Communication and Synchronization between Processes

Asist. Felician ALECU
Catedra de Informatică Economică, A.S.E. Bucureşti

The main requirement for modifying shared data is to make sure that the modification is completed before any other process can interfere. An atomic operation is defined as an operation that cannot be interrupted by other processes. To create an atomic operation, a mechanism to make processes wait is needed. The program execution time will be increased by forcing processes to wait for access to shared data.
Keywords: *synchronization, mutual exclusion, contention, serialization, critical section, semaphore, mutex, barrier.*

Excluderea mutuală

Calculatoarele cu memorie partajată dispun de un spațiu comun de memorie care poate fi accesat de către toate procesoarele din sistem. Procesoarele din sistem comunică prin date memorate în spațiul unic de memorie partajată. Utilizarea unui spațiu unic de memorie poate conduce la conflicte de acces la memorie atunci când mai multe procesoare încearcă să utilizeze aceeaşi zonă de memorie sau când doresc să utilizeze o variabilă partajată la care un alt procesor are acces exclusiv. În plus, utilizarea memoriile cache atrage după sine necesitatea implementării unor mecanisme software sau hardware care să asigure consistența acestora. La modificarea unor date aflate în memoria locală a unui procesor este nevoie să se actualizeze toate copiile acestor date aflate în memoriile cache. Aceste mecanisme care asigură coerența memoriilor cache contribuie la creşterea traficului efectuat în cadrul rețelei de interconectare.

Una dintre principalele cauze generatoare de erori în execuția unui program paralel o reprezintă inconsistența temporară dintre o variabilă locală şi copia ei din memoria globală. Dacă variabila este accesată concurent de către mai multe procese, atunci rezultatele obținute vor fi eronate datorită faptului că valoarea variabilei globale nu a fost actualizată în concordanță cu modificările operate asupra copiei locale. Această situație se manifestă atât în cadrul calculatoarelor cu procesoare multiple cât şi în cazul sistemelor secvențiale multitasking în care se execută concurent mai multe procese.

Pentru a preveni astfel de situații este necesar ca actualizarea valorii unei variabile partajate să se realizeze ca o operație atomică care nu poate fi întreruptă de către un alt proces care doreşte să acceseze aceeaşi zonă de memorie. Cel de-al doilea proces va putea actualiza variabila globală doar după ce primul proces a eliberat zona respectivă de memorie. Serializarea accesului la variabilele partajate se obține prin implementarea mecanismelor de excludere mutuală. Astfel, pentru actualizarea în siguranță a unei variabile globale este necesar ca operația respectivă să se execute într-o secțiune critică. Procesul care intră în secțiunea critică are dreptul exclusiv de a accesa variabila partajată asociată secțiunii respective. Atât timp cât procesul execută secțiunea critică, toate celelalte procese concurente nu vor avea dreptul să acceseze variabila partajată respectivă. Acest mecanism poartă numele de excludere mutuală deoarece un proces exclude temporar accesul celorlalte procese la variabila partajată.

Protocoale de aşteptare utilizate în excluderea mutuală

Atunci când un proces doreşte să acceseze o resursă care nu este disponibilă va fi necesar ca acesta să aştepte eliberarea resursei respective. Protocoalele de aşteptare care pot fi utilizate în astfel de cazuri sunt următoarele:

- *aşteptarea activă (busy wait)* – procesorul va cicla într-o buclă aşteptând eliberarea resursei pentru a intra în secțiunea critică. Un

astfel de protocol este destul de simplu de implementat dar conduce la o eficienţă scăzută datorită faptului că se consumă timp de execuţie cu un proces care nu avansează.
- *aşteptarea pasivă (sleep wait)* – permite suspendarea unui proces care doreşte să acceseze o resursă ce este ocupată de un alt proces. Sistemul de operare suspendă procesul în cauză care va fi reactivat atunci când resursa devine disponibilă.
- *aşteptarea combinată* – în primă fază procesul întră în aşteptare activă pentru un interval prestabilit de timp. Dacă resursa dorită nu se eliberează atunci procesul este suspendat şi trecut în aşteptare pasivă.

În sistemele uniprocesor multitasking este implementat protocolul de aşteptare pasivă prin care procesul este suspendat până când resursa solicitată devine disponibilă. În sistemele multiprocesor pot fi implementate toate cele trei tipuri de protocoale. Aşteptarea activă se poate dovedi mai eficientă decât cea pasivă deoarece redă mai rapid controlul procesului. Din acest motiv există tendinţa ca acest protocol de aşteptare să fie mai frecvent folosit.

Mecanisme hardware de implementare a excluderii mutuale

Excluderea mutuală în sistemele uniprocesor este asigurată cu ajutorul instrucţiunilor de validare/invalidare a întreruperilor care asigură accesul exclusiv al unui proces la o resursă.

Atunci când procesul intră în secţiunea critică, sistemul invalidează toate întreruperile care ar putea conduce la comutarea procesului curent şi la planificarea altuia. Procesul execută operaţiile dorite după care, la ieşirea din secţiunea critică, sistemul va valida întreruperile respective.

Este posibil ca prin invalidarea întreruperilor să fie afectate şi procese care nu au legătură cu secţiunea critică şi cu procesul care o execută. În plus, invalidarea întreruperilor este un mecanism periculos atunci când este utilizat în programele de aplicaţii deoarece poate conduce la blocarea sistemului. Este de preferat ca operaţiile de validare şi invalidare a întreruperilor să fie implementate sub forma unor funcţii de sistem care să poată fi utilizate doar de către rutine ale sistemului de operare.

Invalidarea întreruperilor este un mecanism ce nu poate fi aplicat în cadrul sistemelor paralele deoarece instrucţiunile respective afectează numai procesorul curent, fără a putea împiedica alte procese să se execute pe alte procesoare. Aceste sisteme folosesc instrucţiuni de tipul *TestAndSet* pentru a soluţiona conflictele de acces la resurse partajate. Instrucţiunile de tipul *TestAndSet* sunt incluse în setul de instrucţiuni ale procesoarelor din care este format sistemul de calcul.

Programele de aplicaţii pot folosi în mod direct aceste instrucţiuni însă de regulă ele sunt utilizate pentru implementarea unor mecanisme de sincronizare de nivel ridicat cum ar fi semafoarele, mutex-urile, barierele, etc.

Instrucţiunea *TestAndSet* acţionează în două etape ce sunt executate ca o operaţie atomică, indivizibilă:
- mai întâi testează valoarea variabile de control primită ca operand cu o valoare constantă care indică faptul că resursa este ocupată. În urma testului se setează în mod corespunzător flag-urile de condiţie ale procesorului.
- se setează operandul la valoarea constantă care a fost folosită în cadrul testului.

Aceste mecanisme sunt utilizate pentru obţinerea unor obiecte de sincronizare de nivel ridicat cum ar fi semafoarele, mutex-urile, barierele etc. Mecanismele de sincronizare între procese permit serializarea accesului concurent la resursele partajate din sistem prin folosirea excluderii mutuale şi a operaţiilor atomice.

Semafoare

Semafoarele reprezintă unul dintre cele mai generale mecanisme de sincronizare prezente atât în sistemele multitasking uniprocesor cât şi în cele multiprocesor. Ele au fost introduse de către Dijkstra în anul 1968.

Un semafor reprezintă o variabilă partajată care permite sau interzice accesul unui proces la o resursă comună. Semafoarele pot fi împărţite în două categorii în funcţie de valorile care le pot fi asociate:

- *semafoare binare* – pot lua una din valorile 0 (ocupat) şi 1 (liber);
- *semafoare generale* – pot lua orice valoare întreagă. Valoarea 0 indică faptul că resursa este ocupată în timp ce o valoare mai mare ca 0 semnifică faptul că resursa este liberă.

Orice semafor suportă două operaţii de bază definite de Dijkstra ca fiind primitivele P(s) şi V(s). P(s) încearcă să decrementeze valoarea semaforului cu o unitate iar dacă semaforul era deja zero atunci procesul rămâne în aşteptare până când variabila devine pozitivă. Primitiva V(s) are ca efect incrementarea cu o unitate a valorii variabilei semafor. Operaţiile asupra semafoarelor sunt atomice şi indivizibile pentru a preveni situaţia în care două procese ar accesa concurent resursa partajată pentru care a fost definit semaforul.

În continuare va fi descrisă implementarea în pseudocod a operaţiilor cu semafoare folosind aşteptarea activă:

```
P (semafor)
      -- câştigă semaforul
while (semafor = 0)
      end while
      semafor = semafor - 1
End

V (semafor)
      -- eliberează semaforul
      semafor = semafor + 1
End
```

Folosirea aşteptării active pentru implementarea operaţiilor cu semafoare are ca principal dezavantaj o eficienţă scăzută ce decurge din faptul că se consumă timpi de execuţie cu un procese care se află în aşteptare. În plus poate exista posibilitatea ca un proces să aştepte foarte mult timp pentru obţinerea semaforului atunci când mai multe procese concurează pentru obţinerea resursei deoarece nu există implementat nici un fel de sistem de priorităţi.

Pentru prevenirea unor astfel de situaţii se poate folosi aşteptarea pasivă prin definirea unui sistem de aşteptare pentru fiecare semafor asociat unei resurse partajate. Procesele care doresc să acceseze resursa vor fi introduse într-o coadă de aşteptare iar servirea se va face conform unei discipline prestabilite (FIFO - *First In, First Out*, LIFO – *Last In, First Out*, SIRO – *Service In Random Order*). Procesele din coada de aşteptare sunt suspendate până în momentul în care resursa devine disponibilă iar procesul respectiv poate fi planificat. Procesele suspendate nu mai consumă timp procesor deoarece sunt eliminate din lista proceselor gata de execuţie.

Implementarea primitivelor în acest caz poate fi descrisă în pseudocod în felul următor:

```
P (semafor)
      -- câştigă semaforul
      if semafor = 0
            suspenda_proces
      else
      semafor = semafor-1
end if
End

V (semafor)
      -- eliberează semaforul
      if lungime_coada > 0
            planifica_proces
      else
      semafor = semafor+1
end if
End
```

Eficienţa implementării semafoarelor prin aşteptarea pasivă creşte simţitor faţă de cazul anterior datorită faptului că procesele suspendate nu consumă timp de execuţie deoarece sunt eliminate din lista proceselor gata de execuţie. În plus, datorită faptului ca există o disciplină de servire asociată firului de execuţie, fiecare proces va primi accesul la resursa partajată într-un interval de timp finit. Semafoarele reprezintă cele mai utilizate mecanisme de sincronizare folosite în cadrul sistemelor uniprocesor multitasking şi al celor multiprocesor datorită uşurinţei cu care pot fi implementate şi utilizate pentru serializarea accesului concurent al mai multor procese la o resursă partajată.

Mutex-uri

Mutex-ul (*MUTual Exclusion* – excludere mutuală) este un mecanism de sincronizare care asigură în exclusivitate dreptul de folosire a unei resurse partajate. Resursei partajate i se asociază un *mutex* care va avea ca rol serializarea accesului concurent al mai mul-

tor procese la resursa partajată. Operațiile care se pot executa asupra obiectului de sincronizare sunt:

- *operația de obținere a mutex-ului* – este executată de către un proces care doreşte să obțină accesul al resursa partajată. Dacă obiectul este liber atunci acesta va fi trecut pe valoarea *ocupat* într-o operație atomică iar procesul va începe să execute secțiunea critică. În cazul în care *mutex-ul* are deja valoarea *ocupat*, procesul va trebui să aştepte până când obiectul de sincronizare va fi eliberat. Protocolul de aşteptare implementat poate fi prin aşteptare activă, aşteptare pasivă sau aşteptare combinată. Sistemele uniprocesor multitasking folosesc în exclusivitate protocolul de aşteptare pasivă;
- *operația de eliberare a mutex*-ului – este executată de către fiecare proces la ieşirea din secțiunea critică. Eliberarea obiectului de sincronizare nu poate fi efectuată decât de către procesul care l-a şi ocupat.

Bariere

Barierele reprezintă mecanisme de sincronizare de nivel înalt utilizate pentru sincronizarea unor procese concurente într-un anumit punct al execuției programului.

O barieră are asociat un identificator unic. În punctul de sincronizare, fiecare proces va apela o funcție de sincronizare care are ca parametri identificatorul barierei şi numărul de procese *N* pentru care se doreşte sincronizarea. Până la îndeplinirea condiției, toate procesele sosite vor fi trecute în stare de aşteptare (activă, pasivă sau combinată). Când numărul proceselor ajunse în punctul de sincronizare şi care au apelat bariera respectivă va fi egal cu *N*, atunci condiția de sincronizare se consideră îndeplinită şi toate procesele în aşteptare vor fi reactivate pentru a-şi continua execuția.

Implementarea barierelor se face folosind instrucțiuni atomice care garantează actualizarea în siguranță a variabilei în care se memorează numărul de procese care au ajuns în punctul se sincronizare asociat barierei respective.

Bibliografie

[Gro03] D. Gross, C. M. Harris, *Fundamentals of Queuing Theory*, Wiley, New York, 2003

[Jos03]J. Joseph, C. Fellenstein, *Grid Computing*, Prentice Hall, 2003

[Dod02] Gh. Dodescu, B. Oancea, M. Raceanu, *Procesare paralelă*, Editura Economica, Bucureşti, 2002

[Jor02] H. F. Jordan, H. E. Jordan, *Fundamentals of Parallel Computing*, Prentice Hall, 2002

[Tan99] A. S. Tanenbaum, *Organizarea Structurată a Calculatoarelor*, Computer Press Agora, 1999