Mr. sc. Mirko Varga
Fakultet organizacije i informatike
Varaždin

UDK: 655:681.3
Pregledni rad

PROBLEMATIKA MEĐUSOBNOG ISKLJUČENJA PROCESA IMPLEMENTIRANA RAZLIČITIM TEHNIKAMA

U radu se obrađuje problematika međusobnog isključenja procesa, te se navode najznačajnije tehnike implementacije. Posebno se razmatraju slijedeće tehnike: korištenje zajedničkih varijabli, Dekkerov algoritam, Test & Set instrukcija te najuspješnija tzv. semaforska tehnika. Semaforska tehnika najčešće se koristi zbog niza prednosti na koje se ukazuje u radu. Na kraju se naznačena problematika ilustrira na dva problema posebno konstruirana za ovakvu primjenu i to na problemu "Čitatelja i Pisaca" te čuvenom problemu pod nazivom "Ručak filozofa" kojeg je postavio Dijkstra.

Ključne riječi: Međusobno isključenje procesa; Dekkerov algoritam, Semaforska tehnika.

1. UVOD

Problematika međusobnog isključenja koja se odnosi na određen konačan procesa, može se općenito prikazati slijedećim pseudo kodom (vidi sl. 1).

Kada dva ili više korisnika ili procesa zahtijevaju pristup do zajedničkih resursa, javlja se problem definiranja odnosa među procesima. Odnose među procesima i načine njihovog komuniciranja možemo promatrati kao: (Kvaternik, 1983.).

- međusobno isključenje procesa (engl. "mutual exclusion")
- sinhronizacija procesa (engl. "synchronization")
- blokiranje procesa tj. zastoj (engl. "deadlock")

```
loop mutex
        entry section
          critical section
        exit section
            non-critical section
end loop
```

Sl. 1 Prikaz pomoću pseudo koda

Suvremen računalni sustav uključuje niz različitih resursa koje možemo svrstati u dvije osnovne kategorije:

- djeljivi (engl. "sharable")
- nedjeljivi (engl. "nonsharable")

Suvremeni načini procesiranja podrazumijevaju paralelne obrade, tj. procesi nastoje koristiti što više resursa u paralelnom odnosno kvaziparalelnom izvođenju. Paralelno procesiranje znači da su dva ili više procesa aktivna u određeno vrijeme i da dijele resurse zajedničke računalne platforme.

Kada proces pristupa djeljivom resursu, on ulazi u tzv. kritičnu sekciju procesa. Prije samog ulaska u kritičnu sekciju proces mora ispitati (provjeriti) da li se možda već neki drugi proces nalazi u njoj. Ukoliko to nije slučaj, on može nesmetano ući u kritičnu sekciju te mora registrirati (signalizirati) svoj ulazak, tj. korištenje resursa naznačiti drugim procesima. Kada je korištenje završeno, proces signalizira da je ranije korišteni resurs slobodan, tj. da izlazi iz svoje kritične sekcije.

Iz ovog proizlazi da je kritična sekcija procesa međusobno isključiva sekcija jednog ili više procesa, ali ne neophodno i svih procesa (K. Yue, 1988.).

Indikacija o tome da li je resurs u korištenju ili je slobodan može se provjeriti te implementirati na različite načine. U nastavku ukazat ćemo na evoluciju razvoja od najjednostavnijih tehnika koje se temelje na korištenju zajedničkih varijabli do najdjelotvornije semaforske tehnike.

2. TEHNIKE IMPLEMENTACIJE

Najpoznatije tehnike pomoću kojih se može riješiti međusobno isključenje procesa jesu:

- korištenje zajedničke varijable/varijabli u memoriji (engl. "memory interlock", "flags")
- Dekkerov algoritam
- Test & Set instrukcija
- Semaforska tehnika

2.1 Korištenje zajedničke varijable

Najjednostavniji način međusobnog isključenja procesa može se izvesti upotrebom zajedničke cjelobrojne ili logičke varijable. Pokazat ćemo to na primjeru dvaju procesa (P_i, P_j), a rješenje je dato u pseudoAlgolu (vidi sl. 2).

U našem primjeru prikazana su dva procesa (P_i, P_j) koji mogu doći u različite vremenske ovisnosti prilikom procesiranja. Pri tom je važno osigurati da se izbjegne preklapanje procesa u njihovim kritičnim sekcijama što se izvodi pomoću logičke varijable L. Npr: ukoliko je proces P_i prvi ušao u svoju kritičnu sekciju, to će rezultirati postavljanjem zajedničke logičke varijable L na .FALSE. što će prouzročiti da proces P_j neprestano izvodi instrukciju označenom labelom L2, tj. neproduktivno troši vrijeme na testiranje varijable L što ne doprinosi progresu (napredovanju) procesa, tako dugo dok proces P_i ne izađe iz svoje kritične sekcije postavljajući logičku varijablu L na vrijednost .TRUE. čime se omogućava procesu P_j da uđe u svoju kritičnu sekciju.

```
BEGIN
    LOGICAL L; L:=.TRUE.
        PARBEGIN
            Pi: BEGIN
                    L1: IF (L.EQ.0) THEN GOTO L1;
                        L:=.FALSE.;
                            kritična sekcija Pi
                        L:=.TRUE.;
                            preostali dio Pi;
                        GOTO L1;
                    END;

            Pj: BEGIN
                    L2: IF (L.EQ.0) THEN GOTO L2;
                        L:=.FALSE.;
                            kritična sekcija Pj
                    L:=.TRUE.;
                            preostali dio Pj;
                    GOTO L2;
                    END;

        PAREND;
END
```

Sl. 2 Međusobno isključenje dvaju procesa (P_i, P_j) korištenjem logičke varijable

Time je postignuto međusobno isključenje procesa osim u slučaju da se oba procesa izvode potpuno paralelno, tj. da u isto vrijeme izvedu instrukcije na labelama L1 i L2 te istovremeno uđu u kritične sekcije.

Zbog toga je potrebno modificirati prikazano rješenje na način da se međusobno isključenje osigura u svakom slučaju. Upotrebom više varijabli može se konstruirati rješenje koje će ispuniti ovaj zahtjev. To rješenje prvi je postavio matematičar T. Dekker, a ono je poznato pod nazivom Dekkerov algoritam.

2.2 Dekkerov algoritam

Dekkerov algoritam je rješenje temeljeno na upotrebi više varijabli. Svaki proces koji pristupa djeljivom resursu ima vlastitu varijablu ("flag") koja pokazuje da li se resurs koristi li ne. Ove varijable su zajedničke svim procesima, ali samo vlasnički proces može setirati, odnosno brisati naznačenu varijablu.

Programsko rješenje Dekkerovog algoritma prikazuje slika 3.

Prednost ovog algoritma u odnosu na prethodno rješenje očituje se u tome što će u kritičnu sekciju biti propušten samo jedan proces bez obzira kakva je međusobna vremenska usklađenost procesiranja svih prisutnih procesa. Bolje rezultate Dekkerov algoritam daje kod međusobnog isključenja manjeg broja procesa.

Iako je ovaj algoritam otklonio spomenuti nedostatak i pružio općenito rješenje za međusobno isključenje n procesa, prisutni su i neki ozbiljni nedostaci. Osnovni nedostatak svodi se na značajno veliki interno generirani rad računala (engl. "overhead") što ga uzrokuje neprestano testiranje i "vrtnja u petlji" privremeno zaustavljenih procesa. Takvi procesi troše procesorsko vrijeme iako ono nije u funkciji osiguravanja njihovog progresa (napredovanja) što rezultira zaposlenim čekanjem (engl. "busy wait"). Zbog toga nastojala su se pronaći rješenja koja neće iskazivati taj nedostatak. Takva rješenja temelje se na primjeni "Test & Set" instrukcije iz koje je izvedena i najdjelotvornija tzv "Semaforska tehnika".

```
PROGRAM DekkersAlg.
  CONST: numOfProcess = 2;
  VAR
  flag: ARRAY [1..numOfProcess] OF BOOLEAN;
  turn: BOOLEAN;
  PROCEDURE Process (ProcessNum: 1..numOfProcess;
                     exitCondition: BOOLEAN);
      VAR secondProcessNum: 1..numOfProcess
      BEGIN
      secondProcessNum:= (ProcessNum MOD 2) + 1;
      REPEAT
      flag[ProcessNum]:=.TRUE.;
          WHILE flag[secondProcessNum] DO
              IF(turn.eq.secondProcessNum) THEN
                  flag[ProcessNum]:=.FALSE.;
                      WHILE turn=secondProcessNum DO
          END
              flag[ProcessNum]:=.TRUE.;
              END IF;
          END;
      UseResource;
          turn=secondProcessNum;
          flag[ProcessNum]:=.FALSE.;
              DoRest;
                  UNTIL exitCondition;
      END Process;
      BEGIN
          FOR i:=1 TO numOfProcess DO
              flag[i]:=.FALSE.
              END
          turn:=1;
      ExecuteConcurrentlyAllProcesses;
      END DekkersAlg:
```

Sl. 3 Dekkerov algoritam

2.3 Test & Set instrukcija

Međusobno isključenje procesa možemo realizirati i primjenom Test & Set instrukcije. Definicija te instrukcije može se prikazati slijedećim kodom u Pseudo-Algolu: (vidi sl. 4)

```
Test & Set
      "L1:IF L=0 THEN L:=1" ELSE GOTO L1;
```

Sl. 4 Pseudo kod Test & Set instrukcije

Dio instrukcije koji se nalazi pod znakovima navodnika označava da se taj dio mora izvesti kao nedjeljiva operacija tj. da prilikom njezinog izvođenja ne smije doći do prekida odnosno prekidi su maskirani (zabranjeni) tj. mora se izvesti testiranje varijable L te eventualno postavljanje na novu vrijednost. Ova instrukcija dolazi u paru s instrukcijom "Reset" koja ima funkciju postavljanja varijable L na inicijalnu vrijednost (pretpostavka jest da L = 0.)

Međusobno isključenje n procesa pomoću "Test & Set" instrukcije relativno je jednostavno a shematski ga možemo prikazati na slijedeći način: (vidi sl. 5)

```
Process_1:                    Process_n:
 START                         START
  Test & Set(L)                 .....
   critical section              .....
    Reset(L)                     .....
    .....                        Test & Set(L)
    .....                         critical section
    .....                        Reset(L)
END                               .....
                                  .....
                                END
```

Sl. 5 Međusobno isključenje n-procesa korištenjem Test & Set instrukcije

Onaj proces, koji prvi izvede "Test & Set" instrukciju dobiva pravo izvođenja svoje kritične sekcije. U slučaju posve paralelnog izvođenja na hardverskom nivou je riješeno

da se propusti samo jedan proces u svoju kritičnu sekciju, dok preostali procesi za svoj napredak moraju čekati izlazak dotičnog procesa iz kritične sekcije.

Ovaj postupak omogućava međusobno isključenje procesa na relativno jednostavan način, iako se i ovdje javlja problem zaposlenog čekanja. Tehnika koja se izvodi iz ove a koja ujedno ovaj nedostatak eliminira poznata je pod nazivom "Semaforska tehnika".

2.4 Semaforska tehnika

Jedna je od općenito prihvaćenih tehnika, koju je uveo E. W. Dijkstra 1965. godine a poznata je pod nazivom Semaforska tehnika. Semaforska tehnika koristi se varijablama pod nazivom semafori te posebno definiranim i implementiranim operacijama na semaforima.

Dijkstra je uveo dvije elementarne operacije na semaforima koje je prema originalnom nazivu u holandskom jeziku označio s P (engl. "WAIT") i V (engl."SIGNAL") (vidi sl. 6 i sl. 7).

Vrijednost općenitog semafora (kvantitativnog) definira se kao nenegativna cjelobrojna vrijednost koja nam ukazuje na trenutno stanje u sustavu, odnosno o stanju resursa koji je dotičnim semaforom protekcioniran. Inicijalna vrijednost tako definiranog semafora obično označava instancu resursa odnosno broj raspoloživih jedinica resursa za korištenje.

```
PROCEDURE P(s)
      * P originalno od Dijkstre
            "passeren", means "to pass" *
      BEGIN
            IF s > 0 THEN
                  s := s - 1
            ELSE
                  BEGIN
                  - zaustavi proces i stavi ga u stanje čekanja
                  - stavi proces u red čekanja na semaforu s
                  - oslobodi procesor
      END IF
END P(s)
```

Sl. 6 Definicija operacije P(s) (engl."WAIT")

```
PROCEDURE V(s)
        * V originalno od Dijkstre
                "vrygeven", means "to release" *
VAR process: tProcess
        BEGIN
        * Ispitaj da li je proces suspendiran *
        IF process = SuspendedProcess THEN
                BEGIN
                - uzmi proces iz reda čekanja
                - aktiviraj suspendirani proces
                ELSE
                        s := s + 1
        END IF
END V(s)
```

Sl. 7 Definicija operacije V(s) (engl. "SIGNAL")

Specijalan slučaj semafora predstavlja tzv: binarni semafor koji može poprimiti samo vrijednosti 0 odnosno 1. Slijedeće tri bazične operacije mogu promijeniti vrijednost semafora:

1. inicijalizacija semafora s

2. operacija čekanja (hol. "Passeren", engl. "WAIT") procesa na semaforu s

3. operacija oslobađanja (hol. "Vrygeven", engl. "SIGNAL") procesa na semaforu s

```
PROGRAM MutualExclusion
CONST numOfProcesses = n;
 VAR s: 1..numOfProcesses;
       * Svaki proces ima vlastiti semafor *

 PROCEDURE Process(processNum: 1..numOfProcesses;
                   exitCondition: BOOLEAN) ;
  BEGIN
    REPEAT
     WAIT(s);
       critical section;
         SIGNAL(s);
        non-critical section;
          UNTIL exitCondition;
  END Process;
    BEGIN
     s := 1;
     ExecuteConcurrentlyAllProcesses;
END MutualExclusion
```

Sl. 8 Međusobno isključenje pomoću semafora

Međusobno isključenje n procesa pomoću semaforske tehnike prikazuje slika 8.

Ova tehnika je najdjelotvornija u rješavanju odnosa među procesima, a glavne prednosti u odnosu prema ostalim spomenutim rjesenjima mogu se svesti na slijedeće:

- jednostavno korištenje
- procesi se stavljaju u stvarno čekanje čime se eliminira nezaposleno čekanje (engl. "busy wait")
- mogućnost implementacije u jezgru operacijskog sustava (tzv. nukleus)

Semafori omogućavaju sinhronizaciju i komunikaciju paralelnih procesa, iako su moguće pogreške pri programiranju u različitim programskim jezicima koji podržavaju instrukcije na semaforima. Jedan od glavnih razloga za to jest što prevoditelji ne mogu provjeriti pravilnost upotrebe semafora.

Tipične pogreške svode se na slijedeće:

- izostavljanje operacije "Wait" na semaforu, što uzrokuje direktan skok u kritičnu sekciju, odnosno operacije "Signal" što će uzrokovati čekanje nekog od zaustavljenih procesa na instrukciji "Wait" dotičnog semafora.

- mogućnost neprovođenja zaštite svih kritičnih sekcija u kompleksnim programima.

- nepravilna upotreba operacija na semaforima (zamjena redoslijeda operacija) može dovesti do blokiranja procesa odnosno zastoja (engl. "deadlock").

Bez obzira na spomenute poteškoće prilikom implementacije u odgovarajućem programskom jeziku ova tehnika općenito je prihvaćena u dizajniranju operacijskih sustava, a osim toga često se koristi i u teorijskim razmatranjima.

3. Ilustrativni primjeri

Problematiku međusobnog isključenja ilustrirat ćemo na dva problema posebno konstruirana za ovakvu primjenu i to na problemu "Čitatelja i Pisaca" te čuvenom problemu koji potječe od Dijkstre pod nazivom "Ručak filozofa".

3.1 Problem "Čitatelja i Pisaca"

Dva paralelna procesa, nazvana "Čitatelji i Pisci" dijele jedan resurs. Čitatelji mogu istovremeno koristiti resurs, dok svaki pisac ima ekskluzivno pravo pristupa resursu. Kada je pisac spreman koristiti resurs, to mu treba i omogućiti u što kraćem vremenu.

Sinhronizaciju dva navedena procesa prikazuje slika 9.

```
PROGRAM Readers & Writers
    Writers
        entry section: WAIT(mutex);
        exit section:  SIGNAL(mutex);

    Readers
        entry section: WAIT(gate);
                N_Readers ← N_Readers + 1;
                IF N_Readers = 1 THEN WAIT(mutex);
                SIGNAL(gate);
        exit section: WAIT(gate);
                N_Readers ← N_Readers − 1;
                IF N_Readers = 0 THEN SIGNAL(mutex);
                SIGNAL(gate);
```

Sl. 9 Sinhronizacija problema "Čitatelji i Pisci"

3.2 Problem "Ručak filozofa"

Dijkstra je 1965. godine konstruirao i riješio problem sinhronizacije pod nazivom "Ručak filozofa". Problem se sastoji u slijedećem: 5 filozofa sjede oko stola. Ispred svakog filozofa nalazi se tanjur špageta, a s njegove lijeve i desne strane po jedna viljuška. Špageti su tako skliski da filozof treba uzeti dvije viljuške (sa svoje lijeve i desne strane) kada poželi jesti. Život filozofa sastoji se od alternativnih perioda jedenja i razmišljanja.

Rješenje ovog poznatog problema može dovesti do zastoja ukoliko se ne vodi dovoljno računa o svim elementima. Prikaz jednog takvog rješenja dat je na slici 10.

Iz prikazanog rješenja možemo izvući zaključak da je potrebno modificirati ovaj pseudo kod u smislu da program provjerava nakon što filozof uzme lijevu viljušku da li je desna slobodna. Ukoliko to nije slučaj, otpusta viljušku, čeka neko vrijeme te opet ponavlja cijeli postupak. Sama implementacija moguća je uvođenjem jednog binarnog semafora.

```
PROGRAM "The Dinning Philosophers Problem"
# define N 5

filozof(int i ) % 0 <= i <= N
{
while (TRUE) {
        razmišljaj(); % filozof razmišlja
          uzmi_viljušku(i);    % filozof uzima lijevu viljušku sa stola
          uzmi_viljušku((i+1) mod N); % filozof uzima desnu viljušku sa stola
        jedi(); % filozof jede spagete
          otpusti_viljušku(i); % filozof otpusta lijevu viljušku na stol
          otpusti_viljušku((i+1) mod N); % filozof otpušta desnu viljušku
    }
  }
```

Sl. 10 Rješenje problema "Ručak filozofa" koje dovodi do zastoja

4. Umjesto zaključka

U ovom radu predstavljene su najpoznatije tehnike za međusobno isključenje procesa. Procesi mogu međusobno komunicirati uz pomoć opisanih tehnika interprocesne komunikacije pri čemu je potrebno osigurati međusobno isključenje procesa u njihovim kritičnim sekcijama. Teoretski, zato se mogu koristiti različite implementacije iako se u praksi zbog navedenih prednosti najčešće koristi semaforska tehnika. Literatura koja obrađuje područje operacijskih sustava diskutira niz klasičnih problema iz ove oblasti, od kojih ovdje ističemo dva problema: problem "Čitatelja i Pisaca" te "Ručak filozofa". Navedene tehnike implementacije najprije se testiraju upravo na takvim tipovima problema te se na temelju toga ocjenjuje njihova uspješnost i mogućnost daljnje primjene.

LITERATURA

[1] E. Dijkstra, Co-operating sequential processes, Programming Languages, Academic Press, New York, 1968.

[2] R. Kvaternik, Uvod u operativne sisteme, Informator, Zagreb, 1983.

[3] A. W. Leigh, Real Time Software for Small System Design, McGraw-Hill, Inc., first edition, 1990.

[4] I. P. Page, R. T. Jacob, The Solution of mutual exclusion problems which can be described graphically, The Computer Journal 32 (1), 45-54 (1989).

[5] S. Silberschatz, Operating Systems Concepts, Addison-Wesley, 1992.

[6] R. Sethi, Programming Languages (Concepts & Constructs), Addison-Wesley, 1989.

[7] K. Yue, R. T. Jacob, An Efficient Starvation-Free Semaphore Solution for the Graphical Mutual Exclusion Problem, The Computer Journal 34(4),345-349(1991).

Primljeno: 1994-05-15

Varga M. The Different Solutions for the Mutual Exclusion Problem

S U M M A R Y

Different techniques for constructing efficient solutions for mutual exclusion problems based on common variables (flags), Dekker's algorithm, Test & Set instructions and semaphores are presented in the paper. The number of semaphores used is equal to the number of processes in the mutual exclusion problem. This paper discusses two of the better-know problems: The Readers and Writers Problem and The Dining Philosophers Problem.