Konkurentnost i sinhronizacija procesa

Vladimir Filipović

Konkurentnost procesa

- Konkurentno ili paralelno izvršavanje više procesa i/ili niti, osim prednosti koje donosi kada je efikasnost sistema u pitanju, može proizvesti dosta problema pri manipulaciji nad zajedničkim podacima.
- Deljeni podaci su veoma osetljivo mesto jer prekidanje procesa koji im je pristupao i propuštanje drugog može dovesti do nekonzistentnosti podatka i neočekivanih grešaka.
- Do sličnih problema može doći i prilikom paralelnog izvršavanja procesa koji modifikuju iste podatke.

Trka za resurse

- Situacija u kojoj krajnji rezultat zavisi od redosleda izvršavanja koraka različitih procesa koji manipulišu zajedničkim podacima naziva se trka za resurse (race condition).

Trka za resurse (2)

Najjednostavniji primer koji ilustruje trku za resurse je problem uvećavanja zajedničke promenljive. Neka je X promenljiva kojoj mogu pristupati različiti procesi koji žele da je uvećaju za jedan:

$$X = X + 1;$$

Operacija sabiranja nije atomična operacija, tj. ne izvršava se „odjednom“ već kroz tri instrukcije, odnosno u tri koraka.

Primer 3.1. **Operacija sabiranja**
1. Smestiti promenljivu *X* u registar *R*; 2. Uvećati registar *R* za 1; 3. Premestiti sadržaj registra *R* u promenljivu *X*;

Trka za resurse (3)

Ako dva procesa konkurišu da obave ovakvu operaciju, može se dogoditi sledeće:

Primer 3.2. **Izvršavanje dva procesa**

Proces 1:
Smestiti promenljivu *X* u registar *R*;
Uvećati registar *R* za 1;
Prekid procesa;

Proces 2:
Smestiti promenljivu *X* u registar *R*;
Uvećati registar *R* za 1;
Premestiti sadržaj registra *R* u promenljivu *X*;
Prekid procesa;

Proces 1:
Premestiti sadržaj registra *R* u promenljivu *X*;

Trka za resurse (4)

Imajući u vidu prethodni primer, može se pretpostaviti da vrednost promenljive X (X ima vrednost 0 na početku) posle izvršavanja k procesa koji konkurentno ili paralelno, uvećavaju promenljivu X u ciklusu od n ponavljanja verovatno neće biti k*n:

Primer 3.3. **Izvršavanje i-tog procesa**
FOR ($j = 0; j < n; j$++) X++; **ENDFOR**

Trka za resurse (5)

Primer potrošača i proizvođača je slikovit i često se koristi kao ilustracija trke za resurse.

Dva procesa imaju zadatak kontrolišu stanje u magacinu. Proizvođački proces ima zadatak da povećava broj proizvoda koji se nalaze u magacinu u trenutku kada se novi proizvod pojavi. Analogno, potrošački proces umanjuje zajedničku promenljivu koja sadrži stanje u magacinu kada se proizvod iznese iz magacina.

Primer 3.4. **Proizvođač – Potrošač**

Inicijalizacija:

```
// Prva slobodna pozicija u magacinu na koju se dodaje novi proizvod
in = 0;
// Pozicija najstarijeg proizvedenog proizvoda u magacinu
out = 0;
// Broj proizvoda u magacinu
brojač = 0;
```

Trka za resurse (6)

Primer 3.4. **Proizvođač – Potrošač**

Proizvođač:

```
WHILE (true)
    Novi proizvod je proizveden;
    // Skladište je puno ako je brojač proizvoda u magacinu jednak broju
    // mesta u magacinu i treba sačekati
    WHILE (brojač == veličina_magacina)
        //Aktivno čekanje dok se ne oslobodi bar jedno mesto u magacinu
    ENDWHILE
    // Novi proizvod se dodaje na prvu sledeću slobodnu poziciju
    magacin[in] = novi  proizvod;
    // Pomera se indeks na sledeću poziciju
    in = (in + 1) mod veličina_magacina;
    // Uvećava se brojač koji govori koliko je proizvoda u magacinu
    brojač++;
ENDWHILE
```

Trka za resurse (7)

Primer 3.4. **Proizvođač – Potrošač**

Potrošač:

```
// Ako je brojač koji broji koliko je proizvoda u skladištu jednak
// nuli to znači da je skladište prazno
WHILE(brojač == 0)
   //Aktivno čekanje da se bar jedan proizvod proizvede
ENDWHILE
// Uzima se najstarije proizvedeni proizvod
prodaja = skladište[out];
// Povećava se brojač koji pokazuje na proizvod koji je
// najduže u magacinu
out = (out + 1) mod veličina_magacina;
// Ažurira se broj proizvoda koji se nalazi u skladištu tako da
// odgovara stvarnom broju tj. umanjuje se za jedan
brojač--;
```

Trka za resurse (8)

Kao i u prethodnom primeru, ni kod proizvođača i potrošača uvećanje i umanjenje za jedan nisu atomične operacije.

Jasno je da će doći do nekonzistentnosti, odnosno do razlike u broju stvarnih proizvoda u magacinu i vrednosti promenljive brojač.

Rešavanje problema ovakve prirode podrazumeva da se implementira neka vrsta zaštite kojom bi se obezbedilo da osetljivim delovima može pristupiti samo jedan proces u jednom trenutku.

Kritična sekcija

- Deo programa u kojem se pristupa zajedničkim podacima naziva se kritična sekcija.
- Dovoljna garancija da ne dođe do neželjenih rezultata ovakvog tipa je da se obezbedi da kritičnu sekciju u jednom trenutku može da izvršava samo jedan proces.
- Međutim, ovakvo rešavanje problema kritične sekcije nije trivijalno, naročito u sistemima gde je efikasnost, odnosno viši stepen mulitiprogramiranja primaran.

Kritična sekcija (2)

- Rešenje koje se prvo nameće u sistemima koji imaju samo jedan procesor je da se onemogući prekidanje procesa koji je u kritičnoj sekciji.
- Na ovaj način proces koji pristupa deljenim podacima neće biti prekinut pa samim tim i ne postoji mogućnost da neki drugi proces pristupi tim podacima.
- Rešenje sa isključivanjem prekida se ne primenjuje često, jer je suština implementacije multiprogramiranja zasnovana na sistemu prekida, pa bi njegovo često isključivanje dovelo do smanjenja efikasnosti sistema.

Kritična sekcija (3)

- Rešenje koje se prvo nameće u sistemima koji imaju samo jedan procesor je da se onemogući prekidanje procesa koji je u kritičnoj sekciji.
- Na ovaj način proces koji pristupa deljenim podacima neće biti prekinut pa samim tim i ne postoji mogućnost da neki drugi proces pristupi tim podacima.
- Rešenje sa isključivanjem prekida se ne primenjuje često, jer je suština implementacije multiprogramiranja zasnovana na sistemu prekida, pa bi njegovo često isključivanje dovelo do smanjenja efikasnosti sistema.

Kritična sekcija (4)

- Rešenje koje podrazumeva isključivanje prekida je pogodno primenjivati u situacijama kada je u kritičnoj sekciji mali broj instrukcija.
- Rešenje sa isključivanjem prekida se ne može koristiti u višeprocesorskim sistemima jer ne pruža garanciju da različiti procesi koji se istovremeno izvršavaju neće modifikovati deljene podatke.
- Zbog prethodno navedenih nedostataka, rešenja sa isključivanjem prekida se vrlo retko primenjuju.

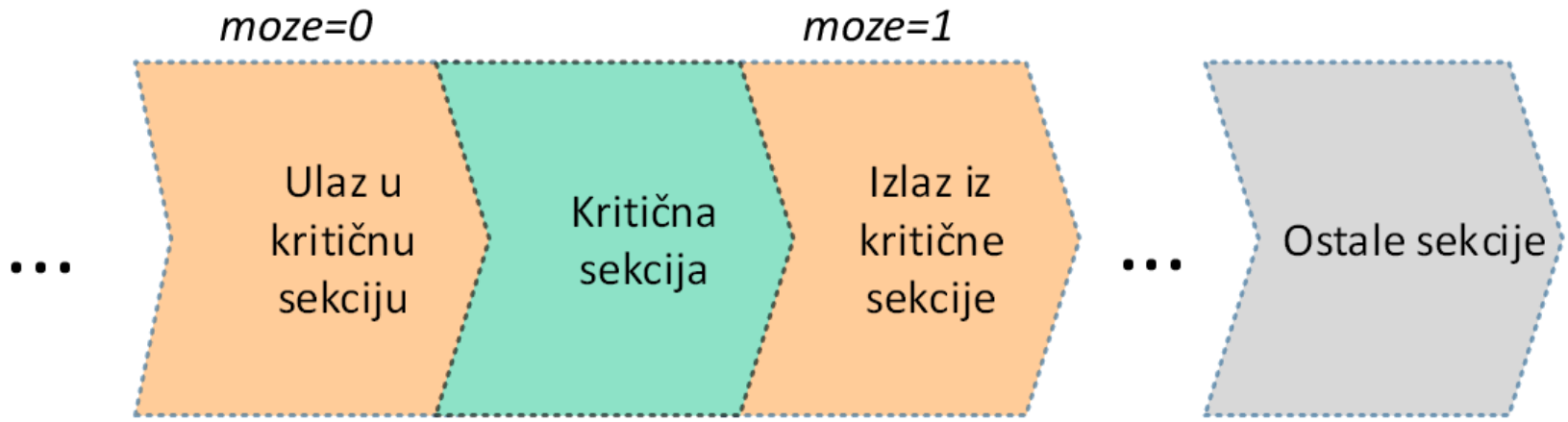
Kritična sekcija (5)

Dobro rešenje za zaštitu kritične sekcije trebalo bi da zadovolji sledeće uslove:

- Uzajamna isključivost, koja podrazumeva da dva procesa ne mogu u isto vreme da budu u kritičnoj sekciji, što je i ključni zahtev bez kojeg nema rešenja za zaštitu kritične sekcije.
- Uslov progresa, koji zahteva da proces koji nije u kritičnoj sekciji i ne želi da uđe u nju ne treba da ometa druge procese da u nju uđu.
- Uslov konačnog čekanja, koji podrazumeva da bi trebalo da postoji razumna granica koliko jedan proces može da čeka na ulazak u kritičnu sekciju, odnosno da se, ni pod kojim uslovima, ne može dogoditi da neki od procesa neograničeno dugo čeka.

Rešenja sa aktivnim čekanjem

- Više rešenja za zaštitu kritične sekcije se zasniva na tzv. aktivnom čekanju, odnosno na čekanju u ciklusu dok se ne stvore uslovi za ulazak procesa u kritičnu sekciju.
- Izgleda da je relativno jednostavno implementirati graničnike koji bi omogućili zaštitu kritične sekcije:



Zaštita kritične sekcije uz pomoć graničnika

Rešenja sa aktivnim čekanjem (2)

- Softverska rešenja su zanovana na softveru.
- Jedno od najpoznatijih je Lamportov algoritam (Leslie Lamport), koji se često naziva i pekarski algoritam, je dobio ovakav „nadimak“ jer je zasnovan na ideji koja se oslanja na redosled opsluživanja mušterija u pekari: kada mušterija uđe u pekaru ona dobije broj, a mušterije se uslužuju redom počev od onih sa nižim brojevima.
- Pošto se ne može garantovati da dva procesa neće dobiti isti broj jer je i dodela brojeva kritična sekcija, u slučaju da se to dogodi proces sa manjim indeksom se opslužuje prvi.

Rešenja sa aktivnim čekanjem (3)

- Lamportov algoritam (Leslie Lamport), koji se često naziva i pekarski algoritam, je dobio ovakav „nadimak“ jer je zasnovan na ideji koja se oslanja na redosled opsluživanja mušterija u pekari: kada mušterija uđe u pekaru ona dobije broj, a mušterije se uslužuju redom počev od onih sa nižim brojevima.
- Pošto se ne može garantovati da dva procesa neće dobiti isti broj jer je i dodela brojeva kritična sekcija, u slučaju da se to dogodi proces sa manjim indeksom se opslužuje prvi.

Rešenja sa aktivnim čekanjem (4)

Za jednostavniji prikaz algoritma se uvode sledeće oznake:

$(A, B) < (C, D)$ ako $A < C$ ili $A = C$ i $B < D$

$\max(A1, A2, ..., An) = K$, gde je $K \geq Ai$, $i = 1, 2, \ldots, n$

Globalne promenljive:
celobrojni niz (broj[n])
niz logičkih promenljivih (uzima[n])
i oba niza su inicijalizovana na nulu.

Algoritam 3.6. **Lamportov - Pekarski algoritam**

```
Proces i:
  // Zaštita procesa uzimanja broja
  uzima[i] = 1;
  broj[i] = max(broj[0], ..., broj[n – 1]);
  uzima[i] = 0;
  // Glavna petlja koja je graničnik kritične sekcije
  FOR (j = 0; j < n; j++)
    WHILE (uzima[j] == 1)
      // Aktivno čekanje da j-ti proces dobije broj
    ENDWHILE
    WHILE (broj[j] != 0 AND (broj[j], j) < (broj[i], i)
      // Aktivno čekanje da proces koji ima prednost završi
    ENDWHILE
  ENDFOR
  // Kritična sekcija
  broj[i] = 0;
```

Rešenja sa aktivnim čekanjem (5)

- Opis koraka algoritma za i-ti proces:
  - Uz pomoć promenljive uzima[i] štiti se proces dobijanja broja za i-ti proces (tj. broj[i]) koji dobija vrednost koja je za jedan veća od trenutno najvećeg broja koju imaju ostali procesi (članovi niza broj).
  - Zatim se u ciklusu obilaze svi procesi i čeka se da sa izvršavanjem u kritičnoj sekciji završe oni koji imaju bolji broj ili niži redni broj procesa u slučaju kada su brojevi isti, od i-tog procesa.
  - Proces i treba prvo da sačeka da j-ti proces završi sa uzimanjem broja ukoliko je u toj fazi, a zatim i da, ukoliko j-ti proces ima prednost, sačeka da on završi sa pristupom kritičnoj sekciji i postavi broj[j] na nulu.
  - Potom on pristupa kritičnoj sekciji i kada završi sa pristupom broj[i] vraća na nulu.

Rešenja sa aktivnim čekanjem (6)

- Hardverska rešenja za zaštitu kritične sekcije podrazumevaju upotrebu posebnog hardvera.
- Ideja je da se naprave mašinske instrukcije koje su u stanju da urade bar dve operacije bez mogućnosti prekida, atomično.
- Najčešće se za potrebe zaštite kritične sekcije koriste sledeće tri instrukcije:
  - TAS (Test And Set),
  - FAA (Fetch And Add) i
  - SWAP (zamena)

Rešenja sa aktivnim čekanjem (7)

- Instrukcija TAS operiše sa dve promenljive.
  A = TAS(B) - funkcioniše tako što se vrednost koja se nalazi u promenljivoj B prebacuje u promenljivu A, a u promenljivu B se postavlja 1.
  Obično su promenljive A i B logičkog tipa.
- Instrukcija FAA takođe operiše sa dve promenljive. Njena sintaksa je FAA(A, B) - pri njenom korišćenju se vrednost promenljive B prebacuje u A, a vrednost B+(originalna vrednost)A u promenljivu B.
- SWAP operiše sa dve promenljive.
  SWAP(A, B) - rezultat je atomična zamena vrednosti promenljivih A i B.

Rešenja sa aktivnim čekanjem (8)

- Postoje različita rešenja za zaštitu kritične sekcije korišćenjem ovakvih instrukcija ali je u osnovi ideja svih slična.
- Obično postoji jedna promenljiva kojom se štiti ulaz u kritičnu sekciju i procesi aktivno čekaju da se ulaz oslobodi, a kada se to desi onda atomičnost ovih instrukcija omogućava da tačno jedan proces uđe u kritičnu sekciju.

Rešenja sa aktivnim čekanjem (9)

Algoritam koji sledi prikazuje jedno od rešenja za zaštitu kritične sekcije korišćenjem hardverske instrukcije TAS.

Deklariše se globalna promenljiva logičkog tipa zauzeto i postavi se na nulu, dok svaki proces ima lokalnu promenljivu takođe logičkog tipa ne_može.

Algoritam 3.7. **TAS**

```
Proces i:
    ne_može = 1;
    WHILE (ne_može == 1)
        ne_može = TAS(zauzeto);
    ENDWHILE
    // Kritična sekcija
    zauzeto = 0;
```

Kada se oslobodi ulaz, proces atomičnom operacijom uviđa da je ulaz slobodan (ne_može postaje 0) i zatvara kritičnu sekciju za ostale procese jer TAS operacija zauzeto postavlja na 1.

Rešenja sa aktivnim čekanjem (10)

Uslovna mana ovakvog rešenja je činjenica da je to rešenje zasnovano na hardverskoj podršci, odnosno da je potrebno da procesor ima ugrađenu tu instrukciju.

Sa druge strane ovo rešenje se bez bilo kakve modifikacije može primeniti na proizvoljan broj procesa, ali ne postoji garancija koliko će neki proces da čeka.

„Izgladnjivanje“ procesa, tj. dugo čekanje procesa da uđe u kritičnu sekciju, dok drugi procesi prolaze jer imaju više „sreće“ je moguće ali se u praksi retko događa.

Postoje i modifikacije prethodnog algoritma, koje obezbeđuju garanciju da će proces konačno dugo čekati na ulazak u kritičnu sekciju.

Rešenja sa aktivnim čekanjem (11)

Zaštita kritične sekcije se može obezbediti korišćenjem SWAP atomične operacije.

Algoritam 3.9. **SWAP**

```
// Brava je globalna promenljiva inicijalizovana na 0
Proces i:
    ključ = 1;
    REPEAT
        SWAP (brava, ključ);
    UNTIL (ključ == 0)
    // Kritična sekcija
    brava = 0;
```

Brava se postavlja na nulu što označava da je prolaz u kritičnu sekciju slobodan, odnosno da je brava bez ključa.

Sa druge strane, procesi koji se nadmeću za pristup kritičnoj sekciji imaju promenljivu ključ čija vrednost 1 govori da ključ nije u bravi odnosno da čeka da uđe u kritičnu sekciju.

Rešenja sa aktivnim čekanjem (12)

Procesi u petlji pokušavaju da kroz SWAP operaciju ubace ključ u bravu, odnosno da ugrabe trenutak kada je vrednost promenjive *brava* jednaka 0 i da onda (atomičnom) zamenom te vrednosti sa 1 koja se nalazi u promenljivoj *ključ* dobiju uslov za izlazak iz ciklusa i pristupe kritičnoj sekciji.

Istovremeno se brava „zaključava“, tj. promenljiva *brava* dobija prethodnu vrednost promenljive *ključ*, koja je bila 1. Na ovaj način se ulazak u kritičnu sekciju blokira sve dok proces koji je u njoj ne završi i postavi promenljivu *brava* na 0 čime se otvara mogućnost da neki drugi proces uđe u kritičnu sekciju.

Rešenja bez aktivnog čekanja

- Kod rešenja zasnovanih na aktivnom čekanju se u petlji proveravao neki uslov i na taj način čekalo da se ulaz u kritičnu sekciju oslobodi.
- Postoje i rešenja koja su efikasnija, tj. ne sadrže cikluse koji bespotrebno troše procesorsko vreme.
- Ove metode se obično zasnivaju na tehnikama koje podrazumevaju zaustavljanje procesa bez aktivnog čekanja i njihovo pokretanje u odgovarajućem trenutku.

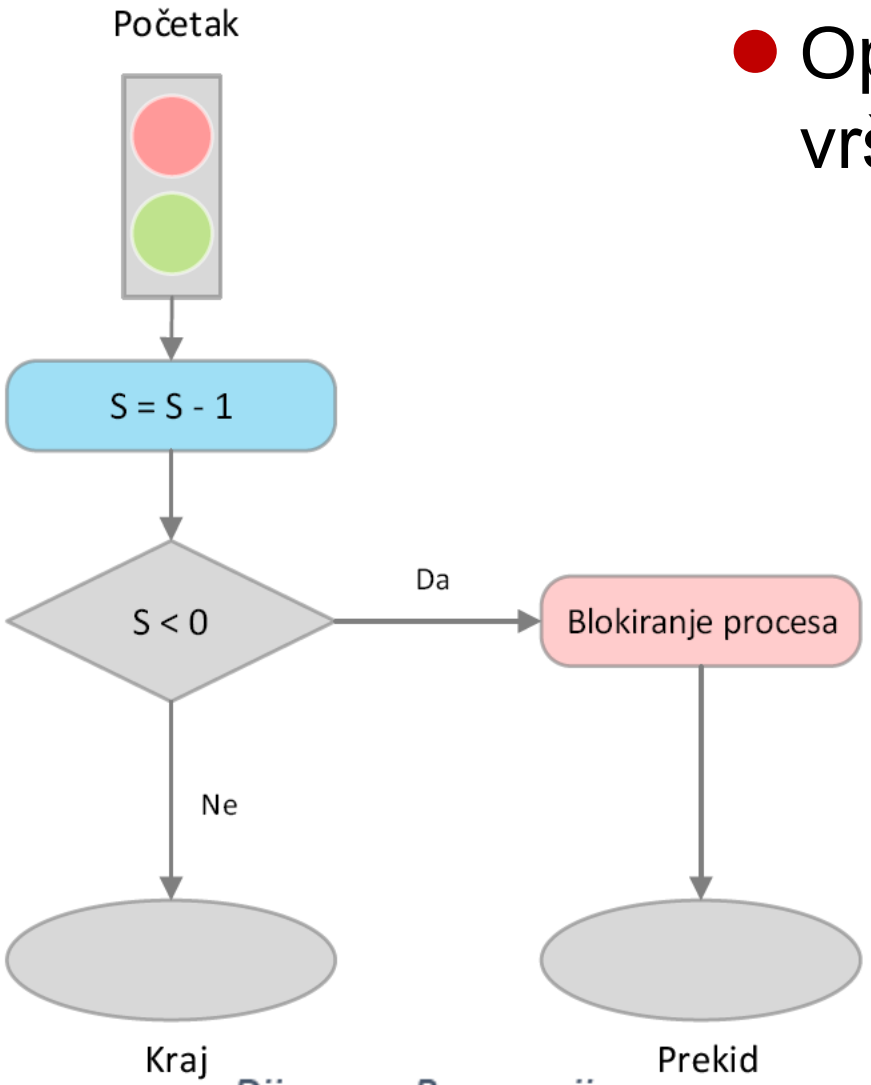
Rešenja bez aktivnog čekanja (2)

- Ideja je da se proces koji ne može da uđe u kritičnu sekciju na neki način blokira pa da se po sticanju uslova da uđe u kritičnu sekciju „probudi“, odnosno „prozove“ i odblokira.
- Ovakva rešenja su bolja od aktivnog čekanja ali je njihova implementacija komplikovanija.
- Najpoznatiji metodi kada je ovakav pristup u pitanju su:
  - Semafori;
  - Kritični regioni;
  - Monitori.

Semafori

- Osmislio ih je E. Dijkstra 1965. godine.
- Semafor je apstraktni tip podataka, odnosno struktura koja može da blokira proces na neko vreme i da ga propusti kada se steknu određeni uslovi.
- Semafor se može definisati kao specijalna, celobrojna, nenegativna promenljiva nad kojom se, pored standardnih operacija kreiranja i oslobađanja mogu izvesti samo još dve operacije: P i V (Up i Down, odnosno Wait i Signal).

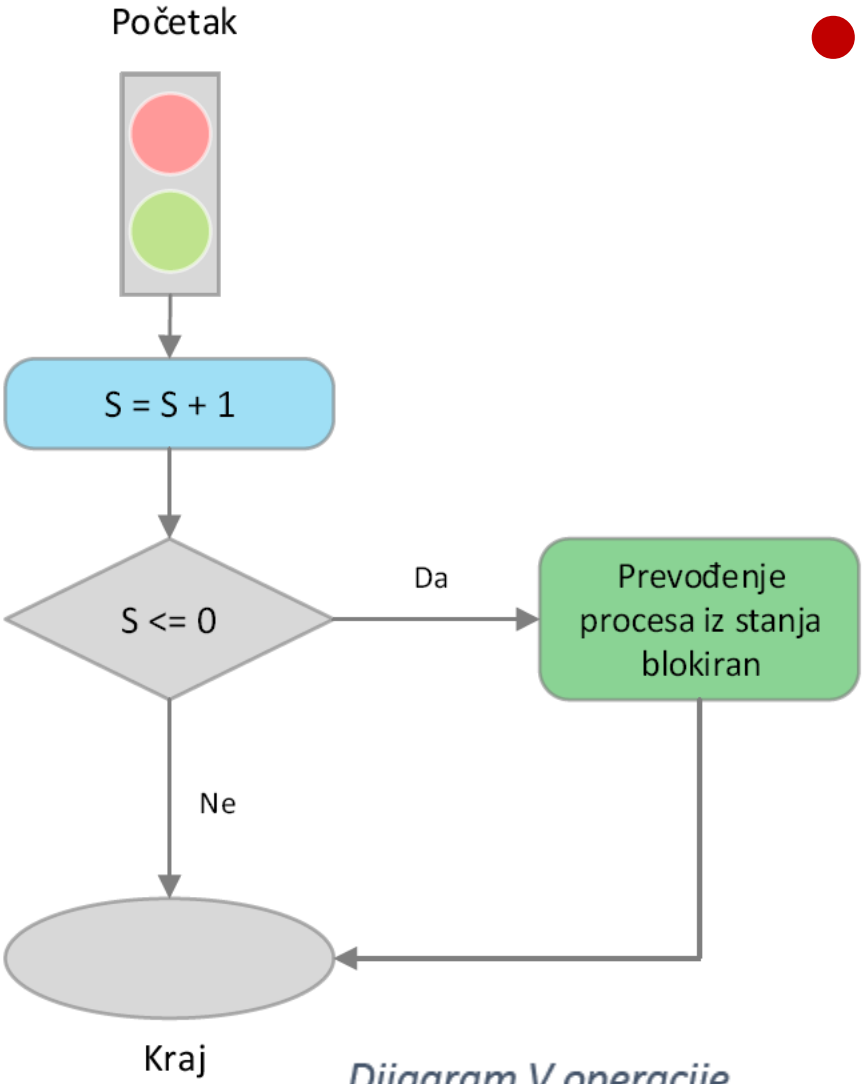
Semafori (2)



Dijagram P operacije

- Operacijom P(S) proces koji vrši tu operaciju:
  - testira da li je vrednost promenljive-semafora S pozitivna;
  - u slučaju da jeste proces se propušta dalje (u kritičnu sekciju) i vrednost semafora se umanjuje za jedan;
  - u suprotnom se proces blokira i stavlja u red za čekanje.

Semafori (3)

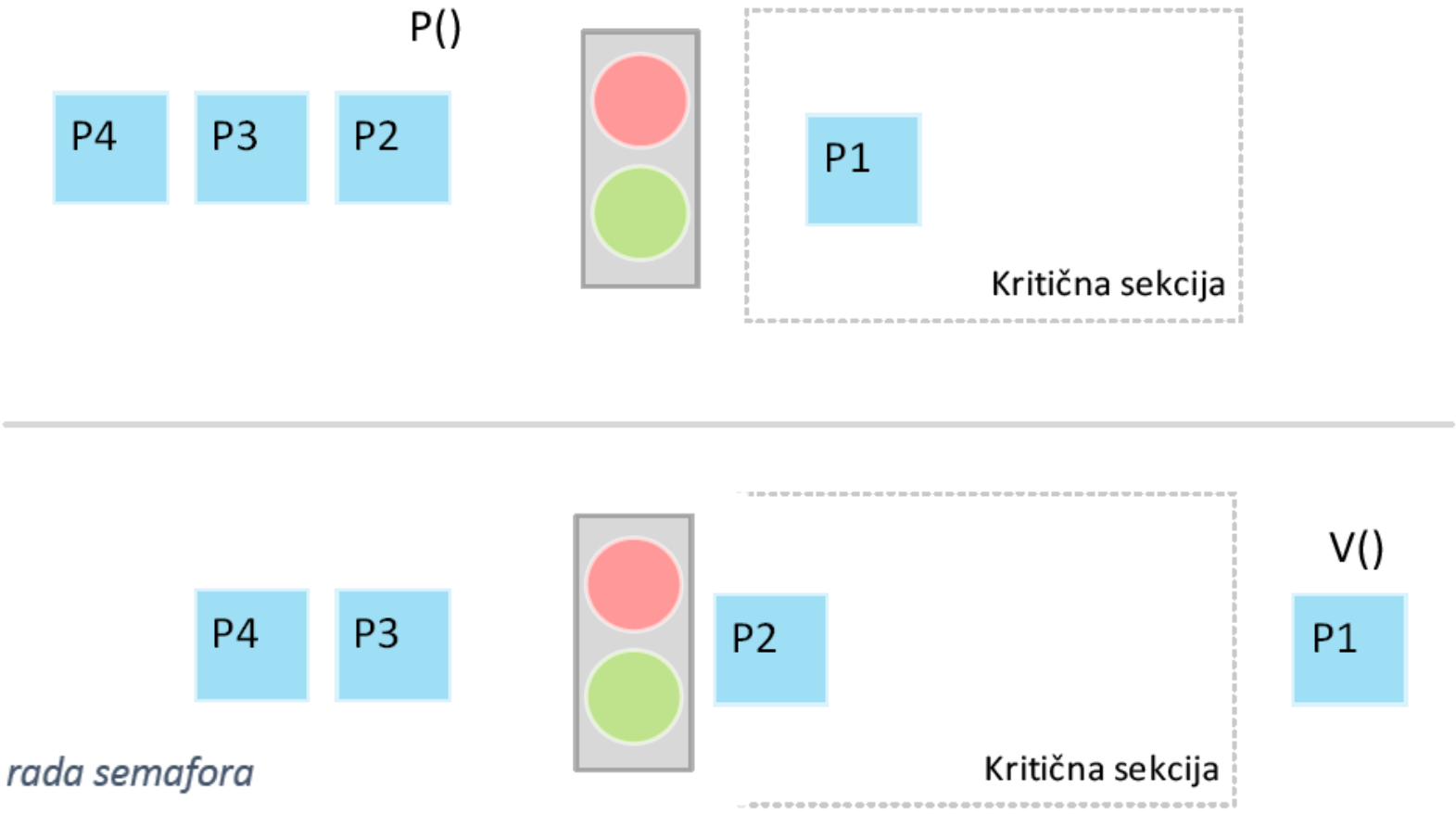


Dijagram V operacije

- Analogno, operacijom V(S) se:
  - ako postoje procesi koji čekaju blokirani na semaforu S (tj. vrednost semafora nije pozitivna) propušta tačno jedan od procesa da nastavi sa izvršavanjem (uđe u kritičnu sekciju), odnosno prekida se njegovo blokiranje;
  - u suprotnom se vrednost semafora povećava za jedan.

Semafori (4)

- Zaštita kritične sekcije se može postići tako što se pre ulaza u kritičnu sekciju postavi binarni semafor.



Princip rada semafora

Semafori (5)

- Semafori mogu biti:
  - opšti ili brojački - semafor ima proizvoljnu nenegativnu celobrojnu vrednost ili
  - binarni - mogu samo imati vrednost 0 ili 1.

Algoritam 3.10. **Primena semafora**

Proces i:

```
//Provera semafora
P(S);
// Kritična sekcija
V(S);
```

Semafori (6)

- Semafori se obično realizuju kao servis operativnog sistema.
- Za njihovu implementaciju se koriste strukture koje osim promenljive (S) najčešće sadrže i pokazivač na listu koja bi trebalo da čuva podatke o procesima koji su blokirani na semaforu.

Primer 3.6. **semafor**

```
semafor: struktura
    vrednost: Ceo broj;
    lista: Lista<Proces>;
```

Semafori (7)

Algoritam 3.11. **Opis implementacije semafora**

```
P(S):
    // Semafor se umanjuje za jedan
    S.vrednost = S.vrednost – 1;
    IF (S < 0)
        // Ako je Vrednost negativna znači da je drugi proces u kritičnoj sekciji i da
        //ovaj treba blokirati i ostaviti u redu za čekanje
        dodati trenutni proces u S.lista;
        blokirati proces;
    ENDIF

V(S):
    S.vrednost = S.vrednost + 1;
    IF ( S <= 0)
        // Skida se jedan proces sa liste i prekida njegovo blokiranje
        izbaciti proces iz S.lista;
        odblokirati taj proces;
    ENDIF
```

Korišćenje semafora

- Korisni u situacijama kada je potrebno da se jedna funkcija izvrši pre druge.
  - Neka su F1 i F2 funkcije pri čemu je potrebno da se prvo izvrši F1 pa tek onda F2.
  - S je semafor koji je inicjalno postavljen na 0.
  - Ukoliko drugi proces dođe do semafora pre nego što se završi funkcija F1, on će tu biti blokiran dok se ne izvrši F1 a onda prvi proces uveća vrednost semafora.

Primer 3.7. **Primena semafora u redosledu izvršavanja funkcija**	
Proces 1: F1(); V(*S*)	**Proces 2:** P(*S*); F2();

Korišćenje semafora (2)

- Implementacija brojačkih semafora uz pomoć binarnih.
  - potrebne dva binarna semafora S1 i S2 i promenljiva C celobrojnog tipa.
  - Pri inicijalizaciji se semafor S1 postavlja na 1, semafor S2 na 0, dok promenljiva C dobija vrednost n.
  - Semafor S1 se koristi da obezbedi ekskluzivni pristup instrukcijama u okviru aritmetičkih operacija nad promenljivom C brojačkog semafora odnosno, da obezbedi atomičnost ovih operacija.
  - Semafor S2 se koristi za čekanje procesa.
  - Promenljiva C čuva vrednost brojačkog semafora odnosno određuje koliko procesa semafor može da propusti.

Korišćenje semafora (3)

Algoritam 3.12. **Implementacija brojačkog semafora korišćenjem binarnih**

```
P(C):
    //Ekskluzivni pristup
    P(S1);
    //Umanjenje brojačke promenljive
    C--;
    //Ako je C negativno semafor
    //nije otvoren
    IF (C < 0)
        // Blokira se proces
        V(S1);
        P(S2);
    ENDIF
        V(S1);
```

```
V(C):
    //Ekskluzivni pristup
    P(S1);
    //Uvećanje brojačke promenljive
    C++;
    //Ako C nije pozitivno postoje
    //procesi koji čekaju
    IF (C <= 0)
        //Oslobađanje jednog procesa
        //koji čeka
        V(S2);
    ELSE
    //Inače se oslobađa semafor koji
    //štiti ekskluzivni pristup
        V(S1);
    ENDIF
```

Korišćenje semafora (4)

- Rešavanje problema proizvođača i potrošača.
  - Problem je definisan na početku prezentacije. Tu se radio o istovremenom pristupu promenljivoj brojač koja bi trebalo da čuva informaciju o trenutnom stanju u magacinu.
  - Za rešavanje je potreban jedan semafor logičkog tipa (nazvan mutex) koji obezbeđuje rad sa skladištem, tj. štiti od istovremenog pristupa više procesa.
  - Potrebne su i dva brojačka semafora (nazvani pun i prazan) koji će čuvati informaciju o broju praznih i punih mesta u magacinu.

Korišćenje semafora (5)

Primer 3.8. **Proizvođač – potrošač korišćenjem semafora**

Glavni program:

```
//Inicijalizacija semafora
mutex = 1;
pun = 0;
prazan  = veličina_magacina;
//Proizvođački i potrošački proces se izvršavaju paraleleno
PARALLEL WHILE (true)
   Proizvođač;
   Potrošač;
END PARALLEL WHILE
```

Korišćenje semafora (6)

Proizvođač:

```
//Proverava da li ima mesta u magacinu
P(prazan);
//Zaštita ekskluzivnog pristupa magacinu
P(mutex);
//Pomera se indeks na sledeću poziciju
in = (in + 1) mod veličina_magacina;
//Uvećanje brojača za jedan
brojač++;
V(mutex);
V(pun);
```

Korišćenje semafora (7)

Potrošač:

```
//Provera da li ima proizvoda u magacinu
P(pun);
//Zaštita ekskluzivnog pristupa magacinu
P(mutex);
//Umanjivanje brojača za jedan
brojač--;
V(mutex);
V(prazn);
```

Kritični regioni

- Semafori su dobro rešenje za zaštitu kritične sekcije ali pri njihovom korišćenju treba biti veoma oprezan - pogrešna oznaka na početku ili kraju kritične sekcije može da je potpuno otvori ili zatvoriti za sve procese.
- Brinč Hansen (Per Brinch Hansen) i Hor (Charles Antony Richard Hoare) su 1971. predložili rešenje u vidu kritičnih regiona.
- Kritični regioni predstavljaju implementaciju zaštite pristupa kritičnoj sekciji na višem programskom jeziku.

Kritični regioni (2)

- Kod ovog rešenja potrebno je definisati koju promenljivu deli više procesa i oznakom region obezbediti ekskluzivni pristup toj promenljivoj u okviru niza naredbi koje se nalaze u tom regionu.

Primer 3.13. **Kritični region**

```
//Definicija deljene promenljive proizvoljnog tipa
v: deljeni tip;
// Označava se region gde je promenljiva v zaštićena
REGION v
   naredba 1;
   .
   .
   .
   naredba n;
ENDREGION
```

Kritični regioni (3)

- Kod ovog rešenja potrebno je definisati koju promenljivu deli više procesa i oznakom region obezbediti ekskluzivni pristup toj promenljivoj u okviru niza naredbi koje se nalaze u tom regionu.
- Kritični regioni su implementirani tako da daju garanciju da dok se izvršavaju naredbe u regionu nijedan drugi proces ne može da pristupa deljenoj promenljivoj (v). Takođe, deljenoj promenljivoj se može pristupati samo u okviru nekog regiona.

Kritični regioni (4)

Primer kojim se najjednostavnije može ilustrovati primena kritičnih regiona je uvećanje promenljive x za jedan.

Primer 3.14. **Atomična operacija uvećavanja**

```
x: deljeni ceo broj;
REGION x
   x = x + 1;
ENDREGION
```

- Promenljiva se prvo proglasi deljenom a onda se označi region tj. operacija uvećanja gde se garantuje ekskluzivni pristup i samim time atomičnost operacije uvećanja.

Kritični regioni (5)

- Kritični regioni rešavaju problem kritične sekcije, ali je sinhronizacija procesa slaba tačka ovog pristupa.
- Napredniji koncepti podrazumevaju uslovne kritične regione (uveo ih je Hor 1971.), koji se aktiviraju samo u slučaju kada je određeni logički uslov tačan.

Kritični regioni (6)

Sintaksa uslovnog kritičnog regiona.

Primer 3.15. **Uslovni kritični region**

```
v: deljeni tip;
REGION v WHILE (uslov)
   naredba 1;
   .
   .
   .
   naredba n;
ENDREGION
```

- Ako je uslov ispunjen i ako nema drugog procesa koji pristupa promenljivoj v, proces ima pravo da pristupi regionu i u njemu ima ekskluzivni pristup nad v.
- Ako uslov nije ispunjen ili ako drugi proces ima pristup nad v, ulazak procesa u kritični region se odlaže.

Kritični regioni (7)

Primena uslovnih kritičnih regiona za rešavanje problema potrošača i proizvođača.

Primer 3.16. **Proizvođač-potrošač**

```
bafer: deljena struktura
    skladište[n]: Proizvod;
    unutra, van, broj: Ceo broj;

Proizvođač:
    REGION bafer WHILE (broj < n)
        //Stavljanje novog proizvoda u magacin
        skladiste[unutra] = novi proizvod;
        unutra = (unutra + 1) mod n;
        broj++;
    ENDREGION
```

Kritični regioni (8)

Primena uslovnih kritičnih regiona za rešavanje problema potrošača i proizvođača.

Primer 3.16. **Proizvođač-potrošač**

```
bafer: deljena struktura
    skladište[n]: Proizvod;
    unutra, van, broj: Ceo broj;

Proizvođač:
    REGION bafer WHILE (broj < n)
        //Stavljanje novog proizvoda u magacin
        skladiste[unutra] = novi proizvod;
        unutra = (unutra + 1) mod n;
        broj++;
    ENDREGION
```

Kritični regioni (9)

Primena uslovnih kritičnih regiona za rešavanje problema potrošača i proizvođača.

Potrošač:

```
REGION bafer WHILE (broj > 0)
    //Uzimanje  proizvoda iz magacina
    proizvod = skladiste[van];
    van = (van + 1) mod n;
    broj--;
ENDREGION
```

Monitori

- Kod zaštite kritične sekcije bez aktivnog čekanja monitori predstavljaju najviši nivo apstrakcije.
- Oni su konstrukcije programskih jezika u okviru kojih su implementirani mehanizmi za zaštitu kritične sekcije ali i za sinhronizaciju.
- Korišćenje monitora predložili su Hor (1974.) i Brinč Hansen (1975.).
- Monitori predstavljaju konstrukcije programskog jezika, slične klasama u objektno orjentisanom programiranju, koje mogu da sadrže procedure, promenljive i proceduru inicijalizacije.

Monitori (2)

- U okviru monitora u jednom trenutku može da bude aktivan samo jedan proces, čime se obezbeđuje zaštita bilo kojeg dela programa pa i kritične sekcije.
- Za potrebe sinhronizacije implementirane su specijalne uslovne promenljive.
  - Nad ovakvim promenljivama su definisane dve operacije: wait i signal, čija sintaksa zavisno od implementacije može da bude: x.wait() ili wait(x) odnosno x.signal() ili signal(x).
  - Korišćenjem ovih promenljivih i operacija proces se može blokirati operacijom x.wait() i kasnije odblokirati signalom x.signal() iz drugog procesa.

Monitori (3)

- Uslovne promenljive monitora nisu ekvivalentne brojačkim semaforima jer kod njih čekanje procesa mora da nastupi pre slanja signala:
  - ako neki proces čeka na signal, poslati signal će probuditi taj proces;
  - u suprotnom, ako nijedan proces nije blokiran (ne čeka signal), poslati signal će biti izgubljen.
- Jedan poslati signal može probuditi tačno jedan proces koji na njega čeka dok ostali blokirani procesi ostaju da čekaju naredne signale.

Monitori (4)

- Procesi koji čekaju na uslovnu promenljivu monitora se obično organizuju u strukturu nazvanu red čekanja.
- Redovi čekanja procesa na uslovnu promenljivu se obično organizuju po FCFS (First Come First Served) algoritmu odnosno prednost će imati onaj proces koji je ranije blokiran na uslovnoj promenljivoj.
- Hor je 1974. je uveo prioritetno čekanje, odnosno x.wait(c) gde je c celobrojni izraz koji predstavlja nivo prioriteta i koji se izračunava u momentu izvršenja operacije.

Monitori (5)

Ilustracija sadržaja monitora.

Primer 3.19. **monitor**

```
monitor: klasa
    S, P: uslovna promenljiva;

    FUNCTION P1()
        ...
    END FUNCTION

    FUNCTION P()
        ...
    END FUNCTION
    .
    .
    .
    FUNCTION Pn()
        ...
    END FUNCTION
```

Monitori (6)

Ilustracija sadržaja monitora.

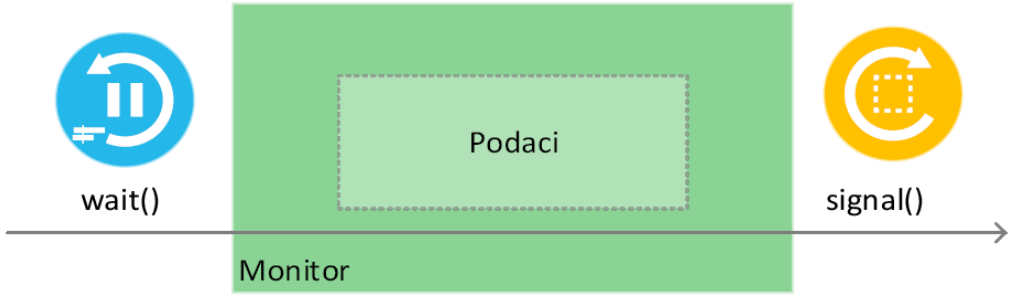
```
FUNCTION Inicijalizacija()
   ...
END FUNCTION
```

Monitori (7)

- Tačno jedan proces može pozvati funkcije koje se nalaze u okviru monitora.
- Ostali procesi ne mogu pozvati ni jednu funkciju iz monitora dok proces koji je pristupio monitoru ne završi sa radom u njemu.
- Sa promenljivama u okviru monitora mogu raditi isključivo procedure iz monitora.
- Pristup kritičnoj sekciji se lako rešava uz pomoć monitora - dovoljno je kritičnu sekciju ubaciti u monitor.
- Prevodilac na poseban način označava monitore i o njima više ne brine programer.

Monitori (8)

- Monitori su karakteristika programskih jezika jer se implementiraju nivou programskih jezika.
- Monitori pružaju više mogućnosti od semafora (koji se implementiraju na nivou operativnog sistema), a i sa njima se jednostavnije radi.
- Prednost monitora u odnosu na kritične regione je postojanje mehanizama (wait i signal) koji olakšavaju sinhronizaciju.



Korišćenje monitora

Korišćenje monitora može se ilustrovati na problemu potrošača i proizvođača.

Primer 3.20. **Proizvođač – potrošač**

```
pun, prazan: uslovne promenljive int;
ima_ih: Ceo broj;

Proizvođač:
   // Ako je magacin pun treba sačekati
   IF (ima_ih = n)
      pun.Wait;
   ENDIF
   stavi novi;
   ima_ih++;
   //Ako ima proizvoda šalje se signal
   IF (ima_ih > 0)
      prazan.Signal;
   ENDIF
```

Korišćenje monitora (2)

- Objašnjenje algoritma:
  - pun i prazan su uslovne promenljive koje služe za blokiranje u situacijama kada proizvođač pokušava da stavi novi proizvod u puno skladište, ili kada potrošač pokušava da uzme proizvod iz praznog skladišta.
  - Na početku rada funkcije Potrošač, ako je skladište puno (imah= n) onda se proces blokira (pun.Wait).
  - Ako skladište nije puno, novi proizvod se stavlja u skladište i uvećava brojač za broj proizvoda u skladištu (niko ga neće prekinuti, jer se to radi u okviru monitora).
  - Na kraju rada funkcije Potrošač se šalje signal prazan.signal da obavesti procese koji eventualno čekaju jer je skladište bilo prazno da se situacija promenila i da je sada bar jedan proizvod u njemu.

Korišćenje monitora (3)

Korišćenje monitora može se ilustrovati na problemu potrošača i proizvođača.

```
Potrošač:
   // Ako nema proizvoda treba sačekati
   IF (ima_ih = 0)
      prazan.Wait;
   ENDIF
   uzmi novi;
   ima_ih--;
   IF (ima_ih == n – 1)
      pun.signal;
   ENDIF
```

Korišćenje monitora (4)

- Objašnjenje algoritma:
  - funkcija Potrošač se blokira ukoliko je skladište prazno (imaih = 0) i čeka na signal proizvođača da je bar jedan proizvod smešten u skladište.
  - Ukoliko su proizvodi na raspolaganju, potrošač uzima jedan i ažurira stanje.
  - Potom, ako je bila situacijia da je Potrošač uzeo proizvod iz punog skladišta, onda se šalje signal proizvođaču (pun.Signal) da se oslobodilo jedno mesto – pa ukoliko ima spremnih proizvoda jedan može da se smesti na to upražnjeno mesto.

Zahvalnica

Najveći deo materijala iz ove prezentacije je preuzet iz knjige „Operativni sistemi“ autora dr Miroslava Marića i iz slajdova sa predavanja istog autora.

Hvala dr Mariću na datoj saglasnosti za korišćenje tih materijala.