Konkurentno programiranje - Komunikacija, sinhronizacija

Nikolina Ležaić i Ana Pantić
mr16113@alas.matf.bg.ac.rs mn16361@alas.matf.bg.ac.rs

22. novembar 2018.

1 Uvod

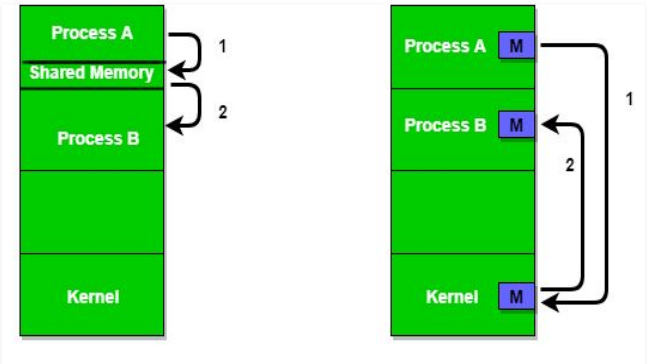
Današnji računarski sistemi omogućuju da se u memoriji istovremeno nalazi više programa koji se izvršavaju konkurentno. Konkurentno programiranje karakteriše više procesa koji se izvršavaju u istom vremenskom periodu, a koji imaju isti cilj. Česte su situacije u kojima jedan proces mora da razmenjuje informacije sa drugim. Da bi se to pravilno izvršavalo, neophodno je da postoji komunikacija između njih. Komunikacija može da se odvija između procesa koji se izršavaju na istom računaru ili na različitim računarskim sistemima u mreži.

U jednoprocesorskom sistemu kod konkurentnog programiranja procesi se prepliću u vremenu da bi se stvorio privid istovremenog izvršavanja. Iako to nije paralelno izvršavanje i postoji gubitak vremena na menjanju procesa koji se izvršavaju, konkurentnim programiranjem se može povećavati efikasnost programa. Za razliku od jednoprocesorskih sistema, u višeprocesorskim je moguća i paralelna obrada tj. procesi se mogu izvršavati istovremeno.

Osnovni problem konkurentnog programiranja je što se brzina izvršavanja jednog procesa ne može predvideti jer zavisi od aktivnosti drugih procesa, operativnog sistema i od prioriteta procesa. Prema tome, ukoliko više procesa pristupa istim podacima, posledica konkurentnog pristupa tim podacima može biti njihova nekonzistentnost. Zbog toga je potrebno da postoji mehanizam koji će obezbediti da više procesa istovremeno ne pristupa istoj memorijskoj lokaciji. To predstavlja problem sinhronizacije.

2 Komunikacija

Komunikacija se odnosi na bilo koji mehanizam koji dozvoljava jednoj niti da dobije informacije koje proizvodi druga. Mehanizmi komunikacije za imperativne programe generalno se zasnivaju ili na deljenoj (zajedničkoj) memoriji ili na slanju poruka. U programskom modelu deljene memorije neke ili sve promenljive programa su dostupne za više niti i da bi par niti komunicirao, jedna od njih piše vrednost promenljive, a druga je čita, dok u programskom modelu koji prenosi poruke, jedna od njih mora izvršiti eksplicitnu operaciju slanja da bi prenela podatke drugoj. Slanje poruka može se ostvariti na razne načine (na primer, u okviru međuprocesne komunikacije, mogu se koristiti tokovi, signali, cevi, soketi, kanali itd.). Što se tiče pouzdanosti slanja poruka, ukoliko se slanje vrši u okviru iste mašine, onda se ono smatra pouzdanim, dok u distribuiranim sistemima nije pouzdano jer podaci putuju kroz mrežu gde se mogu zagubiti, pa se tu koriste različiti protokoli [1].



Slika 1: deljena memorija i slanje poruka

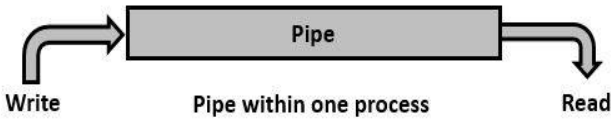
2.1 Tokovi

Tokovi podataka (obično pojedinačnih bajtova ili karaktera) se koriste u modelovanju ulaza i izlaza. Standardni ulaz su obično podaci koji se unose preko tastature koji se upućuju na standardni izlaz koji se prikazuje na ekranu. Pored standardnog izlaza postoji i standardni izlaz za greške koji nas obaveštava o greškama nastalim tokom rada programa koji se takođe prikazuje na ekranu. U mnogim okruženjima moguće je izvršiti preusmeravanje standardnog ulaza tako da se umesto sa tastature, karakteri čitaju iz neke datoteke:
./program <infile
Takođe, u većini programa moguće je izvršiti preusmeravanje standardnog izlaza u neku datoteku:
./program >outfile [2]

2.2 Cevi

Cevi predstavljaju oblik komunikacije koji se izvršava između dva ili više srodnih ili međusobno povezanih procesa. Komunikacija se odvija jednosmerno

tako što jedan proces upisuje u cev, a drugi čita podatke iz nje. Otvorena cev predstavlja oblast glavne memorije koja se koristi dok je čitanje i pisanje aktivno. Ako proces pokuša da pročita pre nego što se nešto upiše u cev, tada se čitanje prekida dok se nešto ne upiše. Da bi se ostvario sistemski poziv cevi, potrebno je kreirati dve datoteke: jedna se koristi za pisanje, a druga za čitanje iz datoteke [3].



Slika 2: Cevi

2.3 Soketi

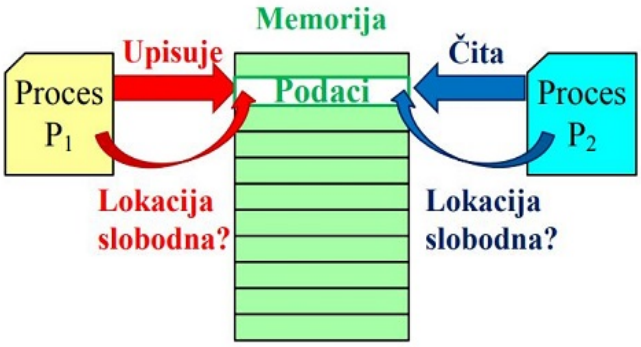
Soket je, u širem smislu, mehanizam koji omogućava komunikaciju između programa koji se izvršavaju na različitim računarima u mreži, a u užem predstavljaju objekte pomoću kojih se šalju (prihvataju) podaci ka (od) drugim soketima. Soket je po svojoj prirodi ulazno-izlazni tok i on se ponaša na sličan način kao sistemski objekti System.in i System.out, pomoću kojih se podaci prihvataju sa standardnog ulaza i šalju ka standardnom izlazu [4].

3 Sinhronizacija

Procesi mogu biti međusobno nezavisni ili zavisni. Ukoliko se procesi izvršavaju uporedo i nezavisno, onda nema interakcije niti razmene informacija između njih, pa rezultat izvršavanja procesa ne zavisi od redosleda izvršavanja i preplitanja sa drugim nezavisnim procesima. Nas interesuje slučaj kada (kooperativni) procesi razmenjuju informacije jer je tada potrebno uvesti međusobnu sinhronizaciju. S druge strane, kod ovih procesa (koji dele podatke) rezultat izvršavanja zavisi od redosleda izvršavanja i preplitanja. Kao što je već rečeno, problem koji se javlja je nekonzistentan rezultat, pa je uočavanje i ispravljanje grešaka veoma teško [5]. Sinhronizacija se odnosi na bilo koji mehanizam koji dozvoljava programeru da kontroliše relativni red u kojem se operacije javljaju u različitim nitima (slika 3). Sinhronizacija je generalno implicitna u modelima koji preuzimaju poruke: poruka mora biti poslata pre nego što se može primiti. Ako nit pokušava da primi poruku koja još nije poslata, sačekaće da bude poslata. Sinhronizacija nije implicitna u modelima deljene memorije [1].

Osnovno pravilo svih zavisnih - konkurentnih procesa glasi:

"Rezultat programa ne sme da zavisi od redosleda izvršavanja i preplitanja, tj. od načina raspoređivanja procesa" [5].



Slika 3: Mehanizam sinhronizacije

Tri osnovna problema:

- sinhronizacija procesa
- međusobno isključenje procesa
- uzajamno blokiranje-zastoj

Postoje dve vrste sinhronizacije: saradnja i takmičenje [5].

3.1 Saradnja

Problem proizvođača i potrošača (producer/consumer) je jedan od najčešćih problema u konkurentnoj obradi. Javlja se kada imamo više proizvođača koji smeštaju neku vrstu podataka u bafer. Sa druge strane postoje potrošači koji podatke čitaju iz bafera redom, jedan po jedan. Sistem mora da obezbedi preklapanje operacija sa baferom, odnosno u istom trenutku baferu može pristupiti ili proizvođač, ili potrošač.

Problem koji se može javiti kod ovog načina sinhronizacije je ograničenost tj. neograničenost bafera. U slučaju neograničenog bafera proizvođač tada stalno proizvodi podatke jer ne postoji memorijsko ograničenje za bafer, dok u suprotnom slučaju proizvođač može da proizvede samo određenu količinu podataka koja se može smestiti u bafer i nakon toga mora da čeka da ih potrošač upotrebi. Ako je ukupna količina proizvedenih podataka jednaka veličini bafera tada proizvođač zaustavlja proizvodnju. Na sličan način potrošač proverava da li postoje dostupni podaci u baferu, i ukoliko ne postoje mora da sačeka proizvođača da ih proizvede, a ako postoje tada ih koristi [6].

3.2 Takmičenje

Uslovi trke se javljaju kada dve ili više niti mogu pristupiti deljenim podacima i pokušavaju da ih promene istovremeno. Da bismo sprečili nastanak takmičenja, obično zaključavamo deljene podatke kako bismo bili sigurni da samo jedna nit može pristupiti podacima istovremeno [7].

Razlikujemo 4 modela sinhronizacije:

1. semafori
2. monitori
3. muteksi
4. katanci

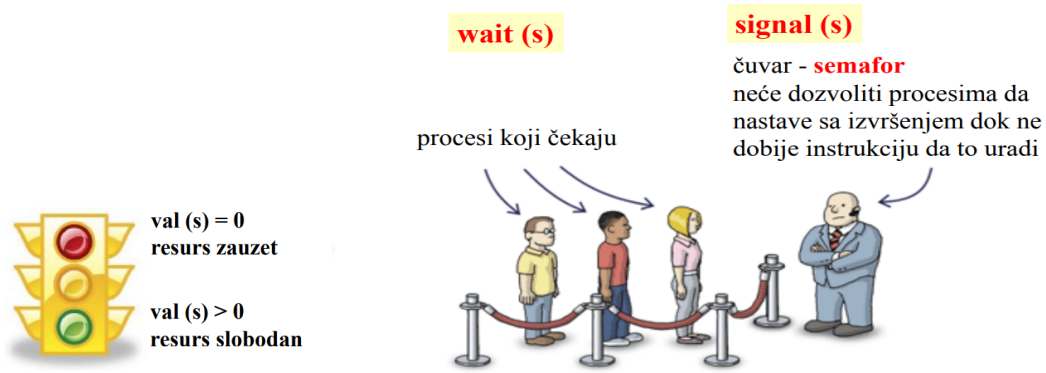
3.2.1 Semafori

Hardverska rešenja problema kritične sekcije [1] kao što su nedeljive instrukcije programerima su prilično teška za korišćenje. Da bi se taj problem prevazišao koristi se tehnika semafora (semaphore). Semafor S je celobrojna promenljiva kojoj se, osim inicijalizacije, pristupa samo preko nedeljive (atomic) operacije: wait() i signal(). Operacija wait() se definiše kao:

```
wait(S) {
       while S <= 0; //ne radi ništa
       S--;
}
```

A operacija signal() kao:

```
  Signal(S) {
     S++;
}
```



Slika 4: semafori

Svaka promena celobrojne vrednosti semafora mora da se obavi nedeljivo, odnosno kada jedan proces menja vrednost semafora, nijedan drugi proces ne sme istovremeno da je menja. Takođe, testiranje celobrojne vredosti semafora u operaciji wait() i njegovo eventualno menjanje (S–) moraju da se obave nedeljivo. Razlikujemo brojačke semafore (counting semaphores) i binarne semafore

[1] kritična sekcija je segment koda koji dovodi do stanja trke [8]

(binary semaphores). Vrednost brojačkog semafora može biti proizvoljna pozitivna vrednost, dok vrednost binarnog semafora može biti 0 ili 1. U nekim sistemima binarni semafori se zovu muteksi (mutex), jer omogućavaju međusobno isključivanje (mutual exclusion). Binarni semafori se koriste za rešavanje problema kritične sekcije za više procesa. Semafore možemo da iskoristimo za rešavanje drugih problema sinhronizacije.

Primer 3.1 *Pretpostavimo da proces P1 izvršava naredbu S1, a proces P2 naredbu S2, i da je zahtev da se naredba S2 izvrši tek pošto se izvrši naredba S1. Problem se može rešiti tako što će P1 i P2 deliti synch koji će biti inicijalizovan na nulu. U proces P1 će biti dodata naredba:*

```
S1;

Signal(synch);
```

A u proces P2:

```
Wait(synch);

S2;
```

3.2.2 Monitori

Monitori predstavljaju konstrukcije u višim programskim jezicima, u kojima se resursi posmatraju kao objekti. Svaki monitor se sastoji od:

1. promenljivih koje opisuju deljeni resurs (objekat), čije vrednosti definišu stanje monitora
2. skupa procedura i funkcija kojima se pristupa objektu (promenljivim monitora)
3. dela programa koji inicijalizuje objekat, a koji se izvršava samo jednom, pri stvaranju objekta.

```
monitor ime_monitora {
/* deklaracija deljenih promenljivih*/
 P1 (...) {/ * definicija procedure/funkcije P1*/}
 P2 (...) {/* definicija procedure/funkcije P2*/}
 ...
Pn (...) {/* definicija procedure/funkcije Pn*/}
{/* inicijalizacija monitora*/} }
```

Dozvoljava da samo jedan proces bude aktivan u monitoru. Ukoliko je potrebna dodatna sinhronizacija, monitoru se moraju dodati uslovne konstrukcije, odnosno definisati uslovne promenljive: [8]

```
condition x, y;
```

3.2.3 Muteksi

Muteksi su objekti koji se koriste za međusobno isključivanje. Iz pojma međusobno isključivanje možemo shvatiti da samo jedan proces može pristupiti datom resursu. Muteks dozvoljava da više niti koriste isti resurs, ali jedna po jedna, ne istovremeno. Kada se program pokrene, zahteva od sistema da kreira muteks objekat za dati resurs. Sistem stvara objekat sa jedinstvenim imenom. Kad god nit pokuša da pristupi resursu, ona zaključava bravu na muteks objektu, koristi resurs i nakon upotrebe oslobađa bravu na objektu. U međuvremenu nijedna druga nit ne može pristupiti tom resursu, već staje u red i čeka da se muteks otključa [9].

➤ **Razlika između semafora i muteksa:**

→ Muteks je ključ u toaletu. Jedna osoba može imati ključ i zauzima toalet u tom trenutku. Kada završi, osoba daje (oslobađa) ključ sledećoj osobi u redu. (u stvari semafor sa vrednošću 1)

→ Semafor je broj besplatnih identičnih ključeva za toalet. Na primer, recimo da imamo 4 toaleta sa identičnim bravama i ključevima. Vrednost semafora – broj ključeva – je podešen na 4 na početku (sva 4 su slobodna), a zatim se vrednost smanjuje kako ljudi dolaze. Ako su svi toaleti zauzeti, tj. više nema slobodnih ključeva, vrednost semafora je 0. Sada, kada jedna osoba oslobodi toalet, vrednost semafora se povećava na 1 (jedan slobodan ključ) i daje se sledećoj osobi u redu [10].

→ Muteks je **mehanizam zaključavanja** koji se koristi za sinhronizaciju pristupa resursu i to je **objekat**, dok je semafor **signalni mehanizam** i predstavlja **celobrojnu vrednost**.

→ Semafor je bolja opcija u slučaju da postoji više instanci resursa, dok u slučaju zajedničkog resursa, bolji izbor je muteks [9].

3.2.4 Katanci

Katanci predstavljaju osnovni način bezbednog deljenja podataka u konkurentnom okruženju, tj. mehanizam sinhronizacije za sprovođenje ograničenja pristupa resursu u okruženju u kojem postoji mnogo niti izvšavanja [11].

Literatura

[1] M. L. Scott, *Programming language pragmatics*. Morgan Kaufmann, 2000.

[2] P. Janičić and F. Marić, *Programiranje 1*. Matematički fakultet Univerziteta u Beogadu, 2015.

[3] "Inter Process Communication - Pipes," on-line at: https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_pipes.htm.

[4] "Rad u mreži," on-line at: http://silab.fon.bg.ac.rs/wp-content/uploads/2016/05/RadUMrezi.pdf.

[5] "Sinhronizacija procesa," on-line at: https://vtsnis.edu.rs/wp-content/plugins/vts-predmeti/uploads/OS%20Predavanje%204%202014.pdf.

[6] "Inter Process Communication," on-line at: https://www.geeksforgeeks.org/inter-process-communication/.

[7] "Race condition," on-line at: https://stackoverflow.com/questions/34510/what-is-a-race-condition.

[8] "Sinhronizacija procesa."

[9] "Difference between semaphore and mutex," on-line at: https://techdifferences.com/difference-between-semaphore-and-mutex.html.

[10] "Difference between binary semaphore and mutex," on-line at: https://stackoverflow.com/questions/62814/difference-between-binary-semaphore-and-mutex.

[11] M. J. Vujošević, *Dizajn programskih jezika*, 2018.