DIPEX software-rendszer*

SZEGHY ISTVÁN
BHG Híradástechnikai Vállalat, Fejlesztési Intézet

ÖSSZEFOGLALÁS

A cikk a DIPEX telefon alközpont család software rendszerét mutatja be. Egy — a tervezők által kialakított — software modell ismertetése után, a megismert virtuális eszközökkel leírja a DIPEX operációs rendszerét. A rendszer I 8085 mikroprocesszorral történt realizálására példákat mutat be.

Bevezetés

A Dipex telefonalközpontról Horváth Imre (akitől egyébként a Dipex (DIgital Private branch EXchange) elnevezés is származik) publikált elsőként áttekintő, részletgazdag rendszerismertetést (1).

Cikkét tekintjük — és ajánljuk — eligazodási alapként dolgozatunkhoz is, amiben a Dipex software-rendszerét (Dipex Software System: DPSS) kívánjuk közelebbről bemutatni.

A DPSS-t mérnökök készítették és nem számítástechnikai szakemberek, ezért a rendszer alapját képező software-modell mérnöki szemléletet tükröz felépítésében és terminológiájában egyaránt. Kidolgozásában azonban sok olyan eredmény segített, amit számítástechnikai alapokról értek el. (Gondolunk itt olyan modellkonstrukciókra, mint az Agent [5], Cell [7], Soma [8]).

Alapvető rendező- és konstrukciós elve a fizikai működés és az elvárt, ill. szükséges információs folyamatok optimális illesztése.

Ezen elv alapján húztuk meg (definiáltuk) az egyes határfelületeket, az elkülönülő részek közötti kommunikáció rendjét, ami egy moduláris programrendszer ökonomikus kialakításán túl jelentős egyéb eredményeket is hozott.

— A modellben szemléletesen fogalmazhatók meg az egyes szolgáltatások megvalósítási módjai.
— Újabb szolgáltatások beépítése a rendszerbe egyszerű.
— A programok kipróbálása áttekinthető környezetbe kerül.
— A DPSS támogatja a rendszer diagnosztikáját mind a bemérés idején, mind üzem alatt.
— Javul a mindig problémás hardware—software-együttműködés, ami legtöbbször abból a körülményből ered, hogy a szakemberek nem értik egymás nyelvét.
A DPSS lehetőséget ad arra, hogy — képletesen szólva — a határfelületek ellentétes oldalán állók rálássanak egymásra és egymással kommunikálni tudjanak.

Beérkezett: 1986. VII. 30.-án (#)

SZEGHY ISTVÁN

Egyetemi tanulmányait a Budapesti Műszaki Egyetemen végezte, ahol 1954-ben gyengeáramú villamosmérnöki oklevelet kapott. 1954—1958 között üzemmérnök az Elektronikus Mérőkészülékek Gyárában (EMG). 1958-tól az Elektromechanikai Vállalatnál (EMV) mint fejlesztőmérnök, majd laborvezető, kommunikációs rádióadók fejlesztésével foglalkozott. Jelenleg a BHG Fejlesztési Intézetében (az EMV jogutódjánál) annak a laboratóriumnak a vezetője, ahol egy digitális telefonalközpontcsalád softwarefejlesztése folyik.

A mérnöki software-modell

Ahhoz, hogy a Dipex software-rendszerét ismertetni tudjuk, először szólnunk kell arról a modellről, amiben a rendszer működését ábrázoljuk.

Modellünk eszközrendszerét — a villamosmérnöki gyakorlatban különösen használatos — helyettesítő képek mintájára alakítottuk ki.

Amint a helyettesítő képpel egy eszköz összes lényeges tulajdonsága leírható, ugyanígy a softwaremodellünk konstrukciói egy-egy program működését szimbolizálják, anélkül, hogy tudnánk valamit is a számítógépen futó objectprogram részleteiről. Software-modellünk eszközei

az SDLCP (SDL-Central Processor) (l. [2])
a VPU (Virtualis processzor, aminek egyedüli funkciója a műveletvégzés a modell specifikációja szerint)
a RAM (írható-olvasható memória)
a VIO (virtualis input/output eszközök)

elemekből épülnek fel.

Modellünk használata gép- és programnyelvfüggetlen.

Vegyük sorra a mérnöki software-modell alapkonstrukcióit!

1. Software-jelzések
A modell — alább ismertetésre kerülő — eszközei közötti kommunikáció alapelemei, amiket a virtuális input/output (VIO)-ok fogadnak, ill. killdenek.
Programozástechnikai példa erre a következő: Két program fut a rendszerben. Az első eredményét a második — mint adatot — kívánja felhasználni. Ezt az adatot, paramétert nevezzük software-jelzésnek. A két program lehet szekvenciális vagy konkurens futású!
2. Folyamatábrával leírható eszközök
Ezek olyan programokat modelleznek, amiknek a működése folyamatos, szemben a 2. pont alatt ismertetett várakozó állapotokkal megszakított működésű programokkal.

2.1 A software-gép
VPU-ból és RAM-ból épül fel. A különböző gépeknek különböző VPU-ja ós RAM-ja van. A software-gép a saját RAM-jában talált adatok alapján hajtja végre feladatát. Példa a csengetés végrehajtása: amikor egy flag engedélyezi a gép működését, az sorban kiolvassa a RAM-jában szereplő mellékállomások kódjait és azoknak meghúzatja a csengetőjelfogóját.

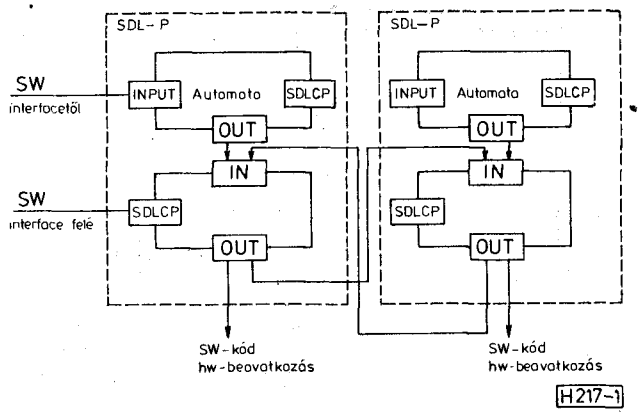
2.2 Software interface
VPU-ból és RAM-ból áll. Feladata a software-jelzések átmeneti tárolása, közvetítése, a modell SDL-eszközeinek aktivizálása.

3. SDL-ben leírható eszközök

Az SDL ismertetése helyett a (2), (4) irodalomra hivatkozunk. Az SDL-ben leírható eszköz alatt olyan szakaszosan működő programokat értünk, amik meghatározott műveletsorozatok végrehajtása után várakozó állapotba kerülnek, majd bizonyos jelzések hatására újra folytatják futásukat.

Természetesen egy ilyen jellegű programfutás eléréséhez meghatározott operációs rendszerrel rendelkező CPU szükséges. Modellünkben építőelemként felvettük az SDLCP absztrakt központi processzort, aminek az a tulajdonsága, hogy ilyen szakaszos programvégrehajtásra alkalmas.

3.1 SDL-automata
Virtualis input/output-tal (VIO) és SDLCP-vel rendelkező eszköz, aminek az a jellemzője, hogy outputként kizárólag software-jelzéseket ad ki. Nem rendelkezik saját memóriával.
A rendszerben hardware-illesztésekre használjuk: segítségével történik a hardware-jelzések előfeldolgozása.
Működése: várakozó állapotából — meghatározott inputjelzések hatására — aktivizálódik. Ekkor végrehajtja a beérkezett inputhoz programozott műveleteket, majd újra várakozó állapotba kerül.
Bizonyos inputok és programok hatására meghatározott outputjelzéseket ad ki.
A DPSS ezzel az eszközzel oldja meg pl. a számjegy-bevételezést. A számjegy-SDL-automata feldolgozza a hardware felől érkező állapotváltozás-, ill. a belső órák időzítősjelzéseit és az alábbi kimenőjelzéseket állítja elő, kódolt formában:
— szám érkezett
— letett a mellék
— időtúllépés
— földelőgomb-működtetés
— nem szám-információ érkezett
Egy SDL-automata mindig hardware-specifikus, de a kimenő jelzése független a hardware-tól. Tehát mindegy, hogy a tárcsázó mellékállomás egyenáramú impulzussal vagy MPC-val küldi a szám információját — ha a megfelelő SDL-automata van rákapcsolva, a további feldolgozás számára, egységes formában kerül tovább a szükséges információ.



1. ábra. SDL — processzorok egymásközti kommunikációja

Alkalmazása — tehát — a rendszer dinamikus átkonfigurálására kiválóan alkalmas. (Vö. [6])

3.2 SDL-processzor
Virtualis input/output-tal (VIO), SDL-központi egységgel (SDLCP), saját írható-olvasható memóriával (RAM) rendelkezik.
Input eszközével mind hardware-, mind software-jelzéseket fogad; outputként hardware-vezérlőjeleket, ill. software-jelzéseket ad ki.
Általában működéséhez egy, az aktuális feladatának megfelelő, SDL-automatát kapcsol fel saját magára.
Ez azt jelenti, hogy a felkapcsolt SDL-automata használja az SDL-processzor memóriaterületét, és output-eszköze kizárólag az SDL-processzornak adja át a jelzéseit.
Az SDL-processzorok képesek VID-jukon keresztül egymással is kommunikálni, közvetlen vagy közvetett (megfelelő SW-interface-en) módon átadott sw-jelzésekkel (lásd 1. ábra).
Konstrukciónkhoz hasonló a (9)-ben ismertetett Finite Message Machine (FMM) virtuális eszköz.

A mérnöki software-modell realizálása

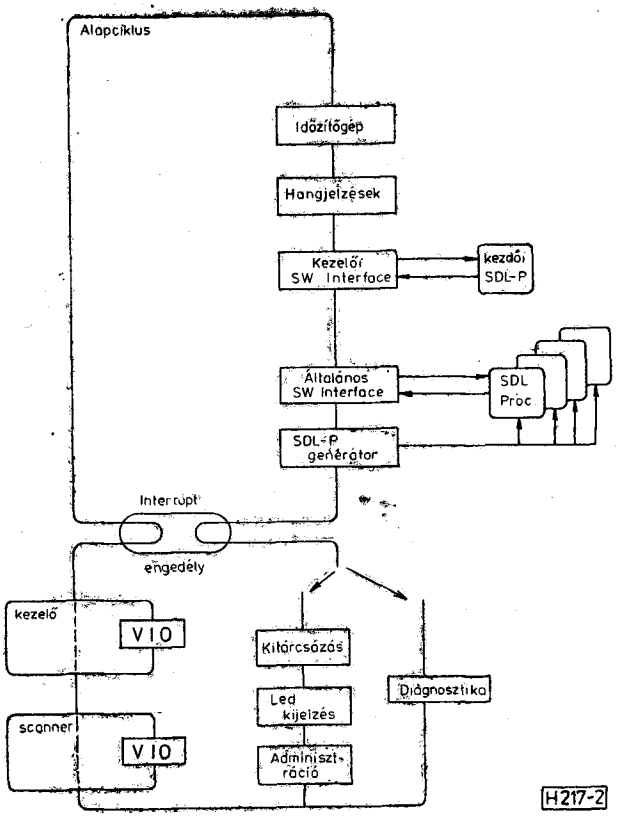
Az előzőekben ismertetett modellt a DIPEX alközpontcsaládban az I 8085-ös mikroprocesszorra realizáltuk.

A folyamatábrával leírható eszközök programozása INTEL assembly-ben történt, az SDL-ben leírható eszközök programozására egy speciális nyelvet fejlesztettünk ki. Ennek a nyelvnek az alapkoncepcióját és konstrukcióját a (2) cikk ismerteti. A cikk megjelenése óta a nyelv további utasításokkal bővült, amik az SDL-processzorok egymás közti kommunikációját támogatják.

A DIPEX operációs rendszere

Az operációs rendszer felépítését és működését a 2. ábra mutatja.

Az operációs rendszer ciklikus felépítésű. Az ábrán látható hurkok — amik sw-modelleszközöket fűznek össze — jelentik ezeket a ciklusokat.



2. ábra. DIPEX operációs rendszere

Az alapciklusra felfűzött eszközök 10 ms-onként feltétel nélkül, a hurkon látható sorrendben, aktivizálódnak.

Ezt nevezzük a központ szívdobogásának. (Hibátlan ismétlődését a CPU-kártyán lévő led villogása jelzi.)

A többi ciklus meghatározott hardware-változások hatására aktivizálódik.

A ciklusokat különböző szintű (prioritású) interruptok indítják. Az ábráról a ciklusok prioritásviszonyai is felismerhetőek. (A nagyobb prioritású ciklus „letakarja" az alacsonyabb prioritásút.)

Jól látható, hogy az alapciklusnak első szakaszát interrupt nem szakíthatja meg. Ebben a szakaszban aktivizálódnak a belső időzítő gépek, a hangjelzéseket generáló gépek, a kezelői és az általános sw-interface-ek és az új SDL-processzorokat generáló gép.

Az alapciklus további szakaszán futnak azok a programok, amik hosszabb futási időt igényelnek, de interrupttal megszakíthatók. Ebben a szakaszban felváltva — páros alapciklusban az egyik, páratlanban a másik — két különböző programhurok aktivizálódik. Az egyik a hívásfeldolgozással kapcsolatos hosszú idejű programokat tartalmazza (led-kijelzéseket, fővonali kitárcsázást, bontások adminisztrálását végző SW-gépek), a másik a real-time diagnosztikai programokat.

Ez utóbbival kapcsolatosan szeretnénk rámutatni arra, hogy néha a legegyszerűbb megoldások milyen hatásosak tudnak lenni. A rendszer kialakításának és az első real-time-programok bemérésének idején ebben a ciklusban egy egyszerű dumpprogramot futtatunk, amivel display-ra egy tetszőlegesen megadható memóriamezőt írattunk ki. Ezzel egy ablakot nyitottunk (Window-nak is hívjuk a mai napig) az élő központra, ahol in vivo vizsgálhattuk a központot. Különösen előnyös volt ez a módszer a hagyományos trace-eljárásokkal szemben, mert egyidejűleg több memóriabyte-ot tudtunk megfigyelni, ezzel a kombinált többszörös hibákat egyszerűen ki tudtuk emelni.

A scannerciklus egyetlen eszközből áll, amit modellünkben VIO-nak neveztünk. Feladata, hogy a scanner-áramkör felől érkező jelzéseket, amik hardware-állapotváltozások hatására generálódnak, beírja az általános SW-interface memóriájába. Hasonló funkciója van a másik interrupt szinten elhelyezkedő kezelői VIO-nak. Ez a kezelőkészletről érkező jelzéseket továbbítja a kezelői SW-interface-hez.

Ez a két hurok, ill. VIO realizálja bemenőoldalon a hardware—software-határfelületet.

Ebből azonnal látható a DPSS egyik alapelve, ami szerint időben elválasztódik a beérkező jelzések fogadása és azok feldolgozása. (A VIO-ok által átadott jelzések feldolgozása az SW-interface-ekben folytatódik, amikor a főciklusban ezek aktivizálódnak.

A DPSS alapciklusának legfontosabb feladata, hogy az software interface-ein keresztül aktivizálja mind a kezelői, mind az általános SDL-processzorokat.

(Utóbbiakhoz zömmel a hívásfeldolgozást végzők tartoznak, de lehetnek speciális célú (diagnostikai, időmérő stb.) SDL-processzorok is a rendszerben.)

Az ábrán a kezelői SW-I-hez egy SDL-processzor (a kezelőé), az általános SW-I-hez több SDL-processzor csatlakozik. Az ábrázolás módja azt kívánja érzékeltetni, hogy ezek az eszközök mintegy egyidejűleg, konkurens módon működnek. A valóságban a rendszer olyan, hogy az egyes SDL-processzorok általában várakozó, inaktív állapotban vannak ós csak akkor aktivizálódnak, ha az SW-I erre utasítja őket. Ez akkor történik, ha egy SDL-processzor számára software-jelzés érkezik: hardwareváltozás vagy -időzítés következtében, esetleg egy másik SDL-processzortól. Természetesen az SW-I soros működésű, de a mikroszinten zajló feldolgozás macroszinten egyidejűséget mutat.

Aki a telefonalközpontot használja, úgy érzékeli ezt, mintha a központ csak az ő hívásával lenne elfoglalva, egyedül csak vele foglalkozna. (Vö. [3]) Ez a körülmény a software tervezője számára is rendkívül leegyszerűsíti a feladatot.

A tervezés során elegendő egyetlen hívásra kidolgozni a feldolgozás programjait, az operációs rendszer elintézi a „sokszorosítást"!

A DPSS-ben a munkavégzés ilyen formában való végrehajtása egyéb előnnyel is jár. Nevezetesen lecsökkenti a műveleti időket, mivel az egyes hívásokkal csak akkor foglalkozik, ha arra igény van. Nevezhetjük ezt úgy, hogy a külvilág információgenerátorát igyekszik optimálisan lezárni az információt feldolgozó-fogyasztó telefonalközpont.

Ehhez a témakörhöz tartozik a főciklus utolsó, nem megszakítható működésű gépe, ami új SDL-processzorokat generál.

Ennek az a feladata, hogy a mellékállomás kezdeményezésére újabb SDL-processzort vezessen be

a rendszerbe. A gép programjának algoritmusától függ, hogy ezt mikor engedélyezi és mikor utasítja vissza. Ha ez az algoritmus bizonyos forgalmi terhelésektől függő visszacsatolásokkal rendelkezik, az előbbi információillesztés esetleg még tovább javítható, de mindenképp túlterhelés elleni védelmet jelent.

Példák a DPSS telefon-alközpontokban történt realizálására

A mérnöki software-modellban szereplő virtualis processzort (VPU) a DIPEX egyprocesszoros rendszerében úgy realizáljuk, hogy az operációs rendszer az egyes modelleszköz számára — annak működési idejére — „kölcsönadja" a központ CPU-ját. Tehát a DPSS-ben szereplő eszközök VPU-i mind a közös hardware-vezérlőprocesszorral azonosak. Ez a működések soros jellege folytán lehetséges.

A DPSS alapvetően SDL-proeesszorcentrikus. Minden kommunikációs akció, ami a rendszerben él, ehhez az absztrakt eszközhöz van kapcsolva. Vannak:

1. hívásorientált SDL-processzorok
   Ezek egy-egy önálló, belső folyamat (pl. egy hívás) számáragenerálódnak a központ működése folyamán. Memóriaméterük és szervezésük meghatározott, elhelyezkedésük a memóriában dinamikusan változik (ahol hely van!).
   Megszűnnek, ha az általuk képviselt folyamat is megszűnik (pl. a házi beszéd állapotban mindkét fél bont).
2. ívpontorientált SDL-processzorok
   Ezek működésükkel annak az ívpontnak viselkedését képezik le, aminek aszámára generálódtak.
   Lehetnek:
   2.1 meghatározott ívponthoz állandóan hozzárendeltek. Ilyenek a kezelő- és a fővonalak SDL-processzorai. Ezek a rendszer generálásakorlétesülnek, mindegyiknek a memóriacíme és -mérete állandó és nem szűnnek meg.
   2.2 egyes ívpontokhoz dinamikusan hozzárendeltek. Ezek az igényeknek megfelelően generálódnak, amíg szükség van rájuk, élnek a rendszerben, majd megszűnnek. Memóriaszervezésük és méretük állandó, de elhelyezkedésük a memóriában (memóriacímük) dinamikusan változik. Ilyen típusú SDL-processzorokkal írható le a több hardware-processzoros rendszerek működése is!

Példa a hívásorientált SDL-processzor működésére

A hívásorientált SDL-processzor szervezési módot azokban a DIPEX-alközponttípusokban választottuk, ahol egyszerűbb szolgáltatás választékot kellett megvalósítani.

Példaként leírjuk röviden, hogyan épít fel egy házi beszéd-kapcsolatot — ebben a rendszerben — egy mellékállomás.

1. Amikor a mellékállomásunk felemeli kézibeszélőjét, a scanner-áramkör b-ágas változást detektál és az operációs rendszernél interrupttal jelentkezik. Az interrupt elfogadásakor a scanner-SW-gép beírja az általános SW-interface memóriájába a hívást kezdeményező mellékállomás ívpontadatait (az ívpont sorszámát és az ívpont logikai állapotát leíró kódot).
2. Az általános SW-interface, mikor mellékállomásunk felől érkezett kód feldolgozásához ér, megállapítja, hogy egy eddig szabad, inaktív ívpont aktivizálódott, ezért egy új SDL-processzorgenerálást kér attól az SW-géptől, aminek ez a feladata.
3. Az SDL-processzor generáló gép, ha a megfelelő feltételek teljesülnek — létrehoz egy új SDL-processzort.
   Felkapcsol egy számjegybevételező SDL-automatát.
   Magát az SDL-processzort beállítja a számjegybevételezés alapállapotába, majd időzítéssel és egyéb adminisztratív adatokkal látja el. Létesít egy virtuális outputot, mellékállomásunk számára, ami a felkapcsolt SDL-automata inputjára adja át az ívponton létrejövő jelzéseket.
   Ezek után azt mondjuk, hogy a híváshoz rendelt SDL-P mint hívót vonta hatáskörébe a mellékállomásunkat.
   A hívó mellékállomás számára tárcsahang kapcsolódik fel. Az új SDL-P várakozó állapotba kerül ós a vezérlés visszaadódik az általános SW-interface-nek.
4. Mellékállomásunk, miután betárcsázta a hívott mellék hívószámát, a számjegybevételezés állapotában lévő SDL-P megvizsgálja, hogy szabad-e a hívott. Ha szabad, lekapcsolja az eddigi SDL-automatáját és egy csengetőautomatát kapcsol magára fel. Egyben létesít egy virtuális inputot a hívott ívpont felé, amin keresztül érzékelni tudja majd annak jelentkezését.
5. Miután a hívott jelentkezik és sikerült a beszédutat felkapcsolni a két mellékállomás között (azaz volt szabad időrés-pár), a híváshoz rendelt SDL-P:
   — egy beszédfigyelő SDL-automatát kapcsol magára
   — létesít egy virtuális outputot a hívottja számára (amin keresztül az átadhatja az ívponti viselkedését a felkapcsolt SDL-automata számára)
   Ezt az akciót itt úgy nevezzük, hogy a hívás processzora a mellékállomást mint hívottat vonta hatáskörébe.
   — felveszi a „mellék—mellékbeszéd"-állapotot.
6. A beszédkapcsolat megszűnik, ha az egyik fél bont. A bontást a beszédfigyelő SDL-automata érzékeli a virtuális inputján keresztül. Erről software-jelzést ad SDL-processzorának, ami a szükséges műveleteket elvégzi. (Beszédútkikapcsolás, időrés-felszabadítás, a virtuális -output eszköz eltávolítása a letett mellékállomás ívpontjáról.)
   A hívás SDL-processzora most a beszédkapcsolat „állva maradt" mellékállomását vonja hívóként hatáskörébe és számára foglaltsági hangot kapcsol fel.
   Új SDL-automata aktivizálódik (az általános figyelő automata) és maga az SDL-processzor a „Foglaltsági hang" állapotba kerül.

7. Miután a második résztvevő is letett (aki a foglaltsági hangot kapta), megindul a bontás folyamata.
   Ez magába foglalja a foglaltsági hang lekapcsolását, az ívpontján lévő virtuális output eltávolítását és a hívás SDL-processzorának megszüntetését.

Példa az ívpontorientált SDL-processzor működésére

Ha olyan szolgáltatások megvalósítása válik szükségessé, amiben kettőnél több résztvevő aktivitását kell koordinálni (ilyen pl. az a visszahívás, amiben mindhárom résztvevő bontani tud, a konferenciakapcsolás stb.), akkor a hívásorientált SDL-processzoros megoldás — a különböző virtuális pontok elhelyezése és kezelése miatt — kényelmetlenné, áttekinthetetlenné, nem egy esetben lehetetlenné válik.

Az ívpontorientált SDL-processzoros modell ereje szemléletességén túl abban van, hogy implicite magába foglalja a több különböző, konkurens módon működő hardware-processzoros realizálás tervezésének lehetőségét.

Az SDL-modell eszközök programozására kifejlesztett programnyelvünk legújabb változata rendelkezik olyan konstrukciókkal, amik támogatják az ilyen típusú működési módot. Nevezetesen a
SEND (kód) TO (cím)
syntaxisú utasítás, amivel az egyik eszköz a másiknak software-kódot tud küldeni. (A [cím] egy software-úton (programmal) megvalósított interfaceeszközt jelent, amit a felhasználó definiál)

Továbbá egy
Procedure input do
IF (kód) THEN (akció) ELSE input end
syntaxisú utasítással megvalósított software-inputkezelés.

Példaként bemutatjuk egy olyan visszahívásállapot kezelését, amiben mindhárom résztvevőnek saját SDL-processzora van és mindhárom egymástól független akciókra képes.

(Általában a hazai viszonyok között a mellékállomás—fővonali beszédállapotból a főközpont nem tud bontani. Példánkban szereplő fővonal tud bontani!)

Tekintsük azt a szituációt, amikor a B azonosítójú mellékálomás parkol (várakozik), az A azonosítójú mellék beszél az FV azonosítójú fővonallal. Két akciót részletezünk:

1. Letesz a parkoló „B" m. állomás
   (Példánk a software-üzenetátadásokat mutatja be.)
   A letevés kritériuma közismerten az, hogy az ívpont nyugalmi (szabad) állapotának megfelelő logikai állapotot eredményező b-ágas változást — meghatározott ideig — ne kövesse újabb b-ágas változás.
   Ebből látszik, hogy a letevés kiértékelése egy összetett folyamat.
   Ezt a folyamatot a B mellék SDL-processzora szuverén módon végrehajtja és hatására bontja — megszünteti — saját magát, felszabadítva így a B mellék ívpontját.
   Bontási folyamatának beindítása előtt egy kódot küld az „A" m. állomás SDL-processzorának, amivel értesíti azt, hogy bontott. Az „A" m. állomás SDL-processzora az eddigi visszahívásos beszéd állapotát megváltoztatja normál beszéd állapotra és kódot küld a fővonali SDL—P-nek, hogy a kettőjük közötti beszédállapotban nincs tovább harmadik várakozó!
2. Letesz az „A" m. állomás.
   (Példánk az SDL-processzorok szuverén működését mutatja be.)
   Az „A" m. állomás SDL-processzora megállapítja, hogy a mellék bontott, egy kódot küld beszédpartnerének — a fővonalnak —, hogy letett.
   A fővonal SDL—P-a ennek a kódnak hatására jogossági vizsgálatot hajt végre, hogy a „B" m. állomás jogosult-e fővonali összeköttetésre? (A kóddal együtt automatikusan adatot is átadunk.) Ha igen, beszédállapotra kapcsolja magát a „B" m. állomás felé és kódot küld vissza „A" SDL—P-ának, aminek hatására az egy kódot küld a parkoló „B" SDL—P-ának, majd bont. A parkoló „B" aktivizálódik és beszédállapot jön létre a fővonal és a „B" mellékállomás között.
   Azt, hogy a közvetlen hardware-beavatkozásokra melyik SDL—P adjon utasítást, triviálisnak érezzük azt a megoldást, hogy mindig az, amelyik közvetlen észleli a hardware-változást! Az egyes analízisek elvégzésénél (pl. a jogosultságok meghatározása) nem ilyen egyértelmű a helyzet.
   Példánkban a fővonali jogosultságot a fővonal SDL—P-ával végeztetjük és nem a m. állomáséval, azzal a meggondolással, hogy az az eszköz analizáljon, amelyiknek működése az analízis eredményétől függ.
   Olyan szituációban, amikor a visszahívásban három mellékállomás vesz részt és a „parkoltató" m. állomás letesz, nincs szükség jogosultsági analízisre, hogy a két m. állomás között beszédállapot jöjjön létre.

IRODALOM

[1] *Horváth I.:* Magyar fejlesztésű kis kapacitású digitális alközpont család. Híradástechnika, XXXV. évf. 1984. 6. sz.
[2] *Szeghy István:* SDL-processzor. Híradástechnika, XXXV. évf. 1984. 6. sz.
[3] *dr. Kóczy T. László:* Tárolt programvezérlésű telefonközpontok operációs rendszere. Híradástechnika, XXXVI. évf. 9. sz.
[4] CCITT ajánlások. Yellow Book, Vol. VI/7
[5] *N. Natarajan:* Communication and Synchronization Primitives for Distributed Programs. IEEE Trans. Vol. SE—11. No. 5. April 1985.
[6] *J. Kramar and J. Magee:* Dinamic Configutarion for Distributed Systems. IEEE Trans. Vol. SE—11. No. No. 4. April 1985.
[7] *S. Silberschatz:* Cell: A Distributed Computing Modularization Concept. IEEE Trans. Vol. SE—10. No. 2. March 1984
[8] *J. L. W. Kessel:* The Soma: A Programming Construct for Distributed Processing. IEEE Trans. Vol. SE—7. No. 5. September 1981
[9] *R. Arranz, R. Conroy, L. Katzschner:* Structure of the Software for a Switching System with Distributed Control. SESSION 41 A Paper 3. ISS'81 CIC Montreal 21—25. Sept. 1981