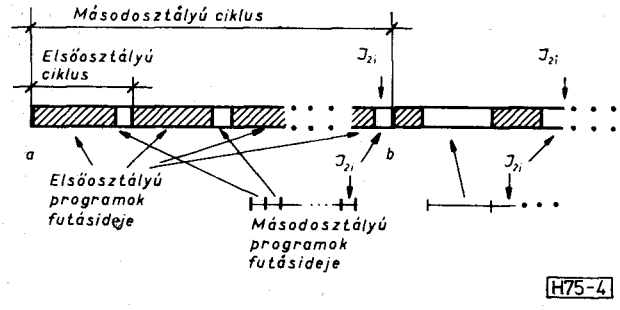
annyi, hogy a címeken valamilyen mutatóváltozó segítségével végiglépkedjen, s az egyes feladatok végrehajtását indítsa. E feladatoknak megfelelő programok saját maguk döntik el, hogy az adott ciklusban végzendő-e tevékenység, illetve hogy hány beszélgetéssel, terminállal stb. kapcsolatban. A programok végén mindig visszatérés az ütemezőbe szerepel. A feladatcímtábla végén általában egy olyan speciális program található, amely voltaképpen az ütemezési rendszer része, ennek feladata a ciklus kezdet visszaállítása, pl. a mutatóváltozó tartalmának kiinduló helyzetbe való visszaírása stb.

E megoldás hátránya, hogy a legszigorúbb követelményhez igazodik, így az időkihasználás nagyon rossz. Lényegesen jobb a helyzet a *több feladatosztályt megkülönböztető időosztásos* megoldásnál. Ekkor az egyes programblokkok az aktivizálási gyakoriság előírt minimális értéke szerinti osztályokba vannak rendezve, pl. 20—150 ms, 200—1500 ms, 2—60 s stb. Az egyes osztályokon belüli besorolásnál természetesen a szigorúbb követelmény a mérvadó. A legegyszerűbb megoldásnál az osztályok száma 2—3, mindegyik osztály külön feladatcímtáblával rendelkezik. Az ütemezés alapciklusa a leggyakoribb behívás periódusideje, minden alapcikluson belül mindenik, az első prioritási osztályba tartozó program aktivizálásra kerül. Tegyük fel, hogy az alapciklusidő 20 ms, ekkor a helyesen méretezett vezérlőjű központban a 20 ms-onként aktivizált programok együttes futási ideje (a felütemezés idejét is beleértve) 20 ms-nál lényegesen kevesebb lesz. (Pl. 10 ms.) A fennmaradó üres időben történik a 2. osztályú feladatok végzése, természetesen a 2. feladatcímtábla alapján. Legyen pl. a 2. osztály programjainak ütemezési gyakorisága 200 ms, ekkor egy másodrendű ciklus (200 ms) alatt ezen feladatok számára rendelkezésre álló összidő pl. 100 ms lesz. (Hiszen 200 ms alatt 10-szer futnak a ciklusonként 10 ms-ot igénybe vevő 20 ms-os programok.) Ha nincs több feladatosztály, az összfutásidő megközelítheti a fennmaradó 100 ms-ot, ha van további osztály is, természetesen nem használható ki a 100 ms sem.

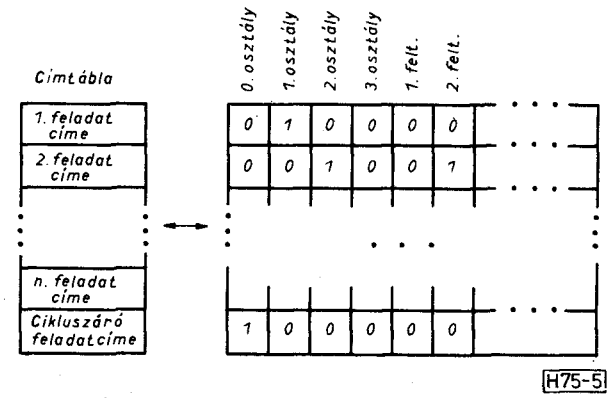
Minden méretezés alapjául valamilyen feltételezett forgalmi terhelés szolgál. A valóságban azonban előfordulhat, hogy a tényleges forgalmi érték akár lényegesen túllépi a tervezett csúcsértéket. Ekkor az adott forgalom által érintett programok futási ideje megnövekszik, hiszen ha pl. egy 20 ms-os ciklusban 30 előfizető emelte fel a kézibeszélőjét, a hurokzárást észlelő és erre reagáló (t-hangot kiadó) program futási ideje lényegesen magasabb, mint ha csak 3 hurokzárás történt. (Bár az sem igaz, hogy a futásidők aránya 10-szeres, hiszen bizonyos vizsgálati lépéseket el kell végezni akkor is, ha az adott vonalon nem történt változás: éppen e vizsgálatok alapján dönt a program, hogy további tevékenységre szükség van-e.) Ha ez az összfutásidő növekedés elér egy kritikus értéket, veszélybe kerül a többi feladatosztályba tartozó programok aktivizálásának a lehetősége. Ha az előbbi példa adatait tekintjük 2 feladatosztály esetén, a ciklusonként tervezett maximális futásidő legyen osztályonként 10 ms, illetve 80 ms. Ekkor a tervezett forgalmi értékek esetén 200 ms-onként $200-10\cdot10-80=20$ ms üresjárási idő marad. Ennél kisebb forgalom esetén természetesen még nagyobb ez az idő. Az ilyen üresjárási időt általában ki szokták használni, az ún. háttérfeladatok futtatására (karbantartás rutinszerű ellenőrzései stb.). Ha azonban az első osztályú feladatok összfutásideje 20%-kal megnő a tervezett értékhez képest, az üresjárási idő eltűnik: $200-10\cdot12-80=0$. További növekedés esetén már negatív volna ez az érték, a gyakorlatban ez azt jelenti, hogy — mivel az első osztályú feladatok futtatása mindig az elsődleges — a második osztályú programok közül néhány egyáltalán nem kerülhet aktivizálásra a 200 ms-os cikluson belül! Ha ez több cikluson keresztül ismétlődik, a központ már nem teljesíti feladatát.

Ilyen esetben egy lehetséges megoldás a ciklushossz rugalmas tágítása. Ebben a megoldásban az egyes ciklusok hossza a túlterhelésnek megfelelően nő, tehát a tényleges aktivizálás gyakorisága az előírt érték alá csökken. Ez természetesen megint csak oda vezet, hogy egy bizonyos határon túl a központ nem tudja teljesíteni feladatait.

Megfigyelhetjük azt is, hogy a fenti ütemezési módszernél kellemetlen tény a második és esetleg további osztályba tartozó feladatok tényleges aktivizálási időpontjának igen bizonytalan volta. Egy második osztályú feladat futása kezdődhet a másodrendű ciklus (pl. 200 ms) vége felé, ha az első osztályú feladatok a másodrendű ciklus első részében nagyon sok időt lefoglaltak *(4a ábra)*; vagy az elején, ha az első osztályú feladatok futása általában nagyon rövid idő alatt befejeződött *(4b ábra)*. A $J_{2i}$-vel jelölt második osztályú feladatot végző program mindkét esetben nagyjából azonos ponton aktivizálódik a saját osztálya számára rendelkezésre álló összfutásidőn belül. Ez azonban a tényleges másodrendű cikluson belül nagyon eltérő időpontot jelenthet. Ha egy *a* típusú ciklust *b* típusú követ, ez azt eredményezi, hogy a valóságban 200 ms-nál lényegesen kevesebb, ha fordítva, *b* típusút követ *a* jellegű ciklus, akkor pedig lényegesen több idő telik el a $J_{2i}$ program két aktivizálása között. Ez az idő pl. 20 ms-os és 200 ms-os ciklusidőknél elvileg




*4. ábra.* Második osztályú feladat ($J_{2i}$) sorrakerülése különböző terheléseknél




*5. ábra.* Ütemezési táblák EP központoknál

kb. 20 ms és 380 ms között mozoghat! Ha egy *b* típusú ciklust *a* típusú követ, amelyik *túlterhelt* és $J_{2i}$ nem is kerül futásra, ez az idő 380 ms-nál jóval nagyobb is lehet. Az első értékek általában nem jelentenek nagy problémát, a felső határ közelébe eső idők viszont már okozhatnak hibát. Szerencsére az ilyen szélső értékek sorozatosan csak akkor fordulhatnak elő, ha maguk a ciklusidők is rugalmasan tágulnak. A ciklusok időzítése viszont már az operációs rendszer általános időzítő feladatkörével függ össze, melyet itt részletesen nem tárgyalunk.

Az időosztásos ütemezés fentiekben ismertetett két alapváltozatát ilyen közvetlen formában, illetve önmagában a tényleges rendszerekben nem alkalmazzák (éppen a fentiekben említett nehézségek miatt).

Röviden ismertetjük azonban a második változat továbbfejlesztett megoldását, melyet a BHG által gyártott EPEX központcsalád (nem EP μ) elemeinek céljára fejlesztettek ki [7]. E megoldás lényege az, hogy az összes ütemezendő feladat címei egyetlen táblázatban szerepelnek (tetszőleges sorrendben). A feladatcím táblával párhuzamosan létezik egy feladatosztály flegtáblázat, amely jelzi az adott program osztálybasorolását. Lehetőség van olyan feladatok ütemezésére is, amelyek az aktivizálási gyakoriság szerinti osztályok egyikébe sem sorolhatók: a csak bizonyos feltételek teljesülése esetén felütemezendő programok (pl. ha a program számára üzenet érkezett, vagy a program számára üzenetküldési lehetőség adódott). A táblázatok szerkezetét l. az *5. ábrán*. Bármely feladat ütemezési gyakoriságát a 0., 1., 2. vagy 3. osztályba való tartozás határozza meg; megengedett ugyanannak a feladatnak (programnak) több osztályba való egyidejű besorolása is. A 0. osztály egyébként speciális, a vezérlő szoftver belső működéséhez szükséges alapprogramok számára van fenntartva, az 1—3. osztály a 10 ms, 100 ms és 2500 ms felütemezési gyakoriságot jelenti. Mindig a 0. osztályba tartozik a ciklust lezáró ún. újraütemező (l. feljebb). Példánk mutatja, hogy valamely program ütemezése történhet ciklikusan és feltételtől függően is. A mindenkori aktivizált feladat ütemezésének „oka", vagyis az adott felütemezést előidéző flag rögzített helyen rendelkezésre áll, tehát az egyes feladatok mindig el tudják dönteni, hogy az adott aktivizálás miatt, ill. milyen célból történt.

Ez utóbbi módszernek továbbra is hátránya az egyes feladatok tényleges futáskezdeteinél megmutatkozó bizonytalanság — pl. egy első osztályba tartozó feladat két aktivizálása között extrém esetben 19 ms (sőt, a rugalmas ciklushossznövelés miatt még ennél több) is eltelhet, nagy előnye azonban, hogy egyetlen egységes feladatcímtáblázatot tartalmaz, így az egyes programok üzemközbeni kiiktatása, helyettesítése más, esetleg másik osztályba tartozó feladattal, nem okoz problémát.

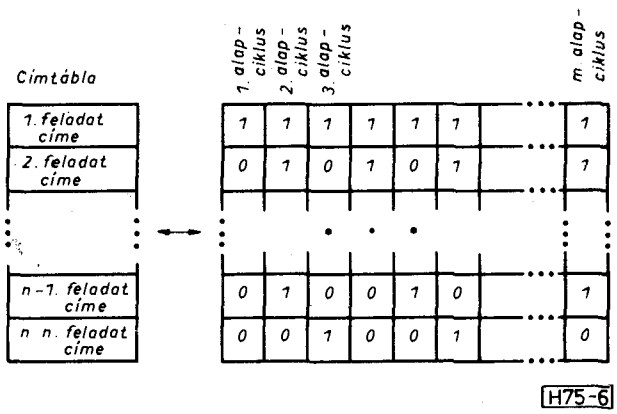
A következőkben rátérünk a funkcióorientált ütemezés fejlettebb változatára, az ún. funkcióosztásos elv ismertetésére.

## 4. A funkcióosztásos ütemezés

Az ún. funkcióosztásos ütemezési elv alkalmazásának célja az, hogy az egyes programblokkok felütemezésének konkrét időpontja jól kézbentartható legyen. Lényege az időosztásos megoldáséval egyezik, itt azonban a feladatok merev osztálybasorolása helyett lehetőség van az előírt aktivizálási gyakoriság nagy pontosságú betartására. Az ütemezés alapja itt is valamilyen rövid időtartamú ciklus, az egyes feladatok aktivizálása azonban elvileg az alapciklus idő tetszőleges többszörösével történhet.

Klasszikus, merevebb változata a *feladat-idő flagtáblázatos* módszer. Ez az elv került alkalmazásra a Bell Laboratories első ESS központjaiban [8], [9]. A megoldás lényegét a *6. ábra* segítségével ismertetjük. Az ütemezendő feladatok címtáblázatához egy olyan további táblázat kerül hozzárendelésre, amely az alapidő ciklus valamilyen véges többszörösére ($m$) tartalmazza az ütemezési tervet. A táblázat egy oszlopa az ütemezés egy alapciklusát szimbolizálja, az oszlop $i$. pozíciója az $i$. feladat aktivizálásának előírását tartalmazza, ha értéke 1. A flag 0 értéke esetén az adott ciklusban az adott program nem kerül behívásra. Legyen az alap ciklusidő $t$, ekkor vannak olyan programok, amelyek aktivizálása csak $2t$, $3t$, $\ldots \frac{m}{2}t$ időközönként szükséges, ezeknél a feladatoknál csak minden 2., 3. ill. $\left(\frac{m}{2}\right)$. oszlopban 1 a flag értéke, a többi oszlopban 0.

A fenti módszer lényege, hogy tetszőleges üteme-




*6. ábra.* Feladat-idő flag táblázat (ESS központoknál)

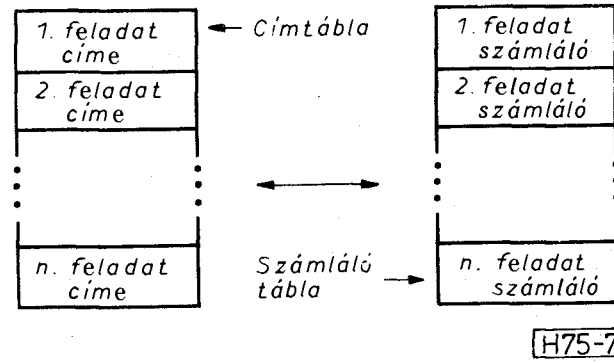
zési gyakoriság esetén $\pm t$ pontossággal meg lehet adni az ugyanazon program két felütemezése között eltelt időt. Az időosztásos megoldásnál ez az érték elvileg csak $\pm mt$ volt — az $mt$ ciklusidejű prioritási osztály esetén. A flagtáblázat hátránya, hogy nagyon sokféle behívási gyakoriság esetén mérete igen nagy.

Megjegyezzük azonban, hogy már a korai ESS központoknál is összetettebb ütemezési megoldást alkalmaztak, nem a fenti funkcióosztásos megoldás alkotta az ütemező rendszer egészét. A teljes rendszer alapja egy időosztásos hurok, amelyben a nem szigorú behívási gyakorisággal rendelkező programok ciklikus aktivizálása folyik. E programok között szerepel azonban egy, a hívástárak információi alapján, az ún. tranziens hívástár feldolgozási feladatokat végző rutinokat aktivizáló program is. Ennek a működése már az ütemezési elvek másik csoportjába tartozik, ez ugyanis egyszerű példája a hívástár bemenetű hívásosztásos ütemezésnek. A fenti keretbe illeszkedik az időzített megszakítás által indított funkcióosztásos ütemező — tulajdonképpen ezek ürejárási idejében történik a nem fix gyakoriságú programok ciklikus felütemezése.

Példaként megemlítjük, hogy az ESS No. 2. rendszernél az alapciklusidő 25 ms, az időosztásos ciklus ideje pedig kb. 100 ms. Jellemző, hogy a kötött ütemezési gyakoriságú processzorok behívási periódusai között ennél nagyobb érték is található: pl.: az MFC számjegyküldő program 150 ms-onként hívódik be. (Ha ugyanis két behívás között a távolság lényegesen kisebb volna, a vétel biztonságát veszélyeztetné.) Az aktivizálás pontossága tehát nem csak a maximális, hanem a minimális aktivizálási időköz szempontjából is lényeges. A fenti adatok összehasonlítása végett megadjuk az ESS család lényegesen korszerűbb tagjának, az ESS No. 4-nek hasonló időadatait [10], [11]: Az alapciklus hossza 10 (+3) melyhez 11—35 ms időtartamú időosztásos ciklus tartozik. Az időadatok jelentős eltérését az magyarázza, hogy a központcsalád első tagjainál számos olyan funkciót huzalozott vezérlésű interfészáramkörök láttak el autonóm módon, amelyeket a korszerűbb rendszerekben már a tpv vezérlő vett át. Megemlítjük, hogy a nagy kapacitású No. 1 típusú központokban folyamatosan kicserélték a vezérlő rendszert a No. 4-ben alkalmazott No. 1A jelű processzorra, ill. a vezérlő szoftver is korszerűsödött. (Erre a kérdésre a későbbiekben még visszatérünk.)

A funkcióosztásos ütemezés legrugalmasabb változataként említjük az *aktivizálási számlálók* alkalmazását. Ezt a megoldást alkalmazták az L. M. Ericsson által kifejlesztett AXE és rokon központok központi vezérlőjében [12]. A megoldás lényegét a *7. ábrán* szemléltetjük. Minden feladathoz egy szoftver számláló rekesz van rendelve. Az ütemezési alapciklus kezdetekor minden számláló értékét 1-gyel csökkenti az ütemező. Ezután a táblázaton végighaladva, azok a programok kerülnek felütemezésre, amelyeknek a számlálója 0-vá vált. Minden feladat elvégzése után a megfelelő program a számláló értékét újra beállítja, a következő ütemezés idejére. Ez a módszer biztosítja, hogy az alapciklusidő tetszőleges többszöröse legyen valamely feladat behívási időköze, anélkül, hogy ez az ütemezési táblázatok menetét növelné. Természetesen a véletlenszerűen egybeeső sok behívás esete ellen ez a megoldás nem véd, pl. az összes behivási időközök legkisebb közös többszörösének megfelelő periodicitással olyan túlterhelt ciklusok jelentkeznek, amikor minden feladat ütemezésre kellene kerüljön. Ha a legkisebb közös többszörös igen nagy érték (sok relatív prím található az egyes behívási időközök között), az ilyen túlterhelés valószínűsége igen kicsiny. Megfelelő védekezési technikával az ilyen esetek hatása is közömbösíthető. Szintén hátrányos a módszer időigényes volta, szemben az előbbi változattal.

Az AXE rendszerről alkotott pontosabb kép érdekében megemlítjük, hogy a fenti, funkcióosztásos megoldás csak a központi vezérlő operációs rendszerére jellemző: az ún. regionális processzorokban ettől teljesen eltérő ütemezési megoldást alkalmaznak. Ennek lényege egy egyszerű időosztásos, egyosztályos ütemezés (l. 2. pont), mely azonban egy megszakításra épített magasabb prioritási osztállyal van kombinálva. Így kerülnek ütemezésre a központi vezérlőből érkező üzenetek hatására aktivizált jobok. Mivel a regionális processzorok alapfeladatköre egyszerű tevékenységet tartalmaz, ahol a futásidő nagymértékben független a forgalomtól (pl. vonali letapogatás nyugalmi állapotban), az időosztásos megoldás itt kielégítő pontosságot eredményez a behívási gyakoriságok szempontjából.




*7. ábra.* Ütemezési táblák aktivizálási számlálókkal (AX központoknál)

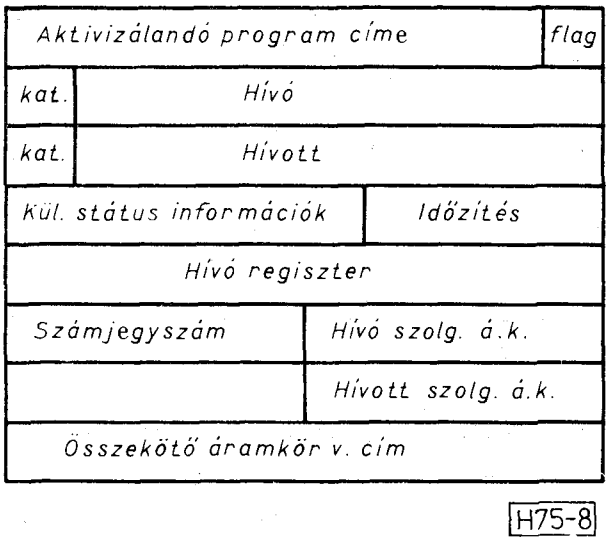
## 5. A hívásosztásos ütemezés

Az ütemezési elvek osztályozása során említettük, hogy a 3. és 4. fejezetben ismertetett időosztásos és funkcióosztásos ütemezések lényegüket tekintve hasonlóak, ezért egy közös főcsoportba soroltuk őket. Az eddig tárgyalt, funkcióorientált ütemezések elvével élesen szembeállítható a folyamatorientált szemlélet, melynek alapgondolata a *klasszikus hívásosztásos* ütemezési módszerekben jelent meg. Ez az alapgondolat a merev periodicitás helyett a feladattól függő, rugalmasan ciklikus felütemezés megvalósítását tűzi ki célul. A módszer lényegének ismertetéséhez vezessük be először a *hívástár* fogalmát.

A hívástár a memóriában elhelyezett olyan információcsomag, mely nem valamely áramkörhöz, ívponthoz, hanem egy fennálló, vagy kezdeményezett állapotban levő híváshoz tartozik. Ennek megfelelően a hívástárak nem állandó, hanem a hívásszituációtól függő, változó tartalmúak, illetve az egy időben a rendszerben található hívástárak száma is változhat, 0 és egy adott maximális érték között. (Ez utóbbit a rendelkezésre álló memóriaterület, illetve a forgalmi kapacitás szabja meg.) Néhány rendszerben a hívástár technikai kivitelezése bonyolultabban történik, pl. a logikailag egységes hívástár fizikailag több, nem összefüggő memóriaterületen található kisebb blokkból tevődik össze, ezek között fix, áram körfüggő rész is lehet.

E helyen nem feladatunk a hívástárak különböző megoldásainak, fejlődésének ismertetése, csupán példaként utalunk egy — klasszikusnak számító — jellegzetes társzerkezetre [9]. Az ESS No. 2 központban a nem beszédállapotban levő híváshoz rendelt ún. tranziens hívásrekord egy lehetséges kitöltése látható a *8. ábrán*. A tár első szava a hívás feldolgozásában soron következő program kezdőcímét és egy vezérlő flaget tartalmaz. A második két szó a hívott és a hívó azonosítóját adja meg, a legfelső biten a kategória (előfizető vagy trönk) található. A következő elem különböző státusinformációkat ad meg, pl. itt lehet jelezni, ha konferenciabeszélgetés épül fel vagy ha az első szabadút keresési kísérlet sikertelen volt. Az időzítés a különböző fázisok esetén előírt várakozási idők, leidőzítési időpontok stb. mérését segíti. Hasonlóan tárolásra kerül a már beadott számjegyek száma és a különböző célokra lefoglalt áramkörök azonosítója.

Természetesen a különböző rendszerekben alkalmazott hívástármegoldások eltérőek, közös azonban, hogy mindig tartalmazzák a hívó és a hívott fél azonosítóját, a hívásfelépítés (vagy lebontás) fázisára vonatkozó státusinformációt és a hívás kapcsán lefoglalt rendszer-erőforrások (áramkörök, memóriablokkok stb.) adatait. A hívástár tehát minden lényeges információt tartalmaz a hívás statikus állapotára nézve. Nem dönthető el azonban egy hívástár alapján, hogy egy adott pillanatban egy hívás állapotában valamilyen változásnak be kell-e következni — ez ugyanis mindig külső vagy belső *események* hatására történik. Az események érzékelése általában áramkörorientált módon történik — letapogatással vagy megszakítással — természetesen pl. egy belső időzítés lejárta csak képletesen tekinthető valamely „áramkör" jelzésének, itt ugyanis egy szoftver modul felel meg a letapogatandó objektumnak.




*8. ábra.* Példa hívástár szerkezetére

A hívástárak és események kettőssége adja a hívásorientált ütemezési elv lényegét. Bármely akció ütemezését valamely összetartozó hívástár — esemény pár észlelése indítja. A hívástárak valamilyen pufferterületen, esetleg láncban vannak elhelyezve. A hívástárorientált ütemezési változat esetében az ütemező a nem üres hívástárakat veszi sorra — valamilyen jól definiált sorrend szerint. A hívástárban található azonosítókból megállapítható, hogy mely események vonatkoznak az adott hívásra (ha ilyen esemény adott pillantban egyáltalán tartózkodik a rendszerben). A hívásállapot—esemény párok alapján valamilyen döntési mechanizmus segítségével az ütemező meghatározza a végrehajtandó job(ok) adatait és felütemezi a megfelelő programo(ka)t. Előfordulhat, hogy az állapotesemény pár még kevés információt ad a hívás fázisváltásához, ilyenkor további programok futtatása válhat szükségessé, melyek olyan további információkat (ún. pszeudoeseményeket) generálnak, amelyek alapján a hívás új fázisa meghatározható. A felütemezett program(ok) a hívástár tartalmának megváltoztatásáról is gondoskodik(nak).

A hívástárorientált hívásosztásos ütemezés természetesen önmagában nem alkalmas egy teljes ütemezési rendszer megoldásaként. Vannak ugyanis olyan szituációk, amelyek a hívástárak végigkövetésével nem ismerhetők fel. Így pl. a híváskezdeményezéseket jelző események még nem vonatkoznak semmilyen meglevő hívástárra: éppen ezek alapján kell új hívástárat generálni. Technikailag ez többnyire úgy történik, hogy az összes potenciális hívásterületek két csoportra vannak osztva: az üres, „nemlétező" tárak csoportjára és az aktív tárak csoportjára. A híváskezdeményezés hatására egy üres tár átkerül az aktív tárak csoportjába (pl. egy kontrol flag átírásával vagy átláncolással) és kitöltődik a hívó azonosítója, valamint a kezdeményezésnek megfelelő állapot. Természetesen további akciókra

is szükség van (pl. tárcsahangkiadás indítása). Innen kezdve már alkalmazható a hívásorientált ütemezés.

A kifejezetten áramkörorientált feladatok (pl. vonalak letapogatása, eseménydetekció) minden esetben a funkcióorientált ütemezés valamilyen megoldása szerint kerülnek aktivizálásra. Utaltunk már az ESS központcsalád első elemeinél alkalmazott vegyes ütemezési rendszerre, ennek lényege a funkcióosztás-hívástárorientált hívásosztás kombinációja (vö. [5], [8], [13], [9]). Ezeknél a rendszereknél a hívástárakon történő előrelépés egy kifejezetten erre a célra alkotott gépi utasítás segítségével történik, mely a megfelelő pointert a soron következő hívástár kezdő címére állítja rá.

A fentiekben ismertetett elv jellemzői a következők:

— Az információk továbbítása nem a funkcionális programok feladata, hanem a híváshoz kötött hívástáré, így egy-egy program tranziens hibája a hívások feldolgozásában általában nem okoz maradó rendellenességet.
— Az esemény és hívásállapot (és esetleg pszeudoesemény) bemenetű döntések mechanizmusa jól egységesíthető, pl. rekurzív fastruktúrával realizált állapotátmeneti tábla segítségével. Ez a funkcionális programok egyszerűsödését eredményezi. (Vö. [5], [6].)
— Bizonyos funkciók (az alapvetően eseményindításúak) ebbe a rendszerbe közvetlenül nem illeszthetők bele, tehát az elvet minden esetben kombinálni kell valamilyen funkcióorientált megoldással.
— A hívástárak állandó és kimerítő letapogatása lehetővé teszi ezek szoftver karbantartását. Ugyanakkor az események pufferterületei (várakozó sorai, láncai) nincsenek jól kézben tartva, mivel itt a keresés asszociatív jellegű. A megoldást tehát mindig ki kell egészíteni valamilyen eseményorientált karbantartási algoritmussal is.

A hívásosztásos ütemezés másik lehetséges megoldása az *eseményorientált* ütemezés. Ennek kezdeti formája a fenti rendszerhez hasonló, itt azonban az események várakozó sora, lánca kerül ciklikus letapogatásra, majd az eseményhez tartozó hívástár asszociatív jellegű megkeresése (pl. esemény által érintett ívpont keresése, mint hívó azonosító) után hasonló jellegű döntés következik, mint a hívástárorientált megoldásban [5], [6j. Ez a megoldás már a következő fejezetben tárgyalandó virtuális processzoros ütemezési elvet valósítja meg, bár erősen korlátozott formában. Az eseményorientált hívásosztás jellemzői:

— A funkcionális programok tranziens *hibáival szemben* ugyanúgy *védett*, mint a hívásorientált rendszer.
— A *döntési mechanizmus* szintén egyszerűsíthető.
— Nincs feltétlenül szükség a kiegészítő funkcióorientált ütemezésre, bár a gyakorlatban ez általában még megtalálható. (Vö. az ESS No. 4-nél alkalmazott megoldást: [10], [11].)
— Az eseménypufferek folyamatos letapogatásával szemben *nem történik meg a hívástár területek automatikus karbantartása*, ezt tehát külön programmal kell elvégezni.

A következő fejezetben részletesen tárgyaljuk az alapvetően eseményorientált hívásosztásos megoldásból kifejlődött virtuális processzorok elvén alapuló ütemezési rendszert.

## 6. A virtuális processzorok elvén alapuló ütemezés

Ha az 5. fejezetben tárgyalt hívásosztásos ütemezési elvet összehasonlítjuk a 3. és 4. fejezetek különböző funkcióorientált eljárásaival, jelentős különbséget észlelhetünk közöttük. Míg az idő-, illetve funkcióosztásos megoldások az elvégzendő feladatokat a konkrét forgalmi szituációtól független típusokba sorolják, melyek konkrét programhoz kötöttek, s ezek ütemezése a feladattípusok rögzített sorrendjében történik, a hívásosztás lényege éppen a felmerülő feladatok forgalmi helyzettől függően súlyozott, a mindenkori igényeket megfelelő erőforrás felhasználással történő megoldása. Ez utóbbi módszer tehát lényegesen rugalmasabb, így alkalmasabb a hívások számos fázisának kezelésére és egyéb, pl. karbantartási célokra is. Továbbra is hangsúlyozzuk azonban, hogy bizonyos fázisokban, mint pl. a vonalak ciklikus letapogatása, a merevebb funkcióorientált ütemezés kiválóan illeszkedik a problémához.

A jelen fejezetben az ütemezés területén alkalmazott legmodernebb megoldások közös jellemzőit szeretnők összefoglalni, e megoldások bonyolultsága miatt azonban ezt csak az eddigieknél általánosabb keretek között tesszük meg. A megoldás alapgondolataként induljunk ki az elvégzendő feladatok néhány jellegzetességéből.

A kapcsoló központokban megvalósított vezérlési rendszer számos *al*rendszerre bomlik. Az erős tagolás jól illeszkedik a méret-, változtathatósági és egyéb okokból is szükségszerűen kialakított moduláris szerkezethez. Az alrendszer forgalmával tulajdonképpen egy-egy konkrét hívás kezeléséig és/vagy egy-egy híváskezelési funkcióig is lemehetünk. Ebben a szemléletben — az alrendszerek közé természetesen a karbantartási, üzemeltetési és egyéb funkciókat is besorolva — igen *nagy szám*ú, egymástól vezérlési szempontból kevéssé függő, illetve csak üzenetjellegű kommunikációs kapcsolatban álló, egymáshoz képest aszinkron módon zajló alrendszerhez, ún. *folyamathoz* jutunk. Ezek mindegyike egy-egy szekvenciális automata modelljével írható le, hiszen kimerete (hatásai, elküldött üzenetei) a bemenetek (kapott üzenetek, külső információk) adott sorozatának a függvényeként alakul ki — a korábbi bemenetek lényeges megőrzendő információit az alrendszer belső állapota tárolja [14]. Egy-egy alrendszer határait az alábbiak szerint érdemes megvonni:

— az alrendszeren belül legyen erős funkcionális összetartozás (kohézió);
— az alrendszeren kívüli feladatokkal (más alrendszerek) a vezérlés csatolása legyen minél lazább (*aszinkronitás*).

A fentieknek megfelelő folyamatok mindegyike belülről úgy látja, mintha a processzor kizárólag vele

foglalkoznék, s mindazok az időszakok, amikor a valóságban más folyamatok futnak, belülről nézve nem léteznek, mivel ekkor a folyamat *felfüggesztett* állapotban van. Az operációs rendszer és ezen belül elsősorban az ütemezés feladata ilyen megközelítésben az, hogy minden folyamat számára biztosítsa a saját *virtuális processzor* használatát.

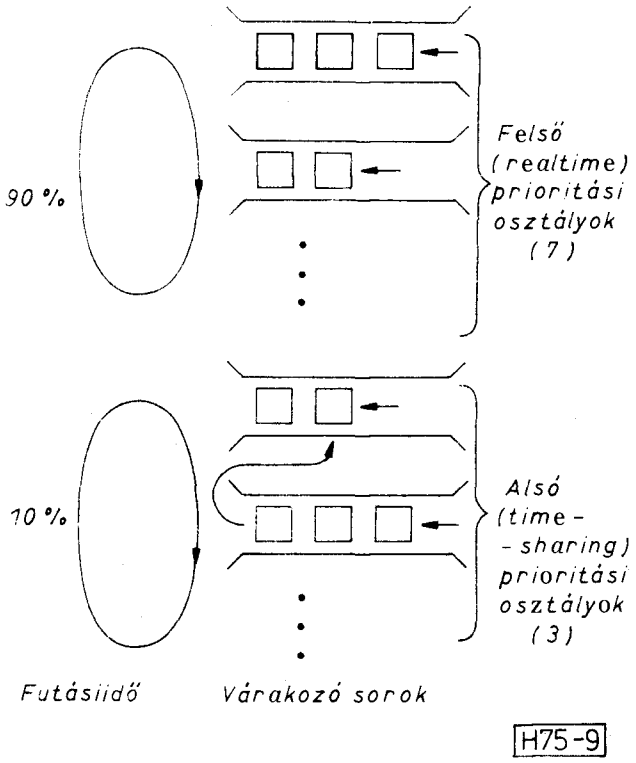
A fentiekben vázolt szemlélet természetesen nem csak a kapcsolóközpontokon belüli feladatokra vonatkozhat. Hasonló alrendszerek jelentkeznek minden nagyobb volumenű valós idejű vezérlési irányítási feladaton belül, így pl. a számítógépes folyamatszabályozásban. Nem meglepő tehát, hogy az egyidejűleg, egymáshoz képest aszinkron módon zajló folyamatok programozásának irodalma és gyakorlata a tpv kapcsolástechnikához hasonlóan a 60-as évekre nyúlik vissza — s eredetileg a kapcsolástechnikától meglehetősen függetlenül fejlődött. Az ún. *konkurrens* programozás alapirodalma számos olyan gondolatot felvet, melyek a legújabb tpv központokban felhasználásra kerültek (l. pl. [15], [16], [17]. A konkurrens programozás első alkalmazásai, illetve a kifejlesztett módszerek jellegükben a tpv kapcsolástechnikai ütemezések első csoportjával, a funkcióorientált ütemezéssel egyeznek, élesen jelentkezik a rokonság pl. a [16]-ban ismertetett elvek és a [12] alapján leírt aktivizálási számlálók módszere között. Legjellemzőbb ezekre az ütemezésekre az alrendszerek (folyamatok, funkcióorientált programok) állandó száma, melyet az operációs rendszer közvetlenül változtatni nem tud. Az alrendszerek tényleges működésében azonban a kapcsolóközpontok esetében kezdettől fogva jelentkezett a környezettől függő feladatmennyiség ingadozása. A rögzített alrendszer — (job típus-) szám esetén ez az egyes programok futási idejének igen erős változásában nyilvánult meg, mely az ütemezési módszerek használhatóságának erős korlátját is jelentette. A változó folyamatszám ezt a problémát a lehető legrugalmasabban oldja meg — természetesen a vezérlő rendszert túlterhelő forgalom hatását semmilyen ütemezési technika sem tudja eltakarni.

A vázolt virtuális processzoros közelítés természetesen még sokféle konkrét megoldást megenged. Az ITT 1240 (System 12) rendszere [14] pl. az elosztott architektúrának megfelelően terminális-pont orientált folyamatokra épít. Ez a technika viszonylag egyszerűen áttekinthető, programblokkonként lényegében függetlenül tesztelhető, ill. cserélhető szoftvert eredményez, melyet különösen a teljes program „réteges" szerkezete biztosít. Ennek lényege a következő: minden terminális jellegű hardver elemet egy hozzárendelt szoftver kezel. E kettő együtt adja a „virtuális terminált". A virtuális terminálok kezelését egy olyan következő programszint végzi, amely a külvilágtól csak funkcionális tartalmú üzeneteket kap — ezt az alsóbb szintre már konkrét, de logikai jellegű üzenetek formájában adja át, a fizikai parancsok csak a legalsó szinten jelentkeznek. Ez a technika lehetővé teszi, hogy pl. a teljes terminális hardvert úgy kicseréljék (természetesen a működtető handler programokkal együtt) hogy pl. az ütemezés szintjén semmilyen változtatást ne kelljen végezni, hiszen a folyamatok funkcionális jellemzői ettől még változatlanok maradhatnak.

Az egyes alrendszerek (folyamatok) természetesen nem egyenrangúak, a prioritási osztályok az egyes virtuális processzorok aktivizálódásának egymáshoz képesti tényleges idejét meg kell hogy szabják. A Bell—Northern DMS—100 családjánál [18] az SOS elnevezésű operációs rendszer a folyamatok 8 prioritási osztályba történő besorolását teszi lehetővé, a folyamatok ún. vezérlő blokkja tartalmazza a prioritási címkét (0—7). Az ütemező az aktivizálásra várakozó folyamatok közül először a legmagasabb prioritásúakat választja ki, az azonos prioritású, egyidőben aktivizált folyamatok egymás között az egyszerű időosztáshoz hasonló elven osztoznak meg az időn. E rendszernél a nagy gyakorisággal végzendő feladatok a központi vezérlőtől függetlenül autonóm módon zajlanak, így az egyes folyamatok futása viszonylag hosszabb lehet. Az ütemezés alapciklusa 6,25 ms, egy folyamat azonban max. 6 ciklus idejét foglalhatja el. (Ilyen hosszú aktivitás azonban igen ritkán, csak néhány speciális hívásfeldolgozási feladatnál fordulhat elő, l. később.) A gyakorlatban egy-egy folyamat 1, legfeljebb 2 ilyen időrést foglal le. A meglepően hosszú idők magyarázata abban a megközelítésben rejlik, hogy a virtuális processzoros folyamatszemléletű ütemezésnél az ütemező már igen bonyolult s így maga az aktivizálás a futásidővel összemérhető nagyságú időfogyasztást jelez. Hosszabb összefüggő futásnál az aktivizálás rezsi ideje százalékosan kisebb. Ennek szellemében az SOS ütemező lehetőleg mindaddig hagyja a folyamatot futni, amíg az minden, az adott információk ismeretében elvégezhető tevékenységet elvégzett, s végül önként mond le a futásról (valamilyen újabb üzenetre, leggyakrabban időzítésre fog várakozni, vagy megszünteti önmagát). A 6 ciklus utáni „erőszakos" futásmegszakítás elsősorban védekezési célú: így nem fordulhat elő, hogy egy hibás folyamat, mely önként nem mond le a futásról, megbénítja az ütemezést.

A hívásfeldolgozási és más sürgős, lehívási gyakorisághoz kötött folyamatok mellett természetesen a DMS rendszerben is vannak háttérfeladatok: olyan karbantartási, üzemeltetési jobok, amelyek aktivizálása soha nem sürgős, de amelyek futása természetesen szintén szükséges a helyes működéshez. Ezek a folyamatok az alsó három prioritási osztályt alkotják s így előfordulhatna, hogy a magasabb prioritású folyamatok hosszabb ideig állandóan kiszorítják őket a futásból. Ennek elkerülésére a teljes futási idő 10%-ában kizárólag ezek a folyamatok kerülnek aktivizálásra (*9. ábra*). Az ütemezés tehát a következők szerint zajlik: Az ütemező megvizsgálja, hogy vannak-e folyamatok a legmagasabb prioritású osztály várakozó sorában. Mindaddig, amíg ez nem üres, e folyamatok kapják meg az őket tartalmazó processzort. Ha ez kiürült, következik a második várakozó sor stb. Ha közben újabb folyamat jelentkezik futásra egy magasabb osztályban, az éppen aktív folyamat lemondásig fut, de utána újra a magasabb prioritási sor kerül kiürítésre. A futásidő meghatározott szeletében ugyanezen szisztéma szerint az alsó 3 osztály várakozó soraiban levő folyamatok kerülnek felütemezésre.

A hosszú futású folyamatok olyan módon kerülik el a 6,25 ms-onként beérkező megszakítást (mely újra az ütemezőbe adja vissza a vezérlést), hogy ezt a




*9. ábra.* Feladatok ütemezési rendje (folyamatok aktivizálása) DMS központoknál

megszakítást maszkolni lehet. 6×6,25 ms után azonban a maszkolás automatikusan leidőzítésre kerül.

A fenti rendszerre még a ki/beviteli pontok megszakításai rakódnak, melyek a legmagasabb prioritású folyamatnál is jogosultak a futás felfüggesztésére. Az ütemező megszakításhoz hasonlóan itt is lehetőség van az időszakos maszkolásra.

Érintettük már a különösen hosszú aktivitási idejű folyamatok kérdését. A DMS—100-ban alkalmazott megoldás e szempontból összetettebb, mint a korábbiak, pl. a Northern Telecom SP—1 központjában megvalósított rendszer. Ennek lényege a rövid folyamataktivitási idő (max. 1—2 ms), mely természetesen egy-egy hívás „életében" sokszori folyamataktivizálást jelent. A rövid futásidő a hívások sok rövid fázisra való bontását teszi szükségessé, ennek az előnye az, hogy a folyamatok soha nincsenek aktív állapotban, ha funkcionálisan várakozni kénytelenek (pl. külső válaszra, pszeudoesemény beolvasására). A nehézséget itt az jelenti, hogy a bonyolult, különleges hívások (pl. pénzbedobós készülékről történő hívás) esetén a folyamatok felfüggesztett állapotban igen sok (közbenső) adatot kell tárolnia, s a következő aktivizáláskor bonyolult algoritmus szerint kell dönteni a továbbiakról. Ehhez a szemlélethez az egyes folyamatok által igényelt virtuális processzor egységes volta illeszkedik, tehát itt — szélső értékben — egyetlen, igen bonyolult program futása fogja jellemezni a folyamatok aktivitását.

A DMS—100 családnál a nem túl bonyolult hívások felépítésére ugyanez a megoldás került alkalmazásra. A különlegesebb eseteket viszont saját virtuális processzor (különböző programmal) bonyolítja le, lehetőség szerint olyan hosszú futással, hogy minimális adat őrzésére legyen csak szükség két aktivizálás között. Ilyenkor általában a híváskezdeményezéstől a tárcsázási hang kiadásáig, a tárcsázás befejezésétől a hívás teljes felépítéséig stb. felfüggesztés nélkül fut a folyamat (a tárcsázást magát autonóm egység bevételezi). A hívások kvázinyugalmi fázisaiban (pl. beszédállapot) a folyamatok teljesen megszüntetik önmagukat, a hívásra vonatkozó információkat csupán egy egyszerű hívástár tartja nyilván.

Maguk a folyamatok általában a következő elemekből épülnek fel [20]:

— *folyamatvezérlő blokk* (PCB), amely tartalmazza a folyamat mindenkori prioritását, állapotát stb.;

— *saját stack* (csak a folyamat számára hozzáférhető adatokkal);

— *globális címinformációk* (több folyamat által közösen használt adatterületek címei);

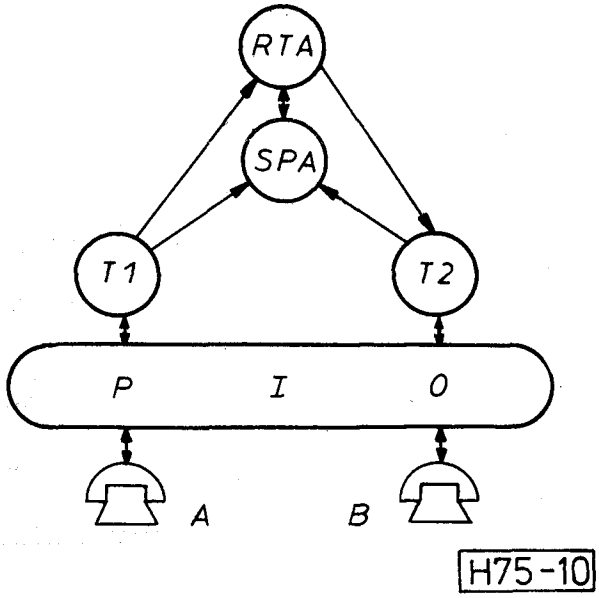
— a folyamat *programja* (mely aktivizálás esetén a folyamat futását valósítja meg).

Az adatok tárolásának különböző lehetőségei természetesen más megoldásokat is megengednek, így a DMS—100-nál a hívástárak ideiglenesen a folyamatok belső adattárolásának feladatait is nagyrészt ellátják.

Az ESS No. 5 központokban a folyamatok nem hívás, hanem híváskezelési funkció szerint különülnek el, ezért van szükség a saját belső tárolásra. A rendszerben megtalálható fontosabb híváskezelési folyamatok: kapcsolóút felépítés és lebontás, útkeresés, illetve végponti választás, konfigurációs adatok kezelése, terminálkezelés (hívó és hívott fél), perifériák kezelései (letapogatás stb.). Ezek mellé adminisztratív, karbantartó (hibakezelési), programkezelési (üzemközbeni módosítás stb.) folyamatok társulnak. Ez a szemlélet [20] közel áll a System 12 rendszeréhez [14]. Egy egyszerű hívás felépítésében pl. a *10. ábrán* látható folyamatok vesznek részt.

A PIO (periféria ki/beviteli) folyamat letapogatja a híváskezdeményezést, majd létrehozza a TI hívó oldali terminálkezelő folyamatot. A választás befejezéséig PIO és TI látja el a feladatokat, ezután TI az információkat az RTA (útkereső és végpont választó) folyamatnak küldi el. A kiválasztott útvonal felépítését SPA (útfelépítő és -lebontó) folyamat végzi el, ugyanez a folyamat hozza létre a T2 hívott oldali terminál folyamatot. A hívott oldal állapotát PIO segítségével T2 állapítja meg és erről jelzést küld Tl-nek stb.

A fentiekben összefoglalt, az ESS No. 5 rendszerben alkalmazott konkurrens programozási technika egyébként szervesen épül a korábbi Bell-központtípusok modernizálása kapcsán kialakult, a 3. és 4. fejezetben ismertetett eljárásoknál hatékonyabb, rugalmasabb ütemezést alkalmazó operációs rendszerekre. A már említett, a No. 1 és No. 4 központok céljaira alkalmas 1A típusú processzor teljesen új alapszoftvert is alkalmaz [21] ennek ütemezési technikája implicit módon a virtuális processzoros megoldást közelíti: A feladatok megoldásának ütemezőben rög-




*10. ábra.* Egy hívás felépítésében részt vevő folyamatok terminálorientált virtuális processzoros megoldásnál (ESS No. 5-nél)

zített sorrendje helyett itt maguk a feladatok (tulajdonképpen folyamatok) hordozzák azokat az azonosítókat, amelyek alapján a mindenkori prioritás eldönthető. Ennél az átmeneti rendszernél problémát okoz, hogy míg a merevebb megoldásoknál az ütemező között működése miatt fel sem merülhetett, hogy két egymást követően felütemezett program működése egymást akadályozza, a ritka virtuális processzor szemléletű megoldásoknál viszont minden folyamat aktivizálódásakor a futáshoz szükséges környezet (pl. stackből) beállítódik, ennél az átmeneti rendszernél egyes programok között interferencia jöhet létre. A megoldást a programok keresztirányú blokkolása adja, minden olyan program (folyamat), amely futását még nem fejezte be (felfüggesztett állapotban van), az ütemező számára megadja azokat a programokat (ill. programcsoportokat) amelyek futása a felfüggesztett állapot alatt nem megengedhető. Az ütemező mindig a legmagasabb prioritású, nem blokkolt programot aktivizálja.

E megoldás azonban csak ideiglenes volt, a 80-as évek elejére kialakult az ESS központok és más a Bell Laboratories által gyártott telefóniai berendezések egységes vezérlője és operációs rendszere [22]. Ez az operációs rendszer valósítja meg a már tárgyalt ESS No. 5 konkrét alapszoftverjét, de a No. 1A és No. 4 központok nagyarányú kapacitásnövelését is (ez utóbbiaknál pótlólagos 3B20D processzor hozzáillesztésével, az ún. APS változatban). Az általános célú DMERT (ill. UNIX) operációs rendszer rugalmassága és lehetőségei maximálisak, alkalmazási területei messze túlnyúlnak a távközlésen [23]. A teljes prioritási hierarchia 16 szintet tartalmaz, ebből a 3 legalsó a nem valós idejű (egyszerű time sharing) folyamatok számára. Az ütemezés alapgondolata hasonló a DMS 100-nál ismertetettéhez, mindig a legnagyobb prioritású várakozó sor kiürítése történik először. Az aktivizálást kérő folyamatok saját szintjük várakozó sorának mindig a végére kerülnek, ez biztosítja, hogy szinten belül megvalósulhasson a ciklikus felütemezés. A magasabb prioritási folyamatok általában megszakítják az alacsonyabb szintű folyamat futását s csak ezek futásról való lemondása után tér vissza az ütemező a megszakított folyamathoz (ilyenkor azonban ez természetesen nem kerül a várakozó sor végére, csak saját futásról való lemondása után). Érdekes a felfüggesztés és aktivizálás együttes idejére vonatkozó adat: a jellemző érték, kb. 320 μs, vagyis a DMERT-nél az ütemezési időveszteség nagyon kicsi.

Az alsó három prioritási szinten belül az operációs rendszer a folyamatok prioritását dinamikusan osztja ki (a valós idejű fokozatok prioritása rögzített), a time sharing szinteken a várakozás hatására növekszik a prioritás.

A folyamatok létrehozása és megszüntetése szintén dinamikus módon történik, a DMERT érdekessége, hogy maga az operációs rendszer is folyamatokra bomlik, melyeket a rendszer *töltő* hoz létre. Az operációs rendszeren kívüli (tehát pl. a központ működését vezérlő) folyamatokat a *folyamatkezelő* folyamat hozza létre, mely maga is a betöltés során létrejött folyamat. További különlegesség, hogy lényegében bármely folyamat létrehozhat új folyamatot önmaga lemásolása útján (ún. folyamat elágazás). Ez a rendszer tetszőleges konkurrens program létrehozására és futtatására alkalmas.

A virtuális processzoros ütemezési megoldások áttekintését zárjuk néhány gondolattal, melyek kissé konkrétabban összekapcsolják az egyszerű ütemezési alapproblémákat az itt alkalmazott fejlett technikával.

Ha valamely hívást a kezdeményezéstől a megszűnésig (vagy valamilyen közbenső nyugalmi fázisig) egy folyamat kezel, a hívás különböző állapotai miatt más és más aktivitási gyakoriság, tehát ütemezési időköz szükséges. Ez az időköz jellemző az egyes állapotokra, így a folyamat rendelkezésére áll. Minden egyes programfutás végén ez az adat megadja, hogy a program következő futására mikor kell sor kerüljön, vagyis a folyamatot mikor kell aktivizálni, ezt az értéket a program elküldött üzenet formájában közli az időzítő folyamattal, amelyik a továbbiakban gondoskodni fog az érintett program megfelelő időpontban történő újraaktivizálásáról — így folyamatunk a futásról lemond és felfüggesztett állapotba kerül. Amikor az időzítő (pl. dinamikusan hozzárendelt belső aktivizálási számlálói segítségével) észleli, hogy egy adott folyamatot indítani kell, elküldi az üzenetet, az ütemező tevékenysége csupán abban áll, hogy az adott pillanatban feldolgozatlan üzenettel rendelkező, tehát aktivizálásra váró folyamatok közül a prioritásnak megfelelően egyet kiválasszon. Természetesen maga az időzítő folyamat is bizonyos időközönkét aktivizálást kér, hiszen különben nem tudná a különböző időzítések számlálóit kezelni. (Pl. történhet megszakítás segítségével is.)

Ilyen módon biztosított az, hogy minden feladat a szükséges időpontban (semmi esetre sem előbb) hajtódjék végre, természetesen nagyon sok folyamat egyidejű futásra várása esetén itt is hasonló problémák állnak elő, mint akár egy időosztásos ütemezésnél, torlódás esetén.

A jellegzetes funkcióorientált feladatok (pl. letapogatás) nem híváshoz, hanem továbbra is funkcióhoz kötöttek, de szintén egy-egy folyamat keretében kerülnek megvalósításra, a folyamatot aktivitása alatt (mely ilyen esetben célszerűen rögzített időközönként történik) a funkcióorientált ütemezéshez hasonló elvű virtuális processzor fogja kiszolgálni. Az ilyen, rögzített időközű (és viszonylag sűrű) tevékenységek természetesen nagy prioritást kapnak. Megállapíthatjuk tehát, hogy e legfejlettebb rendszerek alkotóelemként az összes korábbi eljárást tartalmazhatják, illetve tartalmazzák is.

## 7. Összegezés

Tanulmányunkban áttekintettük a tpv telefonközpontok operációs rendszerének funkcióit, s kiemelten foglalkoztunk az ütemezés kérdésével. A tpv központ szoftver alapvetően konkurrens jellegű, így kezdettől fogva olyan ütemezési megoldások létrehozását tekintették célnak, amelyek sok feladat kvázipárhuzamos, egymástól jórészt független megoldását tették lehetővé.

Az első alkalmazott megoldások az elvégzendő funkció szerinti merev ciklikus végrehajtást alkalmaztak, ezeket együttesen funkcióorientált ütemezéseknek neveztük. Legfejlettebb változataik, a funkcióosztásos rendszerek az egyes feladatok előírt legkisebb végrehajtási gyakoriságának megfelelő rugalmas felütemezési rendszert alkalmaztak.

Az előbbi csoporttal szemben áll az a technika, amikor nem az ütemező rögzített táblázatai, hanem a magukban a végrehajtandó feladatokban (programban, vezérlő blokkban) hordozott információ képezi az aktivizálás alapját. Ezek kezdeti formája a híváskezelés feladatait (hívásosztásos ütemezés), fejlettebb változatai pedig az összes vezérlési feladatot ilyen folyamatszemléletű ütemezéssel valósítják meg. A folyamatok egymástól gyakorlatilag függetlenek és mindegyikük saját virtuális processzorát látja végrehajtóként. E rendszerek alkalmazási köre igen széles körű, ennek megfelelően számos változatban fejlődtek ki, az utolsó fejezetben mozaikszerű képet igyekeztünk adni a legfontosabb ilyen megoldásokról; befejezésként a DMERT rendszer néhány jellegzetességéről, mely az irodalomból ismert legfejlettebb konkurrens ütemezési megoldást tartalmazza — mindenesetre a tpv telefonközpontok területén.

## IRODALOM


[1] *Makay Attila—Hasenauer Miklós—Dr. Reznák Roxán:* TPV telefonközpontok hívásfeldolgozó feladatainak programozása. Híradástechnika, 1983. No. 1. pp. 27—31.

[2] *Horváth Imre:* Magyar fejlesztésű kis kapacitású digitális alközpontcsalád. Híradástechnika, 1984. No. 6. pp. 241—247.

[3] *Pató Lajos:* A TPV központok folyamatos korszerűsítésének szükségessége és feltételei. Híradástechnika, 1982. No. 11. pp. 505—507.

[4] *Eisler Péter:* A kapcsolástechnikai fejlesztések főbb irányai a BHG-ban. Híradástechnika, 1983. No. 8—9. pp. 348—350.

[5] *M. T. Hills—S. Kano:* Programming electronic switching systems—real-time aspects and their language implications. Peter Peregrinus Ltd., Stevenage, 1976. 207. p.

[6] *Kóczy T. László:* Szoftver a kapcsolástechnikában. Magyar Posta Központja, Budapest, 1983. 229 p.

[7] *Kovács Zoltán—Marosdi János—Szebeni Zoltán:* EP512 műszaki információ: System2 számítástechnikai működtető rendszer. INF6-025034-011. BHG FI dokumentáció. Bp., 1984. 69 p.

[8] *J. A. Harr—E. S. Hoover—R. B. Smith:* Organization of the No. 1 ESS Stored Program. BSTJ, 1964, No. 5. pp. 1923—1959.

[9] *R. J. Andrews—J. J. Driscoll—J. A. Herndon—P. C. Richards—L. R. Roberts:* No. 2 ESS: Service Features and Call Processing Plan BSTJ, 1969. No. 8. pp. 2713—2764.

[10] *T. J. Cieslak—L. M. Croxall—J. B. Roberts—M. W. Saad—J. M. Scanlon:* No. 4ESS: Software Organization and Basic Call Handling. BSTJ, 1977. No. 7. pp. 1113—1138.

[11] P. *D. Carestia—F. S. Hudson:* No. 4. ESS: Evolution of the Software Structure. BSTJ, 1981, No. 6. Part 2. pp. 1167—2101.

[12] AXE 10. Data processing system APZ210. Central processor subsystem CPS. pp. 8—53. Telefonaktiebolaget L. M. Ericsson. Stockholm, é. n.

[13] *D. H. Carbaugh—G. G. Drew—H. Ghiron—E. S. Hoover:* No. 1 ESS Call Processing. BSTJ, 1964. No. 5. Part 2. pp. 2483—2533.

[14] *D. A. Lawson:* A New Software Architecture for Switching Systems. IEEE Tr. on Communications, 1982. No. 6. pp. 1281—1289.

[15] *E. W. Dijkstra:* Cooperating sequential processes, in: Programming Languages (ed.: F. Genuys), Academic Press, New York, 1968.

[16] P. *Brinch Hansen:* The Architecture of Concurrent Programs. Prentice Hall Inc., Englewood Cliffs, 1977.

[17] *A. N. Habermann:* Synchronisation of communicating processes, Commun. of the Asso. Comput. Mach., 1972, No. 3. p. 143.

[18] *B. K. Penney—J. W. J. Williams:* The Software Architecture of a Large Telephone Switch. IEEE Tr. on Communications, 1982, No. 6. pp. 1369—1378.

[19] *D. M. Lasker:* Module Structure in an Evolving Family of Real Time Systems. Proc. of the 4th Int. Conf. of Software Engineering, 1979, pp. 22—28.

[20] T. *Duncan—W. H. Huen:* Software Structure of No. 5. ESS: A Distributed Telephone Switching System. IEEE Tr. on Communications, 1982, No. 6. pp. 1379—1385.

[21] *G. F. Clement—P. S. Fuss—R. J. Griffith—R. C. Lee—R. D. Royer:* 1A Processor: Control, Administrative, and Utility Software. BSTJ, 1977, No. 2. pp. 237—254.

[22] *R. W. Mitze—H. L. Bosco—N. X. DeLessio—R. J. Frank—N. A. Martellotto—W. C. Schwartz—R. M. Wolfe:* The 3B20D Processor and DMERT As a Base for Telecommunications Applications. BSTJ, 1983. No. 1. Pt. 2., pp. 171—179.

[23] *M. E. Grzelakowski—J. H. Campbell—M. R. Dubman:* DMERT Operating System, ibidem, pp. 303—322.