INFORMATICA 1/1980

MULTIPROGRAMIRANJE IN MULTIPROCESIRANJE NA VELIKIH RAČUNALNIŠKIH SISTEMIH BURROUGHS

KONRAD STEBLOVNIK

UDK: 681.3:519.682.6

TGO GORENJE, VELENJE

Članek na kratko opisuje izvedbo posla in osnovno podatkovno zgradbo na velikem Burroughsovem računalniku B6700. Opisani so tudi multiprogramski pripomočki, ki jih lahko uporabljamo v okviru sistemske programske opreme na tem računalniku.

MULTIPROGRAMMING AND MULTIPROCESSING ON LARGE SCALE BURROUGHS'S COMPUTERS.The paper describes the concept of the job and basic data structure for Burroughs's large scale computer B6700. Multiprogramming facilities, wich can be used in the environment of sistem software of this computer, are described also.

UVOD

V letu 1961 je združenje Burroughs pričelo proizvajati vrsto obsežnih računalniških sistemov, katerih organizacija se je radikalno razlikovala od konvencionalnih von Neumanovih strojev. Razvoj se je pričel s sistemom imenovanim B5000. Njegova modifikacija je bil B5500. V letu 1969 je ta sistem doživel veliko novosti pod imenom B6500. Še kasneje sta ta dva sistema preimenovana v B6700. Današnja izvedba velikih Burroughsovih računalniških sistemov je B6800. Našteta generacija računalnikov Burroughs se bistveno razlikuje od ostalih tako po organizaciji kot programski strojni opremi. Isti sistemi so bili med prvimi, ki so nudili učinkovito multiprogramiranje in multiprocesiranje - cilj današnjih velikih računalniških sistemov in tudi večine manjših. Načrtovalci teh sistemov so uporabili za algoritme operativnega sistema blokovno zgradbo. Te algoritme so zapisali v razširjenem programskem jeziku Algol 60, ki lahko opisuje takšno blokovno zgradbo. Čeprav ni namen tega zapisa opis strojne arhitekture sistema B6700, naj le omenimo najzanimivejšo posebnost - vhodno izhodni podsistem ter njegova povezava, po kateri se pretakajo informacije (V/I crossbar matrika) med statičnimi moduli (pomnilnik) in aktivnimi enotami (procesorji).

Članek opisuje in navaja multiprogramske pripomočke, ki jih lahko uporabljamo v okviru sistemske programske opreme na računalniku B6700, kakor tudi na vseh ostalih računalnikih Burroughs, ki spadajo k tej družini. Razširjeni programski jezik Algol 60, [1] ga navaja tudi pod imenom Burroughsov Algol, vsebuje naslednje multiprogramske pripomočke: dogodki (event), procesi (process, task), pridevki procesov (task attribute), programske prekinitve (software interrupt) in kritičen del (procure .... liberate). Z uporabo teh elementov multiprogramiranja (multitasking) lahko sočasno izvajamo na enem ali več dejanskih procesorjih, ki so vgrajeni v aparaturno opremo, več sočasnih procesov (taskov). Sistem, ki ga bomo opisali, ima zelo svojsko izvedbo posla (job) in pripadajočo podatkovno zgradbo. Na kratko bomo prikazali poleg omenjenih multiprogramskih pripomočkov, tudi posel tega računalnika, ki se lahko izvaja na enem ali več procesorjih, ter pripadajočo podatkovno zgradbo.

1. POSEL V RAČUNALNIKU B6700

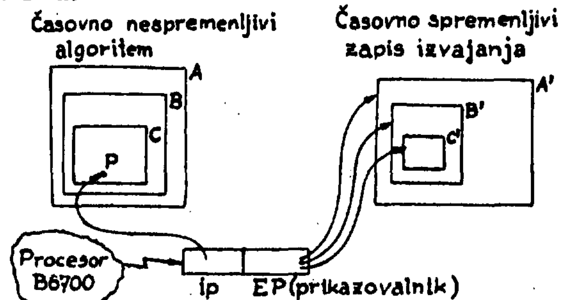
Posel tega računalnika sestavljata časovno nespremenljivi algoritem in časovno spremenljiva podatkovna zgradba, ki je zapis izvajanja tega algoritma. Algoritem sestavlja skupek nespremnljivih zakodiranih segmentov, ki so neposredno naslovljivi.

Zapis izvajanja je mnogonamenska podatkovna zgradba, ki v kateremkoli trenutku podaja:

a) stanje izvajanja posla vključno z vrednostmi za vse spremenljivke;
b) naslovljivo okolje, ki ga stvarni procesor lahko dosega, ko izvaja posel. Možno je doseganje več okolij;
c) nadzor nad preteklim sodelovanjem med bloki, postopki in procesi (taske).

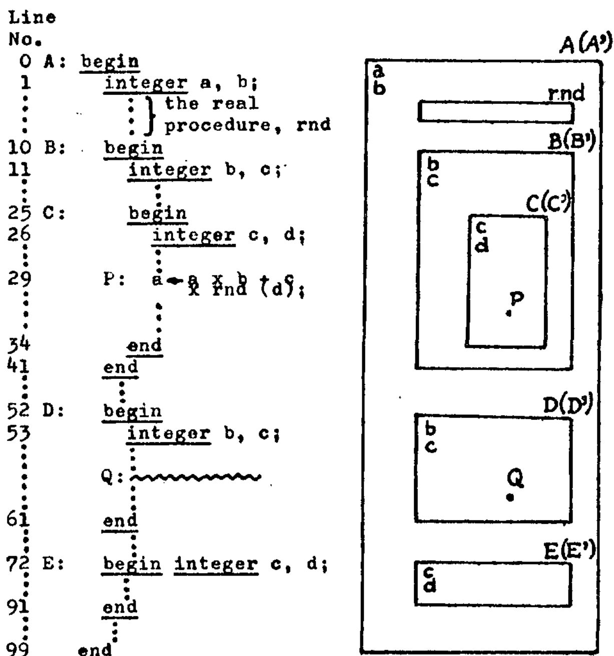
Dejanski procesor izvaja posel s pomočjo ukaznega kazalca (ip) in kazalca na okolje (ep), ki definirata stanje izvajanja.

Slika 1 prikazuje posel v trenutku, ko procesor izvaja ukaz P na katerega kaže ukazni kazalec. Kazalec na okolje EP (prikazovalnik) deluje kot vektor kazalcev, ki kaže na področja zapisa C', B' in A'. A', B', C' so podatkovna območja, ki se dodeljujejo blokom A, B, C ob vsakokratnem pozivu (izvajanju) bloka. Takšna zgradba odgovarja visokim programskim jezikom (ALGOL 60, PL/1) z blokovno zgradbo. Dostopno okolje za procesor deluje kot unija dostopnih področij C', B' in A'.Gnezdenje teh področij odgovarja gnezdenju programskih blokov A, B in C, kateri vsak definira območje dosega programske liste imen.



Slika 1. Skica posla v računalniku B6700.

Lep primer programa, napisanega v programskem jeziku Algol in odgovarjajoče skice zgradbe, je na sliki 2. Takšna skica zgradbe odgovarja tako algoritmu, kot zapisu izvajanja. Posamezna področja E', D', C', B' in A' vsebujejo celice z ustrezno listo imen (npr. področje C vsebuje listo c, d). Te celice so dostopne po imenu.



Algolski tekst Struktura vgnezdenja

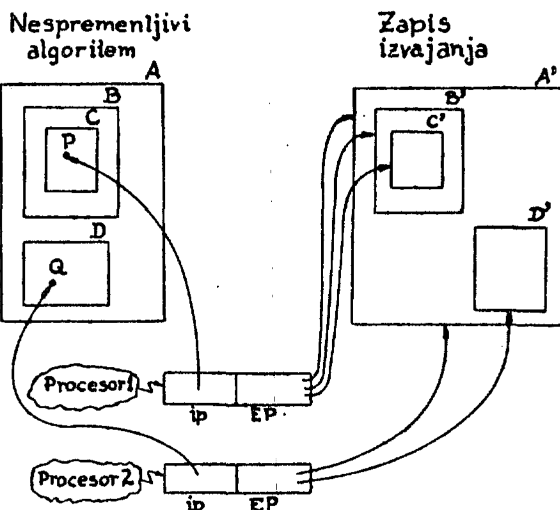
Slika 2. Program zapisan v programskem jeziku Algol in odgovarjajoča skica zgradbe.

Za primer, kako deluje takšna zgradba, si oglejmo trenutek, ko se izvaja ukaz P in zahteva dostop do celice z imenom a. Izvede se pregledovanje zapisnih področij C', B' in A' in sicer v smeri od C' proti A' (to pregledovanje seveda ne vključuje področij D' in E'). Takoj, ko se najde celica z imenom a, se pregledovanje zaključi. Podobno pregledovanje se izvede za celico b v področju B's tem, da so v trenutku izvajanja ukaza P celici b v področju A' in področju D' "nevidni" za procesor. To pregledovanje se izvršuje prek statičnih povezav med posameznimi področji.

V primeru, da postopke (procedure) izvajamo kot procese, ne pa kot podprograme, doseže program t.i. drugi nivo aktivnosti (site of activity).

V računalnik B6700 je lahko vgrajenih tudi dva ali več procesorjev. Slika 3 prikazuje koncept izvajanja procesa; prikazan je shematično. Slika kaže, kako si dva procesorja razpodelita isto kodo in isti zapis izvajanja (pravzaprav del zapisa). Ker je področje A skupno okolju obeh procesorjev, je potrebno izvesti nekatere konstrukte za dosego vzajemne izključenosti (mutual exclusion). Program, ki ga lahko izvajata dva ali več procesorjev, se imenuje algoritem z večkratno aktivnostjo (multiple activity algorithm).

Programu, ki ga lahko izvaja več procesorjev, ni potrebno za učinkovito izvajanje dodeliti več procesorjev. Enojen procesor je lahko razdeljen na več nivojev aktivnosti. V tem primeru lahko en procesor upoštevamo kot virtualen ali psevdoprocesor in je potrebna za izvajanje posla le učinkovita politika razvrščanja (scheduling). V B6700 lahko virtualni procesor zamenjamo z dejanskim procesorjem (in seveda obratno) z enostavnim ukazom.



Slika 3. Skica posla v računalniku B6700, ki ga izvajata dva procesorja.

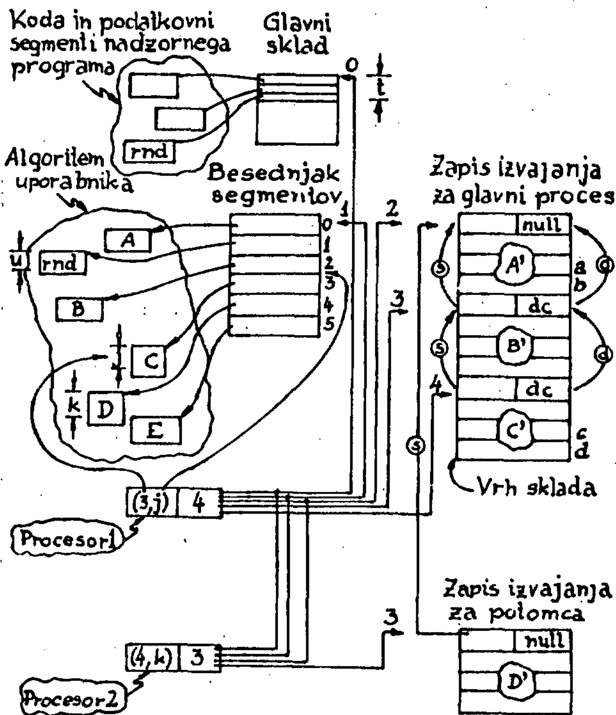
Skica na sliki 3 prikazuje posel v trenutku, ko se izvajata dva procesa na B6700. Glavni proces izvaja ukaz P z dostopnim okoljem C', B' in A', medtem, ko njegov potomec, ki je bil klican v neki prejšnji točki izvajanja glavnega procesa, izvaja ukaz Q z dostopnim okoljem D' in A'.

2. OSNOVNA PODATKOVNA ZGRADBA ZA ALGORITEM B6700

V tem poglavju bomo na kratko opisali podatkovno zgradbo za B6700, ki temelji na principu sklada. Nekatere osnovne pojme si bomo razjasnili ob sl. 4, ki prikazuje algoritem in odgovarjajoči zapis izvajanja iz slike 2. Sestoji se iz dveh skladov, besednjaka segmentov in glavnega sklada. Prikazovalna vektorja za oba procesorja kažeta na naslovno okolje (tudi skladovna zgradba), katerih vrh sta zapisa za C' in D' na nivojih 4 oz. 3.

Koda za B6700 algoritem je razdeljena na bloke in koda za vsak blok je shranjena kot fizično ločen segment. Vsak vstop v besednjak segmentov se izvede s pomočjo segmentnega kazalca. V fizično naslovljivem pomnilniku morajo biti prisotni samo segmenti, ki so dejansko potrebni aktivnemu procesorju. Prisotnost segmenta definira bit prisotnosti. Besednjaku segmentov pripada še odgovarjajoči osnovni naslov. Ukazni kazalec je dvojen: prva komponenta (v tem primeru 3) je odmik znotraj besednjaka segmentov in druga komponenta (j) je odmik znotraj segmenta. Ukazni kazalec ima lahko tudi tri komponente, kar bomo spoznali pozneje. Osnovni naslov besednjaka segmentov se določi prek kazalca, ki je del prikazovalne povezave. Kadarkoli izvajanje programa zahteva nov blok, se dodeli segmentnemu skladu nov aktivirani zapis, ki se doda vrhu sklada. Na predhodni zapis se naveže s pomočjo povezovalne besede na dva načina: statično in dinamično. Statična povezava definira gnezdenje okolja, dinamična pa je v tem trenutku navezovanja kar enaka statični. Dinamične povezave zagotavljajo procesorju vse potrebne informacije za pravilno dodeljevanje in opuščanje aktiviranih zapisov.

Področje z imenom glavni sklad opisuje delovanje sistemskega nadzornega programa in med drugim vsebuje zapis za vse kode nadzornih segmentov in sistemske tabele. Segmenti tega sklada so tudi dostopni prek prikazovalnega kazalca na nivoju 0.



Slika 4. Podatkovna zgradba za računalnik B6700.

Procesor potrebuje seveda še nekaj dodatnega začasnega pomnilnika za začasno pomnenje vmesnih rezultatov npr. pri razvijanju ukazov. V ta namen uporabi poseben operativni sklad za vsak virtualni procesor.

Program na sliki 2 vsebuje tudi postopek z imenom "rnd". Ko procesor prične izvajati ta postopek, se prek dinamične povezave naveže na zapis izvajanja nov aktivirani zapis, ki odgovarja postopku "rnd". Ta zapis vsebuje poleg imen začasnih spremenljivk še povratno informacijo. Ukazni kazalec se v tem primeru sestoji iz treh komponent (1, 1, u). Prvi element vodi do tabele, ki vsebuje opis segmenta ukazov (1 pomeni besednjak segmentov); naslednja dva elementa pa pomenita odmik znotraj besednjaka segmentov (1) in znotraj samega segmenta (u). Aktivirani zapis za "rnd" je dinamično povezan na blok C in statično povezan na blok A, ker je postopek "rnd" definiran v bloku A.

Predpostavimo, da je sedaj "rnd" ali pa katerikoli drugi del programa definiran kot sistemska operacija; opis segmenta, ki vsebuje njegovo kodo, pa je del glavnega dela sklada. V zapisu izvajanja se v trenutku vstopa v ta segment tvori odgovarjajoči aktivirani zapis, ki se dinamično naveže na blok C in je statično povezan z glavnim skladom. Ukazni kazalec se sestoji tudi iz treh komponent (0,t,u):
0 - pomeni glavni sklad, t odmik znotraj sklada in u odmik znotraj segmenta.

V primeru klica procedure (rnd) prvi element v ukaznem kazalcu pove, ali je ta procedura definirana na nivoju našega algoritma ali na nivoju sistemskega nadzornega programa.

B6700 vključuje tudi prekinitve, ki se obravnavajo kot nepričakovani podprogramski klici (postopki).

3. KOMUNIKACIJA MED SEKVENČNIMI PROCESI

Na računalniškem sistemu se izvajajo razni procesi, ki so lahko stalni ali pa začasni. Med njimi obstajajo določene zveze v času in prostoru. Glede na to jih razdelimo v tri vrste:

- neodvisne procese
- paralelne procese
- komunicirajoče ali odvisne procese.

Nas zanimajo predvsem odvisni procesi, ki lahko spreminjajo informacije, ki so jim skupne in so dalje razdeljeni v medlo odvisne procese in sodelujoče procese. Odvisni procesi si morajo deliti različne računalniške vire in jih je potrebno zaradi tega sinhronizirati ter onemogočiti sočasno uporabo teh virov (vzajemna izključenost). S tem v zvezi se pojavi problem kritičnega dela procesa. Ta del procesa je kritičen zaradi tega, ker se znotraj njega, ob uporabi določenega vira spreminja ali dosega informacija, ki je skupna več procesom. V literaturi zasledimo več načinov reševanja problemov, ki nastopijo ob takšni medprocesni komunikaciji. Najbolj znani pobudniki za reševanje teh problemov so Dijkstra, Hoare in Hansen, ki so uvedli metode strukturiranega programiranja na tem področju in s tem v zvezi naslednje pojme:

- binarni in splošni semafor
- pogojno kritičen del procesa
- monitor ali tajnik in
- hierarhija sekvenčnih procesov in monitorjev.

Omenimo naj še multiprogramski jezik Modula [2], ki ga je definiral Wirth in je namenjen za programiranje sprotnih sistemov ter krmilnih programov za vhodne/izhodne naprave.

Na te splošne ugotovitve lahko navežemo teorijo o sestavljanju družine procesov na velikih Burroughsovih sistemih. Ta teorija sloni na razširjenem Burroughsovem programskem jeziku Algol.

3. 1. Klicanje in vsklajevanje procesov

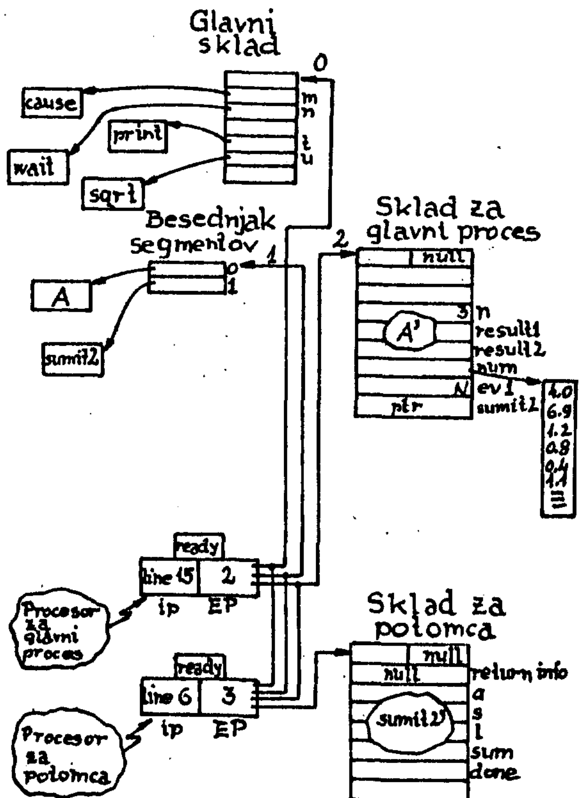
V tem poglavju bomo opisali, kako program doseže takoimenovani drugi nivo aktivnosti; to pomeni, da program v določeni točki kliče proces, ki lahko prevzame del njegovega opravila ter ga izvaja do določenega trenutka asinhrono in neodvisno, vendar sočasno. Algoritem za program, kjer glavni proces sproži svojega potomca (offspring task), je napisan na sliki 5.

```
Line
No.
 0  begin
 1    integer j, n: real result1, result2;
 2    array num 0:99 ;
 3    event ev1, null: comment deklaracija
                          dogodkovnih spremenljivk;
 4    procedure sumit2 (a, s, 1, sum, done);
 5      value s, 1; integer s, 1; real sum;
                   array a[*]; event done;
 6      begin
 7        integer i;
 8        sum ← 0;
 9        for i ← s step 1 until 1 do
          sum ← sum+sqrt (a[i]);
10        cause (done); comment sistemska operacija;
11      end sumit2
12  ;
13   [input value for n  50 and {num;, for j=0step
      1 until 2x n - 1}]
14   process sumit2 (num, n, 2 x n - 1, result2,
     ev1); comment glavni proces sproži svojega
              potomca;
15   sumit2 (num, 0, n - 1, result1, null);
     comment glavni proces kliče proceduro sumit2;
16   wait (ev1); comment sistemska operacija;
17   print (result1 + result2); comment sistemska
                                 operacija;
18  end
```

Slika 5. Algoritem za program v Burroughsovem Algolu, kjer določeno opravilo izvajata dva istočasna procesa.

Na sliki 5 je v vrsticah 3, 5, 10, 14, in 16 večina novih sintaktičnih enot, ki omogočajo enostavno vsklajevanje med procesi. V vrstici 3 je deklarira-

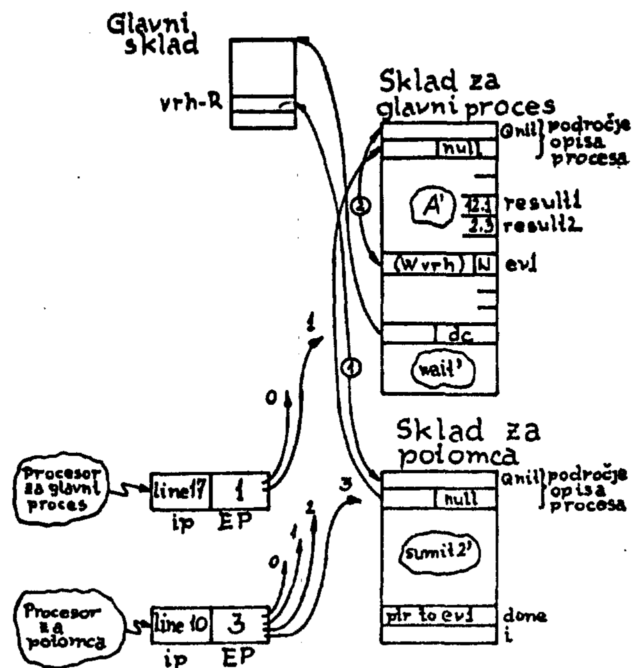
na spremenljivka "ev 1" tipa event (dogodek) kot osnova za vsklajevanje oziroma sinhroniziranje. V vrstici 5 je deklariran prilagoditveni formalni parameter z imenom "done" za postopek sumit2. Burroughsov Algol uvaja tudi ključno besedo proces, da razlikuje klic procesa (vrstica 14) od običajnega klica postopka (vrstica 15). Klic procesa posreduje postopku sumit2 dejanski parameter, da lahko potomec preko njega signalizira glavnemu procesu, da je zaključil svoje opravilo. To opravi prek sistemske operacije cause (vrstica 10). Potem, ko je glavni program sprožil proces v vrstici 14, kot svojega potomca, izvede običajni klic postopka sumit2, ki posreduje referenco null kot sintaktičnega statista za formalni parameter done. Po izvedbi postopka pokliče sistemsko operacijo wait, ki se izvaja toliko časa, dokler dejanski parameter ev1 ne doseže vrednosti, ki se lahko interpretira kot: "dogodek se je zgodil" (N-not happend, H-happend). Po zaključku sistemske operacije wait algoritem pokliče operacijo print za izpis vsote dveh vrednosti result1 in result2. Slika 6 prikazuje podatkovno zgradbo za algoritem iz slike 5 v trenutku, ko glavni program izvaja klic postopka sumit2 v vrstici 15, njegov potomec pa vrstico 6.



Slika 6. Podatkovna zgradba za algoritem iz slike 5

Spremenljivke tipa event so strukturirane in eno polje v tej zgradbi je binarno stikalo, ki se imenuje dogodkovni bit. Začetna vrednost tega bita je: "dogodek se ni zgodil" (N) in operacija cause ga postavi na vrednost: "dogodek se je zgodil" (H).

Slika 7 prikazuje dejansko vključitev naših procesov v sistem B6700 in v njegov nadzorni program.



Slika 7. Dejanska vključitev algoritma iz slike 5 v sistem računalnika B6700.

Sklad za vsak nov proces se prične s področjem opisa procesa (task description area), ki ima stalno širino. Temu sledi prvi zapis aktiviranja. Kombinacija aparaturne opreme in sistemskih programov zagotovi, da je to področje dostopno samo nadzornemu programu. Eden od ključnih delov informacije tega posebnega področja je nit (thread - Q), s pomočjo katere se proces lahko naveže na glavni seznam (list head), ki definira stanje razvrstitve (queue state) procesa. Če je proces pripravljen (ready), je nit Q navezana (črta označeno z 1 na sliki 7) na vrh (head - R) v glavnem skladu. Z uporabo takšne povezave, lahko sistemski nadzorni program izbira procese, ki se naj izvajajo.

Povratni naslov za vsak proces je na sliki 6 označen z null. To bi pomenilo, da preide vsak proces ob svojem zaključku na izvajanje stavka null. Dejansko je na sistemu B6700 podana namesto vrednosti null vstopna točka v sistemsko nadzorno proceduro.

Zapis aktiviranja za sistemsko proceduro, ki zaključi proces, se doda glavnemu skladu.

3. 2. Pridevki procesov

Da dosežemo večji nadzor in/ ali možnost sodelovanja znotraj družine procesov, so vsakemu procesu dodeljene strukturirane spremenljivke, katere elementi definirajo različne lastnosti procesov.

V času sprožitve procesa mora biti deklarirana spremenljivka tipa task. Zaradi tega je algoritem na sliki 5 sintaktično nepravilno zapisan. Pravilen zapis vrstice 14 tega algoritma bi bil v Burroughsovem Algolu tale:

process sumit 2 (num, n, 2 x n-1, result2, evl) [charlie];

če je Charlie ime tega procesa.

Predhodno pa je potrebno deklarirati še spremenljivko Charlie tipa task:

task charlie

Spremenljivka tipa task ima stalno zgradbo in je dodeljena v informacijsko področje procesa kot del

sklada procesa v času, ko se proces kliče.

V času, ko je proces klican, dobijo njegovi pridevki neko začetno vrednost. Ta začetna vrednost lahko kasneje v času, ko je proces aktiven, ostane nespremenjena, lahko jo spremeni proces sam ali pa določeni drugi procesi iz družine, navadno procesi, ki ga kličejo.

Ko je sprožen proces, se lahko nanj predhodno navežejo različni pridevki. V primeru procesa Charlie, bi lahko zapisali takšne pridevke na tale način:

12.1 Charlie. priority ← 5;
12.2 Charlie. maxproctime ← 20; comment in seconds;
12.3 Charlie. stacksize ← 2^8;

Skupek pridevkov procesa, ki so zapisani v vrsticah 12.1, 12.2 in 12.3, razpozna sistem, začetne vrednosti pa postavi proces, ki ga je sprožil. Te vrednosti upošteva sistemski algoritem za razvrščevanje ali drugi moduli za upravljanje z viri sistema.

Posebne medsebojne odnose med različnimi procesi dosežemo z nekaterimi ključnimi pridevki procesov. Opisali bomo štiri različne pridevke tega tipa: status, exeptiontask, exeptionevent in partner.

Pridevek status opisuje trenutno stanje izvedbe določenega procesa. Proces je lahko razvrščen, aktiven, začasno ustavljen, zaključen. Ta pridevek je dostopen kateremukoli procesu v družini, za katerega je ime procesa (Charlie) vidno, to je procesu, ki ga je sprožil ali kateremukoli nasledniku, za katerega je Charlie globalen. Takšni procesi lahko prek spremenljivke status vsilijo določeno stanje procesu Charlie: lahko ga začasno ustavijo, zaključijo itd. Npr.

myself. status ← suspended

Proces, ki je sprožil proces (npr. Charlie ali katerikoli drug proces, ki lahko "vidi" spremenljivke procesa Charlie), lahko deluje kot Charliejev nadzornik ali "starejši brat", ker takšen proces (razen Charlieja samega) lahko spreminja ali čita njegove pridevke. Če je npr. proces Pete sprožil Charlieja, ga lahko aktivira, začasno ustavi ali pa zaključi in tako lahko nadzoruje njegovo stanje prek pridevka Charlie. status. S pomočjo pridevka exceptiontask pa lahko kasneje prenesemo funkcijo nadzora nad procesom Charlie, ki ga je od začetka imel Pete in sam Charlie, na drug proces z naslednjim stavkom:

myself. exceptiontask ← brothertom;

Na ta način se lahko dva procesa povežeta v zaključeno verigo.

S pomočjo pridevka partner lahko omogočimo, da dva procesa delujeta sinhrono kot sosledni operaciji. Procesa A in B lahko delujeta kot sosledni operaciji potem, ko priredimo: A. partner = B in B.partner = A. Če sedaj proces A izvede stavek

continue;

se proces A ustavi in proces B se prične izvajati tam, kjer se je ustavil ob nastopu continue stavka. Trije ali več procesov lahko delujejo kot sosledne operacije tako, da npr. tvorijo obroč:
A.partner ← B, B.partner ← C; C.partner ← A.

Kadar se zahteva kakršnakoli operacija nad temi pridevki, sistemski razvrščevalnik odloči, kdaj se bo to izvršilo. Kasneje lahko samo procesi, ki so obveščeni o tej operaciji, izvršijo nasprotno akcijo.

3. 3. Primer

V prejšnjem poglavju smo podali semantiko spremenljivk procesov in pridevkov procesov in sedaj z njihovo pomočjo rešimo naslednji problem.

Predpostavimo, da imamo tri identične krmilnike (controller) pretoka, ki lahko sodelujejo med sabo in vplivajo drug na drugega. Vsak krmilnik lahko nadzoruje pretok, ki si ga zamislimo kot pretok olja, vode ali zrnatega materiala ali pa kot pretok števil. Vsak krmilnik dovoljuje nadzorovani pretok v zbiralnik ter pozna nakopičeno vrednost v zbiralniku, ker lahko stalno meri vrednost pretoka. Naloga vsakega krmilnika je, da naj v sodelovanju z ostalimi dovoljuje približno enako velikost pretoka, kot se trenutno prevaja preko ostalih. Za zapis računalniškega programa, ki ustreza takšni situaciji, predpostavimo da se pretoka zaporedje celih spremenljivk, ki se iz vhodne datoteke čitajo kot konstantne vrednosti. Za poenostavitev predpostavimo, da krmilni program A meri pretok enostavno tako, da generira vsoto celih vhodnih spremenljivk. To vsoto imenujemo SA. Krmilni program A pozna nakopičeno vrednost v ostalih dveh zbiralnikih, ker ima dostop do generiranih vsot SB in SC ostalih dveh krmilnikov.

Prav tako lahko A periodično nadzoruje pogoj:

SA > SB and SA > SB = true

in če je pogoj izpolnjen, lahko prekine pretok v svoj zbiralnik. Predpostavimo, da lahko krmilni program A učinkovito sporoči najmanj enemu od ostalih dveh to, da je začasno ustavljen. Če pa pogoj ni izpolnjen, pa A nadaljuje s pretokom v svoj zbiralnik. Ko krmilni program A ugotovi, da je pogoj SA > INMAX izpolnjen, lahko zaključi z izvajanjem in preneha obstajati. Ta opis velja seveda za vsak krmilnik A, B in C.

Možnih je več rešitev. Vsakemu krmilniku priredimo lasten proces in ti procesi sodelujejo med sabo. Lahko bi jih tudi izvedli kot sosledne operacije. Na sliki 8 je algoritem, ki uporablja procesna pridevka status in exeptiontask. Ko dani krmilnik ugotovi, da prehiteva ostala dva krmilnika, ju obvesti o tem vedno po round robin algoritmu: A ← B, B ← C in C ← A.

Proces, ki je začasno ustavil samega sebe (vrstica 12) lahko ponovno sproži samo drug proces, ki je njegov izjemni proces (exeptiontask), to je na sliki 8 v vrstici 11.

Opisane spremenljivke procesov in njihovi pridevki za nadzor in komuniciranje med procesi niso vse. Literatura [1] navaja, da obstaja še več drugih pridevkov procesov npr. pridevek locked tipa Boolean ali pridevek taskvalue tipa real. Ta kratek pregled nas lahko prepriča, da ima programer močno orodje za tvorbo kompleksnih podsistemov v obliki družine procesov.

Sintaktične konstrukte, ki smo jih spoznali v tem poglavju, lahko na kratko primerjamo z rešitvami, ki so jih navedli drugi avtorji na področju multiprogramiranja. Pripomoček imenovan proces, je podan tudi v programskem jeziku Modula [2] in je podoben proceduram, s tem da se izvaja sočasno s programom (oziroma procesom), ki ga je klical. Procesi so v Moduli lahko inicializirani samo v glavnem procesu in ne morejo biti vgnezdeni.

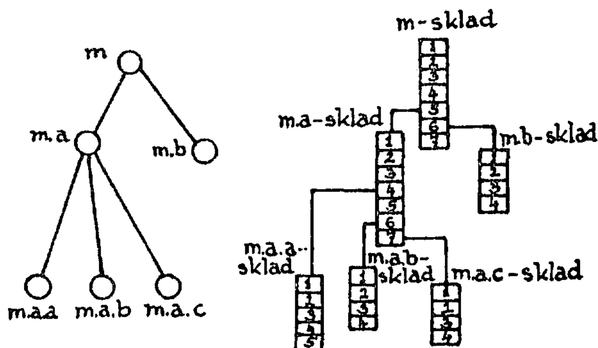
Sinhronizacijo med procesi lahko v našem primeru dosežemo s spremenljivkami tipa event kot argumente operacij "wait" in "cause" (čakaj na dogodek, povzroči dogodek). V programskem jeziku Modula lahko primerjamo s tema dvema operacijama operaciji "wait" in "send" (čakaj na signal, pošlji signal), medtem ko bi dogodku odgovarjala spremenljivka tipa signal. Signalu pa pri Hoarju [4] odgovarja pojem pogoja, pri Hansenu pa pojem vrste.

Pri Moduli je vpliv na potek procesov dosežen z uporabo signalov in skupnih oziroma deljenih spremenljivk. Na B6700 lahko v Burroughsovem Algolu dosežemo takšen vpliv z dogodkovnimi spremenljivkami in pridevki procesov. Kritični deli procesov, ki delujejo nad skupnimi spremenljivkami, so v Moduli de-

```
Line
No.
 1  begin
 2    integer SA,SB,SC, INMAX; task A, B, C;
                               event doneABC;
 3    procedure controller1 (s1, s2, s3, n,done);
 4      value n; integer s1, s2, s3, n; event done;
 5      begin
 6        integer VAL; label L;
 7        L: If s1 > s2 and s1 > s3
 8          then begin
 9            [print values of s1, s2, and s3 in
             appropriate columns
10              of a table, based on the value
              of n];
11            (myself.exeptiontask).
            status ← "wakeup";
12            myself.status ← "suspended";
13            go to L; end
14          else begin
15            [input a value of VAL from input
             file n];
16            s1 ← s2 + VAL;
17            if s1<INMAX then go to L
            else cause(done);
18            end
19      end controller1
20    ;
21    A.exeptiontask ← B;
22    B.exeptiontask ← C;
23    C.exeptiontask ← A;
24    [input a value for INMAX];
25    SA ← SB ← SC ← 0;
26    process controller1(SA,SB,SC,1,doneABC)[A];
27    process controller1(SB,SA,SC,2,doneABC)[B];
28    process controller1(SC,SA,SB,3,doneABC)[C];
29    wait(doneABC);
30  end
```

Slika 8. Program za krmiljenje pretoka števil v tri zbiralnike.

klarirani kot vmesniške procedure in so zbrani v vmesniških modulih (monitor pri Hoaru in Hansenu). V našem primeru se take operacije izvajajo pod kontrolo sistemskega nadzornega programa, ki poskrbi, da ne pride do konfliktnih situacij. S pridevki procesov lahko torej v nasprotju z Modulo kontroliramo potek procesov. Kasneje bomo spoznali še eno možnost vpliva na potek procesov, to je s programskimi prekinitvami.



Slika 9. Drevesna zgradba družine procesov.

V nasprotju z Modulo lahko v Burroughsovem Algolu tvorimo družino procesov, ki imajo drevesno zgradbo. Takšen, zelo poenostavljen primer je prikazan na sliki 9. Na tej sliki je na levi strani drevesna struktura, na desni pa poenostavljena povezava posameznih zapisov izvajanja, ki imajo skladovno zgradbo in jih sestavljajo posamezni zapisi aktiviranja, ki so statično povezani. Prav tako so statično povezani posamezni procesi in sicer tako, da je npr. proces m.a.a. statično navezan na m.a. sklad na zapise aktiviranja 4,3,2 in 1 ne pa na 5,6 in 7. Zapis aktiviranja ali blok s številko 4 v m.a. skladu se imenuje kritičen blok za m.a.a. sklad. Ko se proces m.a. konča ali se kako drugače neha izvajati, njegov naslednik ne more več eksistirati, ker nima več dostopa do okolja svojega predhodnika. Zaradi tega mora proces, ki konča, končati tudi vse svoje potomce.

3.4. Programske prekinitve

Programska prekinitev ima nekatere podobnosti z ožičeno prekinitvijo. Izvor programske prekinitve se nahaja v nekem drugem procesu iz družine procesov in proces, kateremu je prekinitev namenjena, preide, ko prejme prekinitev, na izvajanje neke v- naprej definirane procedure (prekinitvene procedure). Prejemnik prekinitve jo lahko ignorira tako, da začasno onemogoči programsko prekinitev. Če se pa proces, kateremu je namenjena prekinitev, trenutno ne izvaja, sistemski nadzorni program poskrbi, da se prekinitveni signal postavi v vrsto in se obravnava v trenutku, ko se odgovarjajoči proces prične ponovno izvajati.

Prekinitev deklariramo v B6700 Algolu z naslednjim stavkom:

interrupt (name of interrupt procedure);(statement);

Če zapišemo:

interrupt i1; begin . . . end;

pomeni, da je to prekinitvena procedura z imenom i1

Prekinitvena procedura se lahko naveže na dogodkovno spremenljivko s stavkom:

attach (name of interrupt procedure) to (event variable);

Primer:

```
1  event ev;
2  interrupt i1; begin ... end;
3  attach i1; to ev;
4  enable i1;
```

V vrstici 1 je deklarirana dogodkovna spremenljivka, ev; vrstica 2 deklarira prekinitveno proceduro z imenom i1; vrstica 3 naveže prekinitveno proceduro i1 na dogodkovno spremenljivko ev; vrstica 4 pa omogoči prekinitev, ki se povzroči preko spremenljivke ev.

Prekinitev lahko onemogočimo s stavkom:

disable i1;

Na začetku se privzame, če ne vključimo stavka enable, da je prekinitev omogočena.

Dogodkovna spremenljivka se lahko onemogoči s stavkom:

detach i1;

nakar lahko navežemo na prekinitev i1 katerikoli drug dogodek.

3.5. Doseganje virov

Kot smo že na kratko omenili, si morajo procesi, ki se odvijajo na nekem računalniškem sistemu, učinkovito razpodeliti uporabo računalniških virov. V času, ko proces zahteva določen vir, se mora izvesti kritičen del programa, ki ga uvaja Dijkstra [3]. Predhodno smo dogodke opisovali kot dvostanjske s stanji: "dogodek se je zgodil" in "dogodek se ni zgodil". Na sistemu B6700 lahko dogodke interpretiramo še na naslednji način: " dogodek je na razpolago" ali "dogodek ni na razpolago".

Proces, ki izvaja stavek:

```
procure(ev);
```

je začasno ustavljen, če dogodek ev ni na razpolago, v nasprotnem pa izvede nasljednji stavek zaporedja, ki sledi. Kritičen del se v Burroughsovem Algolu zapiše na nasljednji način:

```
procure(ev);
    —
    —
liberate(ev);
```

Operaciji "procure" in "liberate" torej lahko primerjamo z operacijama "P" in "V", ki se izvajajo nad semaforjem in jih je uvedel Dijkstra.

SKLEP

Iz tega zapisa in nekaterih opisanih primerov vidimo, da so načrtovalci programske opreme Burroughsovih računalnikov že zelo zgodaj razvili učinkovite pripomočke za multiprogramiranje na računalnikih, ki so tudi multiprocesorski. Čeprav so verjetno svojo opremo razvijali ločeno od ostalih teoretikov multiprogramiranja, lahko napravimo primerjavo med posameznimi rešitvami.

LITERATURA

[1] Elliot J. Organick: Computer System Organisation, Academic Press 1973, New York and London.

[2] N. Wirth: Modula: a Language for Modular Multiprogramming, Software-Practice and Experience, vol. 7, 335 (1977).

[3] Dijkstra E. W.: Cooperating Sequential processes, in Programming languages, ed. Genuys, Academic Press 1968.

[4] C. A. R. Hoare: Monitors-an Operating System Structuring Concept, Com. of the ACM, vol. 17, no. 10 (Oct. 1974).