Mr Ratimir Kvaternik
Fakultet organizacije i informatike
V a r a ž d i n

UDK 681.142.2
Prethodno saopćenje

P R I M J E R S I N H R O N I Z A C I J E P A R A L E L N I H P R O C E S A U P O T R E B O M S E M A F O R A I O P E R A C I J A N A S E M A F O R I M A

Medjusobni odnosi medju procesima u kompjuterskom sistemu moraju se regulirati na nivou operativnog sistema. Pomoću operacija poznatih pod nazivom WAIT i SIGNAL na varijablama koje zovemo "semafori" moguće je provesti jednostavno i uspješno sinhroniziranje dvaju ili više procesa, što je prikazano na primjeru sinhronizacije procesa tipa "proizvodjač-potrošač". Važno je napomenuti da postoji mogućnost formalnog dokaza točnosti rješenja, što je u ovom primjeru takodjer učinjeno.

Ovakvim rješenjem sinhronizacije procesa pomoću semafora i operacija na semaforima uspješno se rješava problem zaposlenog čekanja (busy wait) što nije moguće dobiti drugim metodama.

1. POJAM PROCESA

Razmatranje vezano uz kompjuterski sistem, gledano sa stanovišta korisnika kompjuterskog sistema, dakle izvana, moguće je provesti promatrajući izvodjenje logičkih jedinica cjelokupnog rada, nazvanih "job" (posao). Svaki se pak job može sastojati od jednog ili više programa, a svaki program od niza instrukcija ili naredbi. Želimo li, medjutim, kompjuterski sistem promatrati kako radi interno, dakle kako se odvijaju i koji su sve postupci potrebni da bi se dobilo ono što se prema korisniku manifestira kao "izvodjenje joba", a posebno ako se radi o dizajniziranju samog operativnog sistema, potrebno je uvesti neke pojmove i tehnike prikaza tog internog rada.

Moderni kompjuterski sistemi uključuju u svom radu niz paralelnih odvijanja raznih postupaka. Paralelnost je moguća zbog velike brzine rada, a posebno zbog velike brzine centralne jedinice. S druge strane paralelnost je i vrlo potrebna u sistemu zbog uspješnog korištenja njegovih osnovnih resursa. Jedan korisnik nije očigledno u stanju zaposliti sistem ni uspješno ni jednoliko. Dozvolimo li da više korisnika koristi sistem istovremeno, tada moramo omogućiti i paralelno izvodjenje pojedinih postupaka. Na hardwarskom nivou paralelnost je ustvari uvijek prisutna. Na nivou operativnog sistema, a to je naš nivo promatranja, takodjer se već i kod jedne centralne jedinice javlja

niz paralelnih postupaka. Naime, kanalske jedinice kod izvodjenja U/I operacija rade stvarno paralelno s centralnom jedinicom. Zbog toga kanale takodjer smatramo procesorima. Procesorom naime smatramo kod tretiranja problemasa stanovišta operativnog sistema svaki hardwarski dio koji je u stanju interpretirati i izvoditi instrukcije. Izvodjenje jednog ulazno/izlaznog postupka jest dakle aktivnost u kompjuterskom sistemu koja se izvodi paralelno s nekim drugim aktivnostima. Svim ovim aktivnostima upravlja operativni sistem izvodjenjem svojih kontrolnih programa, što opet rezultira u odredjenim aktivnostima u sistemu.

Sve te aktivnosti mogu se odvijati paralelno u cijelosti ili u pojedinim svojim dijelovima. Takve aktivnosti nazivamo procesima, odnosno paralelnim procesima. Neki job,koji se sastoji od jednog programa, može u sistemu pobuditi više procesa (aktivnosti) da bi se dobio konačni rezultat. Ono što je dakle cjelina, gledano izvana, sasvim drugačije izgleda kad se promatra sa stanovišta operativnog sistema, odnosno internog rada kompjutera. Zbog toga je bilo i potrebno uvesti pojam procesa. Proces je dinamičkog karaktera, on je aktivnost koja se sastoji iz niza akcija, za razliku od programa (ili joba) koji je statičkog karaktera i sastoji se od niza instrukcija.

Treba napomenuti da iako se paralelnost javlja u radu kompjuterskog sistema, svi korisnički programi rade se potpuno sekvencijalno. Razloga je vjerojatno više. Paralelnost u programu bilo bi teže programirati jer bi trebalo uzimati u obzir potrebe uskladjivanja pojedinih dijelova programa i potrebe izmjene informacija medju dijelovima koji bi se mogli odvijati paralelno. To bi znatno kompliciralo izvedbu programa i otežalo upotrebu kompjutera. Nadalje, programeri za sada nisu trenirani da takve programe rade. Glavni je razlog vjerojatno ipak taj što je naš način mišljenja i rada u principu serijski, a ne paralelan.

Uvodjenje pojma procesa i paralelnosti pokazalo se u operativnim sistemima pogodnim i korisnim jer je time olakšano razumijevanje, opisivanje rada, dizajniranje i izrada operativnih sistema te,što je možda još važnije,teoretsko tretiranje problema i razvoj teoretskih postavki i rješenja.

Postoji nekoliko raznih definicija procesa. Tako na primjer u OS/360 proces se naziva "task" i definira ovako: "Task je osnovna jedinica izvodjenja za kontrolni program. U sistemu MULTICS (poznati operativni sistem razvijen za sisteme firme General

Electric - Honeywel) vrijedi definicija: "Proces je program u izvodjenju, a izvodjenje se provodi na procesoru". U operativnom sistemu T.H.E. (koji je razvio Dijkstra E.W. za firmu Philips) definicija je slijedeća: "Pravila ponašanja zovu se program. Program se izvodi na procesoru. Ono što se dogadja prilikom izvodjenja, naziva se proces". Formalizirana definicija procesa glasi: "Pretpostavimo da postoji skup cjelobrojnih registara $\{X_1, X_2, X_3 \ldots X_n \ldots\}$ Stanjem se onda smatra dodjeljivanje odredjene vrijednosti nekom registru $\{X_i\}$. Procesom se smatra skup pravila koja definiraju preslikavanje $S \rightarrow S'$, tj. preslikavanje od početnog stanja, preko slijedećih stanja, do konačnog stanja. Sva stanja od početnog do konačnog čine historijski skup stanja. Taj historijski skup stanja u potpunosti opisuje sam proces". Na primjer neki proces može biti zadan historijskim skupom $P = \{(1,2),(2,3),(4,4),(8,5)\ldots\ldots(2^i,i+1)\ldots\ldots\}$, ali proces P može takodjer biti zadan početnim stanjem $\{(1,2)\}$ i pravilima prijelaza iz svakog stanja u svako slijedeće stanje $\{(x,y)\} \rightarrow \{(2x,y+1)\}$. Ovakva definicija omogućava teoretska razmatranja nevezana za bilo koji konkretni operativni sistem, odnosno kompjuterski sistem.

2. ODNOSI MEDJU PROCESIMA

Ako u nekom sistemu imamo n aktivnih procesa i p procesora, tada dolazi u sistemu do različitih situacija. Ako bilo koji proces može biti izveden na bilo kojem procesoru i ako je pritom zadovoljena relacija:

$$p > n,$$

procesi će se odvijati paralelno. Ako, naprotiv, broj procesora bude manji od broja aktivnih procesa, tj. vrijedi relacija

$$p < n,$$

doći će do kvaziparalelnog i paralelnog odvijanja procesa. Procesi će u kratkim vremenskim razmacima koristiti procesor i svaki će napredovati pa će za promatrača s višeg nivoa (korisnika sistema) izgledati kao da se procesi odvijaju istovremeno. To zovemo kvaziparalelnim odvijanjem. Pitanja koja se odmah nameću jesu:

- Kojem procesu valja dodijeliti procesor u slijedećem koraku?
- Koliko dugo može neki proces držati procesor za sebe?

Odgovori na ova pitanja nisu sasvim jednostavni. O njima uvelike ovisi ponašanje cjelokupnog kompjuterskog sistema. Medju procesima, dakle, kao da postoji odredjena konkurentnost, natjecanje da dobiju procesor i mogu nastaviti s izvodjenjem. Takav odnos medju procesima, gdje se procesi smatraju neovisni jedan o drugom i promatra se samo njihova medjusobna natjecanja za resurse, naziva se konkurentnost procesa. S druge strane procesi mogu vrlo često biti u drugačijem odnosu, u odredjenoj ovisnosti, direktno jedan odredjeni proces prema drugom odredjenom procesu. Pod odredjenim procesom smatra se unaprijed zadani poznati proces. Može se takodjer dogoditi da procesi postanu ovisni jedan o drugom zbog nastale trenutne situacije u sistemu, što znači da njihova ovisnost nije unaprijed bila odredjena. U svakom slučaju medjusobno ovisni procesi moraju na neki način uskladiti svoje izvodjenje, u protivnom ne može se ni jedan ni drugi dalje izvoditi. Pritom može doći u nekim slučajevima i do medjusobne izmjene informacija pa govorimo o komuniciranju medju procesima. Neki autori nazivaju komuniciranjem medju procesima sve vidove njihove medjusobne ovisnosti jer dolazi do neke vrsti izmjene informacija preko operativnog sistema u svakom slučaju. Odnose medju procesima možemo svrstati u 3 osnovne vrste, i to:

. Medjusobno isključenje procesa (Mutual Exclusion)
. Sinhronizacija procesa (Synchronization)
. Zastoj (Deadlock).

Medjusobno isključenje procesa je takav odnos medju procesima koji ne dozvoljava simultano (paralelno) izvodjenje dvaju procesa u pojedinim njihovim dijelovima. Te dijelove procesa nazivamo kritičnim sekcijama. Takav odnos mora biti riješen u operativnom sistemu putem odredjenog postupka ili, kako se to najčešće naziva, putem odredjenog mehanizma. Medjusobno isključenje javlja se kada dva ili više procesa traže neki nedjeljivi resurs i da ga koriste. Svako takvo korištenje resursa, koji je nedjeljiv, predstavlja kritičnu sekciju jer samo jedan proces može obiti nedjeljivi resurs na korištenje i koristiti ga. Mehanizam za medjusobno isključenje to osigurava i kao rezultat dobivamo serijsko korištenje resursa, tj. resurs dobiva jedan proces i drži ga dok ne prestane s korištenjem resursa, zatim ga obiva drugi proces itd. Drugi vid odnosa medju procesima je sinhronizacija. Općenito gledajući procesi se unutar kompjuterskog sistema odvijaju asinhrono. Medjutim, mogu postojati dijelovi procesa, pa i čitavi procesi, čiji je rad potrebno medjusobno uskladiti, što znači da se dijelovi jednog procesa ne mogu

izvesti prije odredjenih dijelova drugog procesa ili obratno; npr. proces koji predstavlja izlazni spooler ne može početi štampati izlaz na štampaču tako dugo dok sam proces, čiji izlaz treba štampati, ne završi izlaz u spool datoteku. Operativni sistem mora sadržavati mehanizam koji će takvu sinhronizaciju ostvariti. Upravo takav jedan mehanizam bit će u daljnjem tekstu detaljnije promatran i obrazložen.

Treći odnos medju procesima, koji smo spomenuli, jest zastoj. To je ustvari pojava kada nekoliko različitih procesa traži resurse, ali ih ni jedan ne može dobiti zbog trenutne situacije u sistemu. Takav se odnos uspostavlja medju svim aktivnim procesima i ima za posljedicu zaustavljanje svih procesa, dakle zastoj u pravom smislu riječi. Takav odnos potrebno je, naravno, spriječiti, a ako je do njega došlo, valja ga riješiti da bi procesi mogli nastaviti s izvodjenjem.

Jedan od odnosa, koji se svrstava u sinhronizaciju a javlja se vrlo često u kompjuterskom sistemu, jest odnos nazvan proizvodjač-potrošač (procuder-consumer relationship). Taj se odnos sastoji u slijedećem: Skup procesa nazvanih "proizvodjači" i skup procesa nazvanih "potrošači" komuniciraju medjusobno preko memdjumemorije (buffera). Proizvodjači su ciklički procesi koji neprestano "proizvode" neku veličinu (podatak) i "odlažu" tu veličinu u buffer, dok su potrošači procesi koji ciklički ponavljaju postupak "uzimanja" veličine iz buffera i "troše" (konzumiraju) dotičnu veličinu. Tipičan primjer za ovakve procese jest kada "proizvodjači" stavljaju sadržaj za štampanje na štampaču(izlazni slog) u izlazni buffer, dakle "stavljaju liniju" u izlazni buffer, a potrošači su u tom slučaju procesi koji taj sadržaj buffera uzimaju i vrše štampanje. Zbog jednostavnijeg i svrsishodnijeg prikaza upotrebljava se posebni način prikaza procesa u tzv. pseudo-jeziku, odnosno u našem slučaju to će biti tzv. pseudo-Algol notacija, dakle notacija slična Algolu. U toj notaciji procesi se mogu prikazati ovako:

```
PROIZVODJAČ
P: BEGIN
       - Proizvedi veličinu
       - Odloži veličinu u buffer
       GOTO P
   END
```

```
POTROŠAČ
C: BEGIN
       - Uzmi veličinu iz buffera
       - Potroši veličinu
     GOTO C
   END
```

Može se vidjeti da pseudo-Algol notacija omogućava jednostavniji prikaz procesa uzimajući samo dijelove bitne za sinhronizaciju, dok ostale samo opisno navodi. Pretpostavimo nadalje da je buffer takav da može sadržavati N veličina. To je kapacitet buffera. Budući da se ovi procesi odvijaju asinhrono, ne može se predvidjeti koji će prvi započeti, niti kada će drugi uslijediti i u kojem će dijelu biti paralelni. To nije moguće predvidjeti zato jer:

- može u svakom trenutku doći do prekida koji treba obraditi;
- procesi mogu tražiti za svoje korištenje resurse koji su zauzeti od strane drugih procesa i neizvjesno je koliko će te resurse čekati;
- procesima se dodjeljuje procesor za izvodjenje,ali je nepredvidivo u općem slučaju koliko puta i koliko dugo će proces zadržati procesor.

Medjutim, komuniciranje ova dva procesa putem buffera nužno je uskladiti, sinhronizirati u pojedinim dijelovima i situacijama, jer bi u protivnom moglo doći do takve komunikacije koja nije poželjna, odnosno dozvoljena. Moglo bi se na primjer dogoditi da se dvaput odštampa isti redak ili neki propusti, ili pak da se dogodi neki drugi slični pogrešni zahvat. Da do toga ne dodje, mora u našem slučaju u bilo kojem trenutku biti ispunjeno slijedeće:

1. Proizvodjač ne može stavljati veličinu u puni buffer (jer bi uništio prethodni sadržaj koji nije pročitan).
2. Potrošač ne smije "prazniti iz praznog buffera" (jer bi u tom slučaju uzimao dva ili više puta istu veličinu).
3. Proizvodjač i potrošač ne smiju istovremeno zahvaćati u buffer (jer bi opet moglo doći do pogrešnog punjenja ili pražnjenja).

Ovdje se mora napomenuti da se pod "punim bufferom" smatra slučaj kada je napunjeno N elemenata, ali nije ni jedan uzet (pot-

rošen), a pod "praznim bufferom" kada su svi napunjeni elementi potrošeni. Naravno, budući da se radi o bufferu,"uzimanjem" sadržaj buffera se ne razara niti ne poništava pa se zbog toga može govoriti o pražnjenju "praznog"buffera, što bi uzrokovalo uzimanje već uzetog sadržaja. Isto tako punjenje "punog" buffera uzrokovalo bi uništenje neke veličine jer bi se "preko nje" upisala nova veličina.

Označimo li broj veličina stavljenih u buffer s u (upis u buffer), a broj uzetih veličina iz buffera s p (pročitano iz buffera), onda u bilo kojem trenutku promatranja mora biti ispunjena relacija

$$0 \leq u - p \leq N \qquad (1)$$

Ova relacija ustvari matematički izražava pravilo 1. i 2. Pravilo 3. nije pak ništa drugo nego medjusobno isključenje procesa u njihovim kritičnim sekcijama koje su ovdje: postupak stavljanja veličine u buffer kod procesa proizvodjača, odnosno postupak uzimanja veličine iz buffera kod procesa potrošača. Kao što će se nadalje pokazati ovakvu sinhronizaciju tipa proizvodjač-potrošač moguće je postići vrlo jednostavno i elegantno pomoću semafora i operacija na semaforima.

3. SEMAFORI I OPERACIJE NA SEMAFORIMA

Komuniciranje medju procesima moguće je riješiti raznim tehnikama. Jedna od teoretski najboljih i općeprihvaćenih tehnika je tehnika pomoću tzv. semafora i operacija na semaforima. Koncepciju semafora uvodi 1965. godine Dijkstra E.W. i od onda se stalno upotrebljavaju. Semafor nije ništa drugo nego cjelobrojna nenegativna varijabla čija se vrijednost može mijenjati pomoću operacija na semaforima. Postoje svega dvije takve operacije, i to su WAIT i SIGNAL operacija. Ako sam semafor nazovemo s, onda WAIT(s) znači slijedeću operaciju:

```
WAIT : BEGIN
         IF s ≠ 0 THEN s: = s-1
                  ELSE BEGIN
                         - zaustavi proces
                         - stavi proces u red čekanja
                           na semaforu
                         - oslobodi resurse
                       END
       END
```

Operacija koju smo nazvali SIGNAL(s) značit će slijedeći postupak:

```
SIGNAL: BEGIN
          IF red prazan THEN s: = s+1
                        ELSE BEGIN
                               - uzeti proces iz
                                 reda čekanja
                               - aktivirati proces
                             END
        END
```

Važno je napomenuti da operacije WAIT i SIGNAL moraju biti implementirane kao "nedjeljive operacije", tj. operacije kod kojih su prekidi maskirani, dakle te se operacije sigurno u cijelosti svaki puta izvode kada se jedanput počnu izvoditi. Takvu implementaciju moguće je ostvariti.

Analiziramo li što se dogadja prilikom izvodjenja WAIT i SIGNAL operacije, možemo zaključiti slijedeće: WAIT operacija zaustavlja proces i stavlja ga u red čekanja na semaforu (poseban red koji O.S. formira) ako je vrijednost semafora nula. Ako vrijednost semafora nije jednaka nuli, oduzet će se od semafora 1, a proces će dalje nastaviti sa svojim izvodjenjem. SIGNAL operacija naprotiv oslobadja proces iz reda čekanja na semaforu ako neki proces čeka na tom semaforu. Naravno, jedno izvodjenje SIGNAL operacije oslobadja samo jedan proces iz reda čekanja. Ako "na tom semaforu" ne čeka ni jedan proces, SIGNAL operacija će jednostavno povećati vrijednost semafora za 1.

Semafori mogu poslužiti u razne svrhe kao što su medjusobno isključenje procesa, sinhronizacija procesa, zaštita nedjeljivih resursa od simultanog korištenja više procesa itd.

Svakom se semaforu dodjeljuje početna vrijednost c(s). Ako je C(s)=1, onda takav semafor nazivamo binarni semafor. On će propustiti samo jedan proces. Ako označimo ukupni broj SIGNAL operacija na semaforu sa ns(s), a ukupni broj WAIT operacija na semaforu s sa ws(s), onda se vrijednost semafora s u nekom proizvoljnom trenutku promatranja može izraziti relacijom:

$$v(s) = c(s) + n(s) - nw(s) \qquad (2)$$

gdje je sa v(s) označena vrijednost semafora s u proizvoljnom trenutku promatranja. Budući da je per definitionem $v(s) \geq 0$, vrijedi:

$$c(s) + ns(s) - nw(s) \geq 0 \qquad (3)$$

odnosno

$$nw(s) \leq ns(s) + c(s) \qquad (4)$$

Znak jednakosti vrijedi kada je $v(s) = 0$. Ove jednostavne relacije su osnovne relacije koje vrijede za svaki semafor, a poslužit će nam kasnije u dokazivanju ispravnosti rješenja sinhronizacije pomoću semafora i operacija na semaforima.

4. RJEŠENJE PROBLEMA SINHRONIZACIJE TIPA PROIZVODJAČ-POTROŠAČ POMOĆU SEMAFORA I OPERACIJA NA SEMAFORIMA

Svako rješenje problema mora, osim gore navedenih relacija, zadovoljavati i relaciju (1) te mora uz to postojati medjusobno isključenje procesa prilikom pristupa u buffer.Za rješenje problema upotrijebit ćemo tri semafora, i to semafor s imenom PRAZAN, semafor koji neka se zove PUN i semafor kojemu ćemo dati ime SLOBODAN. Ova imena nas ujedno podsjećaju na to za što nam semafor služi. Tako na primjer semafor PRAZAN služi za kontrolu punjenja buffera pa se postavlja na vrijednost N. Dakle,vrijedi relacija

$$c(\text{PRAZAN}) = N \qquad (5)$$

Ovaj se semafor postavlja na početnu vrijednost N zato jer je moguće N puta unositi veličinu u buffer jer je N kapacitet buffera. Semafor PUN postavlja se na početnu vrijednost nula jer u prvom početku nije napunjen, dakle vrijedi relacija

$$c(\text{PUN}) = 0 \qquad (6)$$

Semafor SLOBODAN služi za kontrolu pristupa u buffer, što znači za medjusobno isključenje procesa "proizvodjača" i procesa "potrošača". Zbog toga semafor se postavlja na vrijednost 1,dakle vrijedi:

$$c(\text{SLOBODAN}) = 1 \qquad (7)$$

U početku može u buffer pristupiti samo jedan proces, i to proizvodjač, što će biti osigurano. Rješenje za odvijanje oba procesa bi upotrebom navedena tri semafora glasilo:

```
PROIZVODJAČ
P: BEGIN
        - Proizvedi veličinu
        WAIT(PRAZAN)
        WAIT(SLOBODAN)
        - Odloži veličinu u buffer
        SIGNAL (SLOBODAN)
        SIGNAL(PUN)
        GOTO P
   END

POTROŠAČ
C: BEGIN
        WAIT(PUN)
        WAIT(SLOBODAN)
        - Uzmi veličinu iz buffera
        SIGNAL(SLOBODAN)
        SIGNAL(PRAZAN)
        - Potroši veličinu
        GOTO C
  END
```

Analizirajući bilo kakvu medjusobnu situaciju kod izvodjenja oba procesa može se vidjeti da će uvijek biti ispunjeni uvjeti koje smo prije postavili. Počne li na primjer prvi s izvodjenjem, proces POTROŠAČ bit će zaustavljen na semaforu PUN jer je njegova vrijednost jednaka nuli sve dok se ne izvede proces PROIZVODJAČ. Čim PROIZVODJAČ stavi jednu veličinu u buffer, operacija SIGNAL (PUN) koju se izvodi u sklopu tog procesa oslobodit će proces POTROŠAČ koji čeka na tom semaforu i on će moći uzeti veličinu iz buffera. Ukoliko na primjer u vrijeme uzimanja (nismo odredili koliko ono traje) proces PROIZVODJAČ proizvede veličinu i hoće je staviti u buffer, bit će zaustavljen na semaforu SLOBODAN (medjusobno isključenje) jer resurs (buffer u ovom slučaju) drži drugi proces. Naravno, kad proces POTROŠAČ završi s uzimanjem veličine izvodjenjem operacije SIGNAL(SLOBODAN), oslobodit će proces PROIZVODJAČ iz stanja čekanja i dozvoliti mu pristup u buffer.

Može se prilično jednostavno dokazati da u bilo kojem momentu odnosno medjusobnom položaju kod izvodjenja oba procesa ispunjeni su traženi uvjeti za sinhronizaciju. Primijenimo li relaciju(4), imamo

$$nw(PRAZAN) \leq ns(PRAZAN) + N \qquad (8)$$

Relaciju možemo proširiti i pisati na slijedeći način:

$$0 \leq nw(PRAZAN) \leq ns(PRAZAN) + N \qquad (9)$$

jer je nw broj izvedenih WAIT operacija, pa može biti očigledno samo jednak nuli ili veći od nule. Takodjer vrijedi relacija

$$0 \leq nw(PUN) \leq ns(PUN) \qquad (10)$$

jer možemo primijeniti isti način zaključivanja. Promatrajući proces PROIZVODJAČ možemo postaviti relaciju

$$ns(PUN) \leq u \leq nw(PRAZAN) \qquad (11)$$

Naime, iako se čini da je broj SIGNAL operacija na semaforu PUN i WAIT operacija na semaforu PRAZAN isti i jednak broju upisanih (napunjenih) veličina, to nije u općem slučaju tako jer se može dogoditi da dodje do prekida u vrijeme stavljanja (odlaganja) veličine u buffer pa će u tom momentu vrijediti navedena relacija (11) i to može potrajati odredjeno vrijeme.

Iz procesa POTROŠAČ može se analogno zaključiti da vrijedi

$$ns(PRAZAN) \leq p \leq nw(PUN) \qquad (12)$$

jer proces POTROŠAČ može prilikom uzimanja veličine iz buffera biti takodjer prekinut u izvodjenju.

Iz relacije (11) slijedi

$$u \leq nw(PRAZAN) \qquad (13)$$

Dok iz relacije (13) i relacije (8) jednostavnom supstitucijom lijeve strane slijedi

$$u \leq ns(PRAZAN) + N \qquad (14)$$

Kombiniramo li sada relacije (12) i (14), dobivamo relaciju

$$u \leq p + N \qquad (15)$$

odnosno

$$u-p \leq N \qquad (16)$$

što je i trebalo dokazati.

Budući da su ova zaključivanja važeća za bilo koji vremenski trenutak promatranja, dokaz vrijedi općenito, tj. ako su procesi i njihovi odnosi regulirani na način kao u našoj defini-

ciji odnosno rješenju, uvijek će relacija (16) biti ispunjena. Ta relacija čini, medjutim, samo jedan dio uvjeta koji moraju biti ispunjeni pa treba i za ostale uvjete dokazati da postoje i ispunjeni su u bilo kojem trenutku promatranja.

Na temelju relacije (12) možemo dalje izvesti

$$p \leqslant nw(PUN) \tag{17}$$

Uvrstimo li umjesto nw(PUN) veličinu ns(PUN), relacija se pretvara u

$$p \leqslant ns(PUN) \tag{18}$$

a to možemo jer na temelju relacije (10) slijedi da je veličina ns(PUN) uvijek veća ili jednaka veličini nw(PUN).

Primjenom relacije (11) dobivamo

$$p \leqslant u \tag{19}$$

odnosno

$$0 \leqslant u - p \tag{20}$$

Povežemo li relaciju (20) s relacijom (16), dobivamo

$$0 \leqslant u - p \leqslant N \qquad \text{q.e.d}$$

što je upravo i trebalo dokazati.

Semafor SLOBODAN služi za medjusobno isključenje procesa PROIZVODJAČ i procesa POTROŠAČ prilikom zahvatanja u buffer. Čim jedan od njih izvede WAIT operaciju na semaforu SLOBODAN, drugi je ne može više izvesti, odnosno biva stavljen u red čekanja na tom semaforu. U tom će redu proces čekati tako dugo dok proces koji koristi buffer ne izvede SIGNAL operaciju na semaforu SLOBODAN. Ta operacija oslobodit će proces koji čeka i on će moći zahvatiti u buffer i izvesti svoje operacije s bufferom. Dakle, dok jedan proces ne završi kritičnu sekciju, drugi proces ne može početi s izvodjenjem svoje kritične sekcije, a to je upravo medjusobno isključenje koje se i trebalo u ovom slučaju postići.

5. ZAKLJUČAK

Pretpostavimo li da je moguća realizacija WAIT i SIGNAL operacija na način kao što je ovdje definirano, može se kao što je dokazano riješiti problem sinhronizacije procesa tipa proizvodjač-potrošač. Takodjer je u ovom prikazu bio dan primjer medjusobnog isključenja procesa pomoću semafora. Takav odnos mo-

guće je očigledno riješiti pomoću semafora na vrlo jednostavan način. Mogu se upotrebom semafora riješiti i drugi odnosi koji medju procesima postoje i treba ih riješiti na nivou operativnog sistema. Semafori i operacije na semaforima predstavljaju, dakle, vrlo dobru i uspješnu tehniku za rješavanje odnosa medju procesima.

Stvarna realizacija semafora i operacija na semaforima, kao operacija koje može izvesti samo operativni sistem, je moguća, i to kao kombinirano hardwarsko-softwarsko rješenje. Pritom treba napomenuti da u slučaju kada postoji stvarno istovremeni zahtjev za izvodjenje bilo WAIT bilo SIGNAL operacije od strane dva ili više procesa, mora postojati hardwarski mehanizam koji jedan od procesa propušta, a druge stavlja u red čekanja. Naravno, dalje se nameće pitanje organizacije reda čekanja na semaforu, odnosno oslobadjanja procesa iz reda. Moguće je uzeti jednostavni FIFO, organizaciju na bazi prioriteta ili neku drugu, što je pitanje dizajna samog operativnog sistema, odnosno hardwarea.

Najvažniji rezultat primjene semaforske tehnike je ustvari rješenje problema koji je poznat pod nazivom "busy wait" (zaposleno čekanje). U biti taj problem se sastoji u tome što proces koji čeka i u stanju "čekanja" koristi procesor i izvodi neprestano ispitivanje stanja neke varijable. Proces zbog toga što ne napreduje u svojem izvodjenju praktički "čeka". On čeka dogadjaj koji će promijeniti stanje varijable koju testira i tako mu omogućiti daljnje izvodjenje. Dakle, takvo čekanje predstavlja gubitak procesorskog rada. Neke druge tehnike upravo takvo čekanje koriste za rješenje sinhronizacije i medjusobnog isključenja procesa. Kod semaforske tehnike, medjutim, proces stvarno miruje i ne izvodi se kad je u stanju čekanja jer se stavlja u red čekanja i ne dobiva procesorsko vrijeme za izvodjenje, nego stvarno miruje tako dugo dok ga SIGNAL operacija na semaforu na kojem čeka ne oslobodi iz reda čekanja i ponovno aktivira.

WAIT i SIGNAL operacije ugradjuju se u sam nukleus operativnog sistema. Danas je ova tehnika prihvaćena i priznata u dizajnu i korištenju operativnih sistema. WAIT i SIGNAL operacije ustvari su hardwarsko-softwarsko rješenje jer se ne mogu riješiti kompletno na nivou operativnog sistema. Pozivanje ovih operacija moguće je samo putem operativnog sistema (SVC instrukcija, extracode) tako da ih kompjuterski sistem izvodi samo u kontrolnom stanju (control mode, supervisor mode).

L I T E R A T U R A :

1. *Coffman E.G., Denning P.J.: Operating Systems Theory,Prentice Hall, 1973.*

2. *Shaw A.C.: The Logical Design of Operating Systems, Prentice Hall, 1974.*

3. *Donovan J.J., Madnick S.E.: Operating Systems, McGraw Hill Book Company, 1974.*

4. *Lister A.M.: Fundamentals of Operating Systems, The Mcmillan Press Ltd., 1975.*

5. *Dijkstra E.W.: The Structure of T.H.E. Multiprogramming System, CACM 11, 1968.*

Primljeno: 1980-10-05

Kvaternik R. An Example of Parallel Processes Synchronization Using Semaphores and Operations on Semaphores

S U M M A R Y

For efficient and easy treatment of operating systems problems it is necessary to introduce the concept of a process. The process is an activity within the computer system as opposed to the program, which is only a static set of instructions. Between processes there are different kinds of communications. One of them is a synchronization of processes. Using semaphores and operations on semaphores it is possible to solve the "producer-consumer" problem in an efficient manner. Such a solution, together with the formal proof of its correctness, has been demonstrated in this article.

The final conclusion is that semaphores and operations on semaphores are really an efficient tool for treating communications problems between processes.The semaphoring techniques eliminate "busy wait" completely.Implementation of WAIT and SIGNAL operations has to be solved, however, during operating system design.