**4. Konkurentnost: uzajamno isključivanje i sinhronizacija**

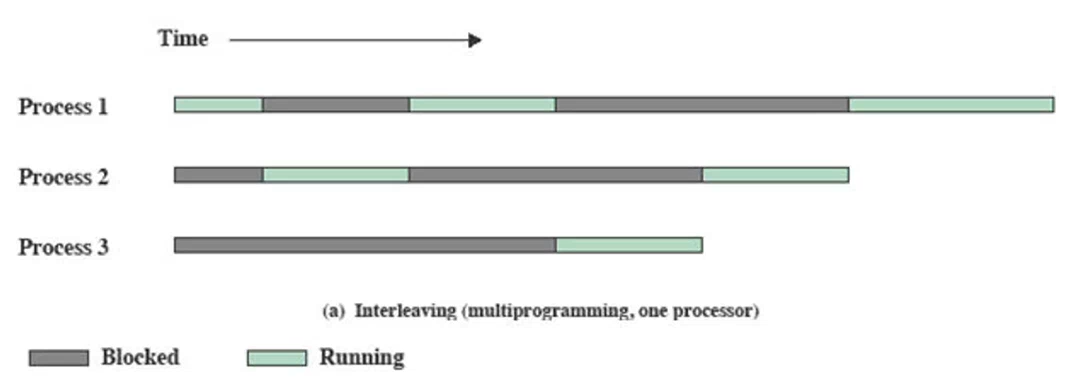
**OS i konkurentnost**

* Centralna uloga OS.a je upravljanje procesima i nitima
  + Multiprogramiranje – Upravljanje više procesa unutar jednoprocesorskog sistema
  + Multiprocesiranje – Upravljanje više procesa unutar multiprocesora
  + Distribuirano procesiranje – Upravljanje više procesa koji se izvršavaju na distribuiranim računarskim sistemima
* **Konkurentnost** predstavlja značajni element u dizajnu OS-a, u okviru:
  + Dodele procesorskog vremena procesima
  + Podele resursa i nadmetanje za resursima
  + Sinhronizacije izvršavanja više procesa
  + Komunikacije između procesa

**Kada se javlja konkurentnost?**

* Konkurentnost se javlja u tri različita konteksta
  + Višestruke aplikacije
    - Multiprogramiranje obezbeđuje deljenje vremena obrade između aktivnih aplikacija
  + Struktuirane aplikacije
    - Aplikacije mogu biti programirane kao skup konkurentnih procesa
  + Struktura operativnog sistema
    - Operativni sistem se implementira kao skup procesa i niti

**Principi konkurentnosti**

* Preplitanje izvršenja procesa na jednoprocesorksom sistemu

Samo jedan proces (zeleno) je u stanu izvršavanja u jednom trenutku

**Problemi sa konkurentnošću**

* Problemi potiču od osnovnih karakteristika multiprogramskih multiprocesorskih sistema:
  + Relativna brzina izvršavanja procesa se ne može predvideti
* Problemi su:
  + Deljenje globalnih resursa
  + Operativnom sistemu je teško da optimalno upravlja deodelom resursa
  + Teško je otkrivanje grešaka u programiranju
* Rešenje za konkurentnost treba da obuhvati:
  + Komunikaciju među procesima
  + Deljenje resursa
  + Sinhronizaciju više procesa
  + Alokaciju vremena procesa za korišćenje resursa

**Ključni pojmovi koji se odnose na konkurentnost**

* **Kritična sekcija (critical section)**
  + Deo koda unutar procesa koji zahteva pristup deljenim resursima i ne može se izvršiti dok je drugi proces u odgovarajućem delu koda
* **Uzajamno blokiranje, zastoj, samrtni zagrljaj (deadlock)**
  + Situacija u kojoj dva ili više procesa ne mogu da nastave sa radom, jer svaki odgovor na promene drugih procesa i pri tom ne rade ništa korisno
* **Zaključavanje uživo (livelock)**
  + Situacija u kojoj dva ili više procesa konstantno menjaju svoje stanje kao odgovor na proemene drugih procesa i pri tom ne rade ništa korisno
* **Uzajamno isključivanje (mutual exclusion)**
  + Zahtev po kome nijedan proces ne može biti u kritičnoj sekciji, koja pristupa bilo kom deljenom resursu, kada je jedan proces u kritičnoj sekciji pristupio deljenim resursima
* **Uslov trke (race condition, haotično stanje)**
  + Situacija u kojoj više niti ili procesa čita i upisuje stavke deljenih podataka, a krajnji rezultat zavisi od relativnog vremena izvršenja.
* **Gladovanje (starvation)**
  + Situacija u kojoj dispečer neprekidno preskače proces koji je spreman na izvršavanje, iako je spreman ta rad, taj proces nikada neće biti odabran.

**Jednostavan primer konkurentnosti**

* Posmatraćemo jednoprocesorski multiprogramski sistem koji podržava jednog korisnika

void echo()

{

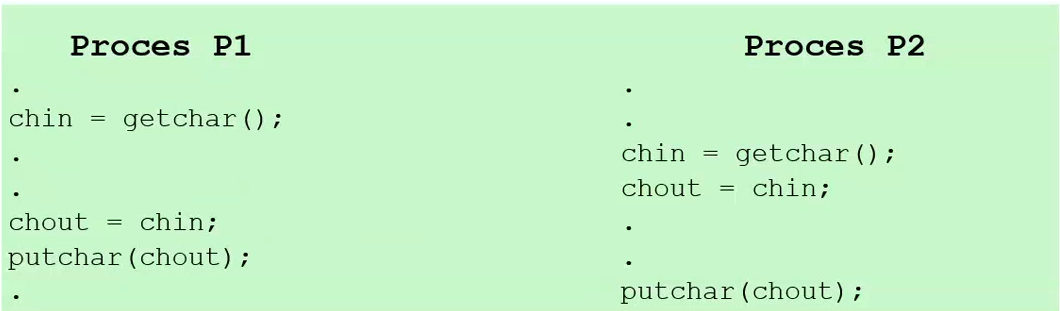
chin = getchar();

chout = chin;

putchar(chout);

}

* Svaki proces može ponavljati poziv ove deljene procedure kako bi prihvatio ulaz sa tastature i prikazivao ga na ekranu korisnika
  + Jedan primerak ove procedure se učitava u glavnu memoriju i zajednički je za sve aplikacije
  + Promena chin je globalna.
* Korisnik se može kretati između aviše aplikacija, ali su tastatura i ekran zajednički za sve aplikacije
  + Svaka aplikacija poziva jedanput, ili više puta ovu proceduru.



**Problem:**

1. Proces P1 poziva proceduru **echo** i biva prekinut nakon prve instrukcije
2. Aktivira se proces P2 i poziva proceduru **echo** koja se izvršava do kraja
3. Nakon procesa P2, nastavlja se izvršavanje procesa P1
   * **Rezultat**: prvi znak je izgubljen, a drugi se prikazuje 2 puta
   * **Srž problema** je deljena globalna promenljiva **chin**

**Rešenje:** Postaviti ograničenje da samo jedan proces u jednom trenutku može biti u proceduri **echo**.

1. Proces P1 poziva **echo** i biva prekinut nakon prve instrukcije (nakon unosa znaka)
2. Aktivira se proces P2 i poziva **echo**, ali kako je proces P1 u proceduri **echo**, proces P2 se blokira
3. Kada P1 nastavi sa radom, on nalazi svoj znak u **chin** i prikazuje ga na ekranu. Kada P1 napusti proceduru **echo**, uklanja se blokada sa P2 i on ulazi u proceduru **echo**
   * **Rezultat:** oba znaka su uspešno prikazana jedanput.

**Uslov trke**

* Uslov trke (race condition) nastaje kada:
  + Više procesa ili niti čitaju ili upisuju deljene podatke
  + Oni to obavljaju na način da finalni rezultat zavisi od redosleda izvršenja procesa.
* Konačna vrednost deljenih podataka zavisi od toga koji proces završi “trku” poslednji.
* Primer:
  + Globalne promenljive: b = 1, c = 2
  + P3: b = b + c
  + P4: c = b + c
  + Ako je P3 prvi, rezultat je b = 3, c = 5
  + Ako je P4 prvi, rezultat je b = 4, c = 3

**Long ass example that demonstrates this problem in detail**

* If you know P3 & P4 are done through 3 instructions lw, add, sw, and that there is a timeout interrupt that can occur while the process is executing, then shit can get messy, so lock your global variables!

**Poslovi operativnog sistema u vezi konkurentnosti**

* OS mora biti sposoban da prati razne procese
* OS mora dodeljivati i oduzimati razne resurse za svaki aktivan proces
  + Procesorsko vreme
  + Memoriju
  + Datoteke
  + U/I uređaje
* OS mora zaštititi podatke i fizičke resurse jednog procesa od nenamernog narušavanja od drugog
* Izlaz mora biti nezavistan od brzine izvršenja drugih konkurentnih procesa.

**Uzajamno delovanje procesa**

* **Procesi koji nisu svesni postojaja drugih procesa**
  + Nezavisni procesi za koje nije predviđeno da rade zajedno,
  + OS vodi računa o njihovom nadmetanju za resurse.
* **Procesi koji su indirektno svesni postajnja drugih procesa**
  + Procesi koji dele pristup nekom objektu (npr. U/I buffer-u),
  + Pri deljenju zajedničkog objekta pokazuju kooperaciju preko deljenja.
* **Procesi koji su direktno svesni postojanja drugih procesa**
  + Procesi koji mogu komunicirati preko PID-a i rade zajednički neku aktivnost,
  + Takvi procesi pokazuju kooperaciu putem komunikacije.

|  |  |  |  |
| --- | --- | --- | --- |
| **Steoen svesnosti** | **Odnos** | **Uticaj koji jedan proces ima na drugi** | **Potencijalni problemi upravljanja** |
| Procesi nisu svesni postojanja drugih procesa | Nadmetanje | * Izvršavanje jednog procesa nezavisno od akcije ostalih procesa * Može uticati na vreme izvršenja procesa * Rezultat itekako zavisi od drugih procesa you dumb fuck | * Uzajamno isključivanje * Uzajamno blokiranje (resurs koji se može obnoviti) * Gladovanje |
| Procesi su indirektno svesni postojanja drugih procesa | Kooperacija deljenjem | * Akcije jednog procesa mogu zavisiti od informacija dobijenih od drugih procesa * Može uticati na vreme izvršavanja procesa | * Uzajamno isključivanje * Uzajamno blokiranje (resurs koji se može obnoviti) * Gladovanje * Povezanost podataka |
| Procesi su direktno svesni postojanja drugih procesa | Kooperacija komunikacijom | * Akcije jednog procesa mogu zavisiti od informacija dobijenih od drugih procesa * Može uticati na vreme izvršavanja procesa | * Uzajamno blokiranje (resurs koji se može obnoviti) * Gladovanje |

**Nadmetanje procesa za rasursima**

* Konkurentni procesi se nadmeću za korišćenje istog resursa
* Pri nadmetanju procesa postoje tri problema upravljanja:

1. **Uzajamno isključivanje** pri korišćenju nedeljivog resursa
   * Takav resurs nazivamo **kritični resurs,** a deo programa koji ga koristi **kritična sekcija**.
   * Kritična sekcija:
     + Samo jedna program u datom trenutku može biti u kritičnoj sekciji
     + Primer: samo jedna proces u jednom trenutku može slati komandu štampaj
2. **Uzajamno blokiranje**
3. **Gladovanje**

**Ilustracija uzajamnog isključivanja**

* N procesa pristupa deljenom resursu u okviru svoje kritične sekcije
  + Funkcije zaključavanja/otključavanja (entercritical/exitcritical)

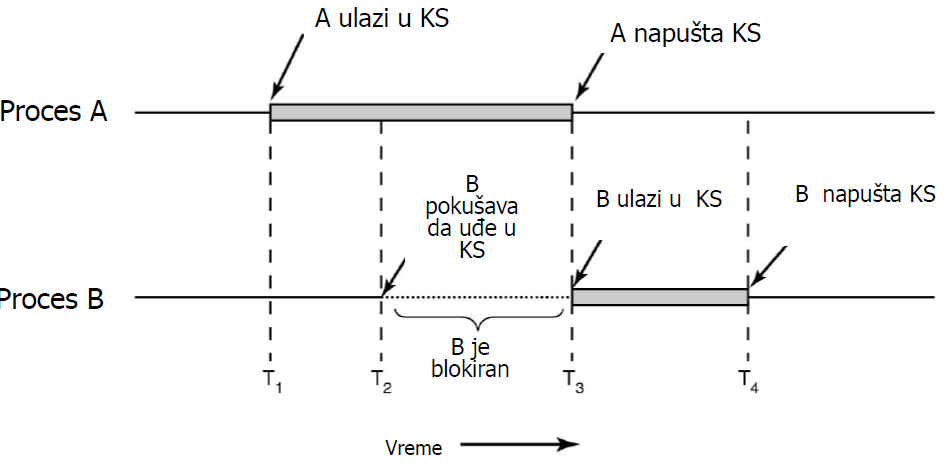
**Kooperacija procesa deljenjem i komunikacijom**

* Konkurentni procesi (ili niti) često imaju potrebe da dele podatke (koji se održavaju u deljenoj memoriji, ili u datotekama) i resurse, ili razmenjuju poruke
* Ako ne postoji kontrolisani pristup deljenim resursima/podacima, tj. ukoliko nema uzajamnog isključivanja, neki procesi će dobiti nekonzistentan pogled na podatke
* Kod kooperacije komunikacijom, nema uzajamnog isključivanja već je moguće uzajamno blokiranje i gladovanje
* Akcije koje izvode konkurentni procesi zavisiće od redosleda kojim se izvršavanja prepliću/preklapaju – uslov trke
* Zbog toga procese treba sinhronizovati

**Zahtevi za uzajamno isključivanje**

* Samo jednom procesu je dozvoljeno da uđe u kritičnu sekciju (KS) radi pristupa deljenom resursu
  + Dva i više procesa ne mogu biti istovremeno u kritičnoj sekciji
* Proces koji se zaustavi u sekciji koja nije kritična mora to uraditi bez uticaja na druge procese
* Proces koji zahteva ulazak u KS ne može biti beskonačno zadržan – **nema zastoja i gladovanja**
* Kada nijedan proces niju u KS, bilo kom procesu koji zahteva ulaz u KS to mora biti dozvoljeno
* Ne prave se pretpostavke o relativnoj brzini procesa niti o broju procesa
* Proces ostaje u kritičnoj sekciji samo konačno vreme

**Uzajamno isključivanje u izvršavanju kritične sekcije**



**Rešenje uzajamnog isključivanja**

* SW rešenja (za samostalni rad, DODATAK A [HAHAHAHAHA])
  + Softverski algoritmi imaju značajno dodatno procsiranje i rizik od nastanka logičkih grešaka
* HW rešenja
  + Vezana za neke mašinske instrukcije i prevenciju prekida
* OS rešenja
  + OS obezbeđuje sistemske funkcije i strukture podataka koje programeri mogu koristiti za programiranje kritičnih sekcija

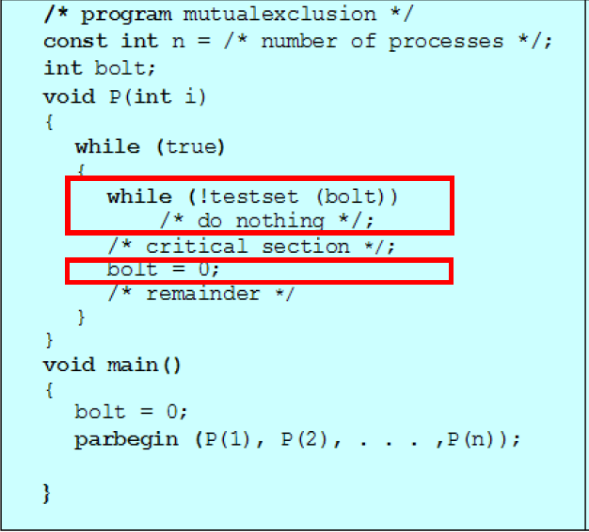
**Uzajamno isključivanje: HARDWARE-ska podrška**

* Postoje dva pristupa:
  + Zabrana (onemogućavanje, disable) prekida pri izvršavanju naredbi
  + Specijalne mašinske instrukcije

**Zabrana prekida**

* Na jednoprocesorskom sistemu:
  + Obezbeđeno je uzajamno isključivanje, ali efikasnost izvršenja degradirana.
  + Dok je proces u KS, nijedan drugi proces se ne može izvršavati
* Na višeprocesorksom sistemu ovaj pristup ne radi:
  + Uzajamno isključivanje nije obezbeđeno (ne možemo da isključimo procese na svim jezgrima)
* Generalno, ovo rešenje nije prihvatljivo!

**Specijalne mašinske instrukcije**

* Na hardverskom nivou, pristup nekoj memorsijskoj lokaciji isključuje ostale pristupe toj lokaciji
* Projektanti HW su predložili mašinske instrukije koje nad istom memorijskom lokacijom izvode dve akcije **atomično** (nedeljivo, u jednom koraku koji ne može biti prekinut)
  + Na primer: čitanje i upis, ili čitanje i testiranje
* Izvršavanje takve instrukcije je uzajamno isključivo (čak i kod multiprocesora)
  + Tokom izvršenja instrukcije, pristup memorijskoj lokaciji je blokiran za sve druge instrukcije koje referencijraju tu lokaciju
* Implementirane instrukcije:
  + Test and Set instrukcije
  + Compare&Swap instrukcija
  + Echange instrukcija (XCHG na intel-u)

**Test and Set instrukcija**

* Instrukcija za testiranje i postavljanje (Test and Set instruction)

**boolean testset(int i){**

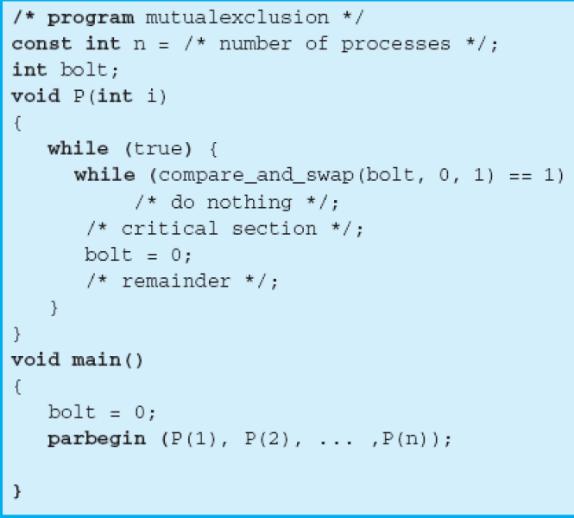
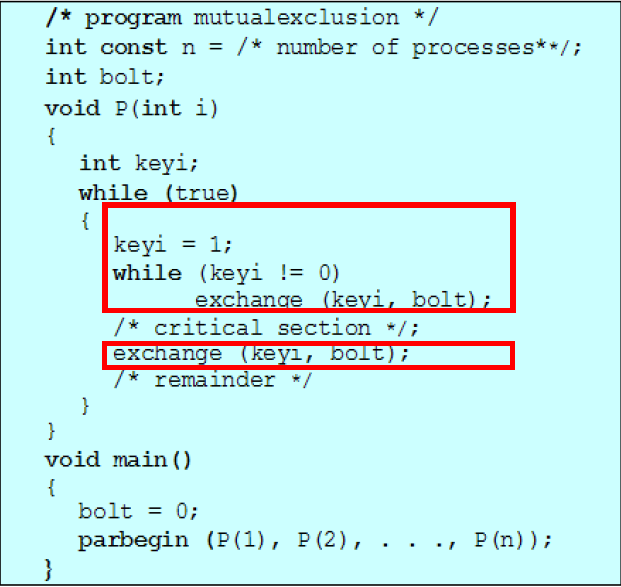
**If (i == 0)**

**{ i = 1; return true; }**

**else**

**return false;**

**}**



**Exchange instrukcija**

* Instrukcija razmene (Exchange instruction)

**void exchange(int register, int memory){**

**int temp;**

**temp = memory;**

**memory = register;**

**register = temp;**

**}**

* Instrukcija XCHG na procesorima INtel-32 (Pentium) i IA-64 (Itanium)

**Compare & Switch instrukcija**

**int compare\_and\_switch(int \*word, int testval, int newval) {**

**int oldval;**

**oldval = \*word;**

**if(oldval == testval)**

**\*word = newval;**

**return oldval;**

**}**

**Svojstva pristupa sa mašinskim instrukcijama**

* Prednosti :
  + Može se primenjivati na bilo koji broj procesa i na jednoprocesorskoj i multiprocesorskoj arhitekturi, gde procesori dele glavnu memoriju
  + Jednostavno je i lako se proverava
  + Može se koristiti za podršku više kritičnih sekcija ; Svaka KS može se definisati svojom promenljivom.
* Nedostaci :
  + Zaposleno čekanje
  + Moguće je gladovanje
    - Kada jedan proces napusti kS, a više čeja na ulaz u KS, izbor sledećeg procesa je proizvoljan
  + Moguće je uzajamno blokiranje
    - Ako je proces nižeg prioriteta P1 u KS-u, prekinut je i procesor je dodeljen procesu višeg prioriteta P2 koji želi da uđe u KS
      * P1 se prekida, P2 dobija procesor
      * Ako P2 želi da koristi isti resurs kao P1, neće mu biti dozvoljeno
      * P2 čekai drži procesor, tako da P1 neće nikad biti raspoređen na procesor, jer je nižeg prioriteta od P2

**Semafori**

* Definisani od strane E. Dijkstra, 1965
* Sinhronizacioni mehanizam koji obezbeđuje OS
* Semafor S je **integer promenljiva** nad kojom su definisane tri **atomične operacije**
  + **INICIJALIZACIJA**
  + **semWait(S)**
    - Alternativna imena **P** (Dijkstra, na holandskom proberen), down (Tenenbaum), wait
  + **semSignal(S)**
    - alternativna imena **V** (Dijkstra, na holandskom verhogen), up (Tanenbaum), signal
* Semafor može biti:
  + **Binarni** – uzima samo dve vrednosti 0 i 1
  + **Brojački (generalni)** – uzima vrednosti 0, 1, 2, ..., n

**Operacije semafora – definicija**

* Proces koji izvede operaciju ***semWait(S)* blokira sam sebe**, umesto da čeka u petlji
* Proces koji je blokiran na semaforu **biće probuđen** kada neki drugi proces izvede operaciju ***semSignal(S)***
* Ove dve operacije moraju biti:
  + **Atomične** – moraju se izvoditi u celini, tj. Bez prekidanja
  + **Uzajamno isključive** – na istom semaforu ne mogu se istovremeno izvoditi...

**semSignal(S):**

**if** (jedan ili više procesa čeka na S) **then**

probudi jedan od procesa koji su blokirani na semaforu

**else**

S = S + 1

**semWait(S):**

**if** (S>0) **then**

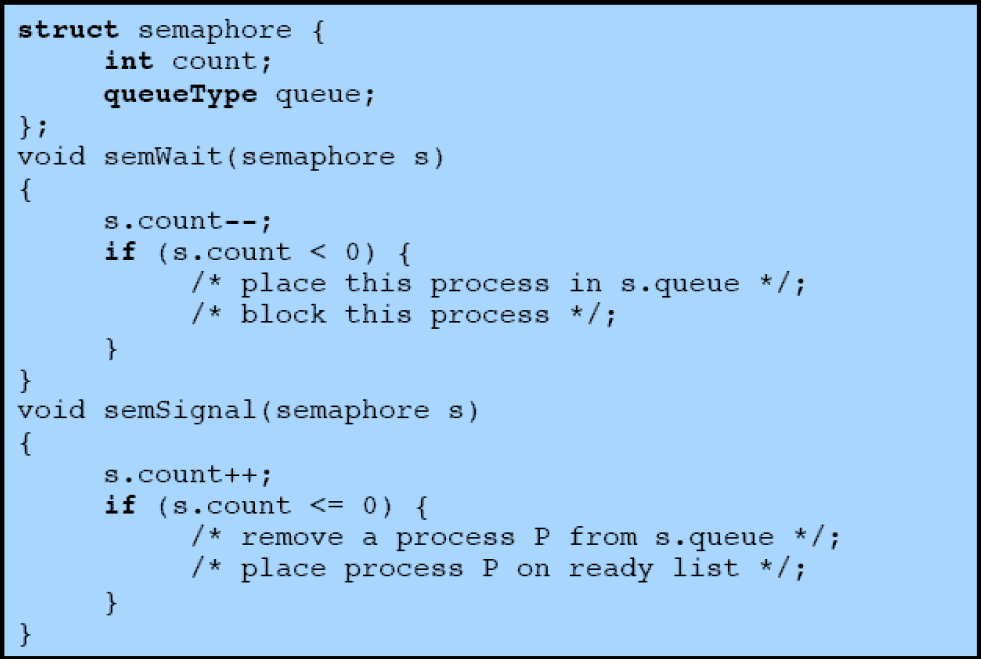
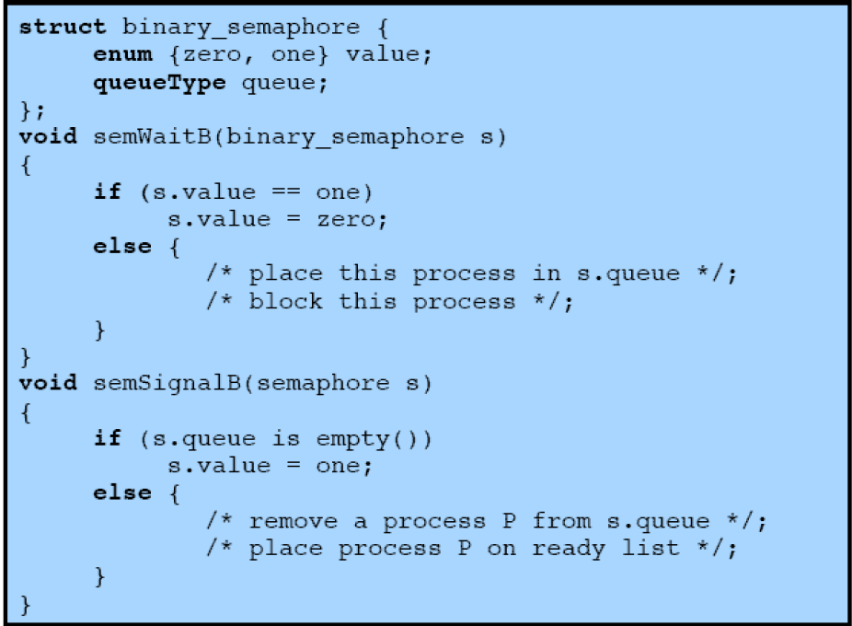
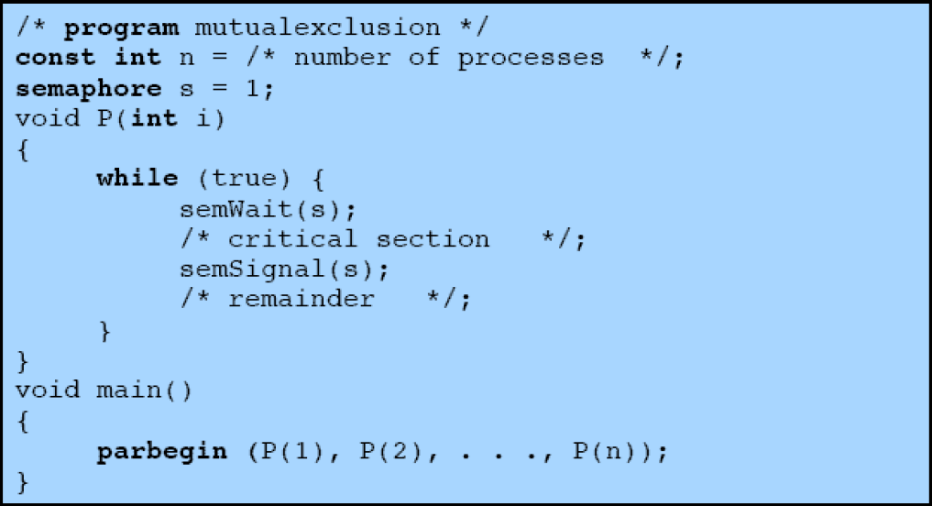
S = S – 1

**else**

{ blokiraj\_proces, aktiviraj\_dispečer }

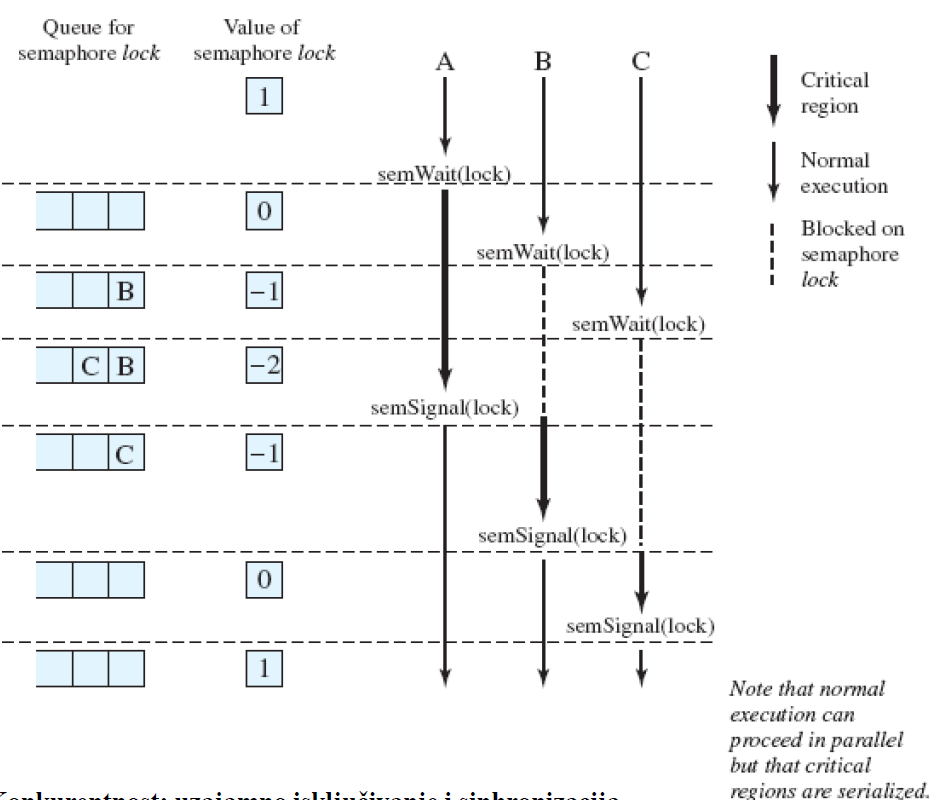
**Semafor - implementacija**

**Operacije binarnog semafora**

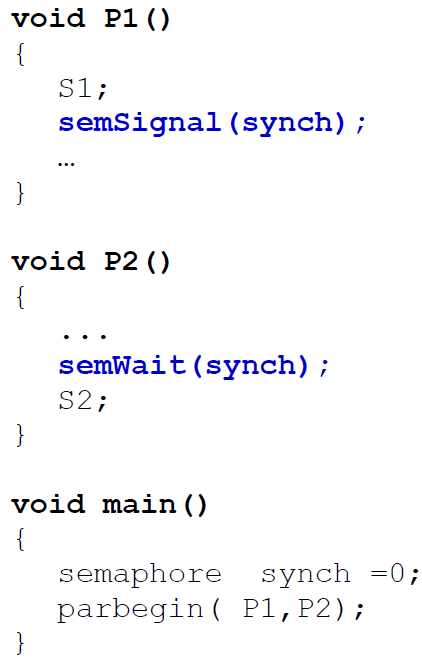
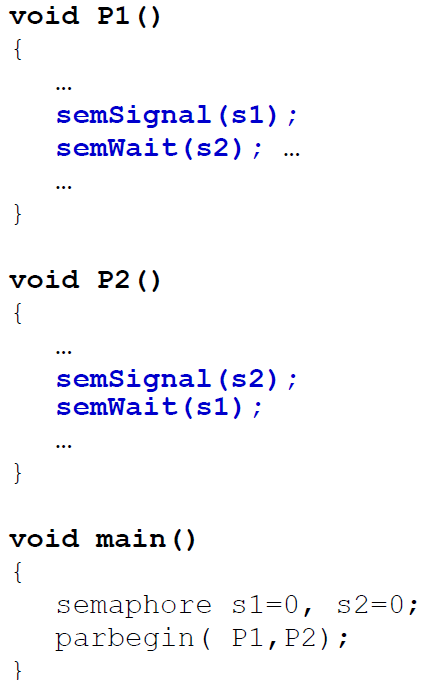
**Primer : Uzajamno isključivanje**

**Uzajamno isključivanje korišćenjem semafora**

* Tri procesa pristupaju deljenim podacima zaštićenim semaforom
* **S.count** >= 0 – broj procesa koji može da izvrši operaciju semWait(S), bez blokiranja (ukoliko u međuvremenu nije izvršena semSignal(S))
* **S.count** < 0 – ABS(S.count) - broj procesa blokiranih u redu **s.queue**



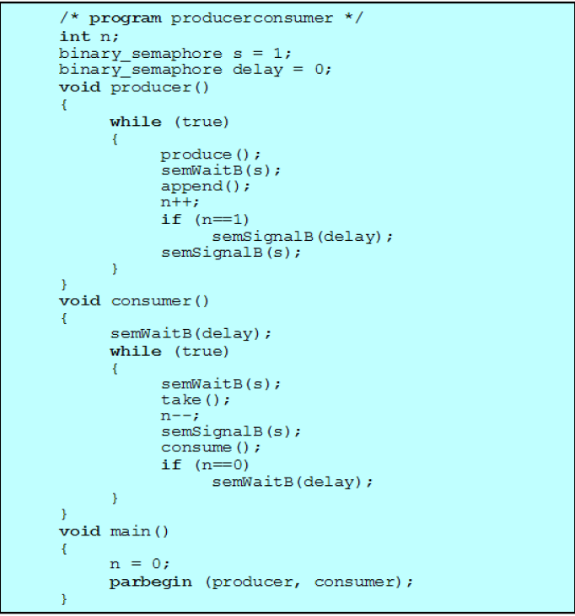
**Korišćenje semafora za sinhronizaciju dva procesa**

* Semafori se mogu koristiti za **sinhronizaciju dva procesa**
* Posmatramo dva procesa P1 i P2
* Naredba S1 procesa P1 treba da se izvede pre naredbe S2 procesa P2
* Koristi se **binarni semafor sync**
* Semafor **sync** se inicijalizuje na 0
* Semafori se mogu koristiti za sinhronizaciju procesa tipa **rendezvous**
* Kod tzv. Sastanka (rendezvous) proces koji prvi dođe do KS mora čekati da drugi proces stigne do svoje KS
* Koriste se **dva binarna semafora S1 i S2**, koji se inicijalizuju na 0

**Porblem proizvođač/potrošač (Producer/Consumer)**

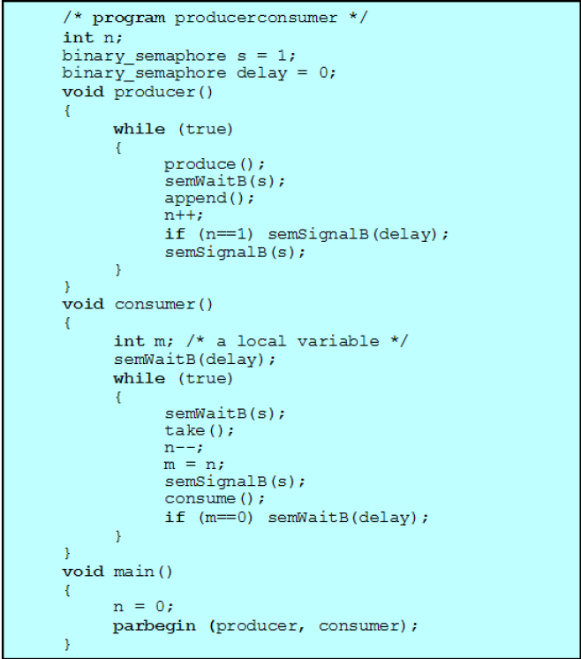
* Jedan ili više proizvođača generiše podatke (slogove, znakove) i stavlja ih u buffer
* Jedan potrošač uzima podatke iz buffer-a jedan po jedan
* Sistem mora obezbediti da se operacije na buffer-om ne preklapaju
* U datom trenutku samo jedan proizvođač ili potrošač može pristupati buffer-u
* Pretpostavimo da je buffer neograničen i sadrži linearan niz elemenata

**Rešenje korišćenjem binarnih semafora**



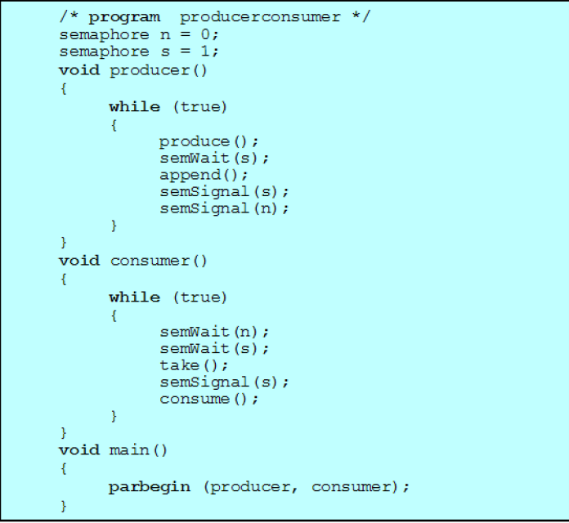
in može da se promeni, jer nije u kritičnoj sekciji

* **Neispravno rešenje** problema proizvođač/potrošač sa beskonačnim baferom uz upotrebu binarnih semafora
* Broj elemenata u baferu N = in – out
* Semafor S obezbeđuje uzajamno isključivanje
* Semafor delay primorava potrošača da se blokira (izvršavanjem semWait) ukoliko je buffer prazan

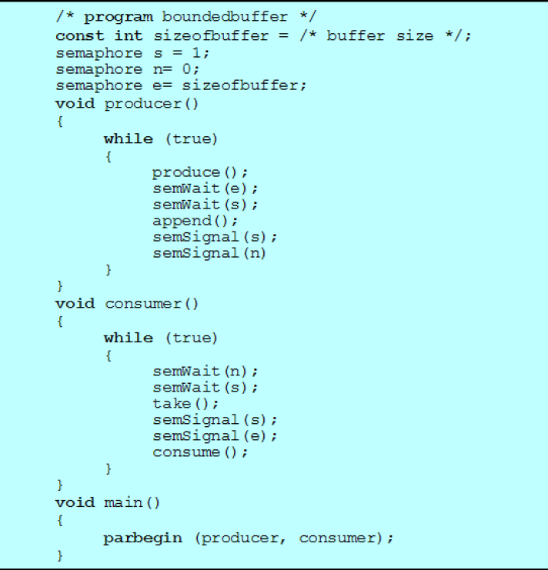


**Korigovano rešenje**

* **Korektno rešenje** problema proizvođač/potrošač sa beskonačnim baferom uz upotrebu binarnih semafora
* Dodavanje **pomoćne promenljive m u okviru kritične sekcije**

**Rešenje korišćenjem brojačkog semafora**

* Konkretno rešenje problema proizvođač/potrošač sa beskonačnim baferom uz upotrebu brojačkog semafora
* Promenljiva n je sada semafor i predstavlja broj elemenata u baferu
* Problem nastaje ako se zamene **semWait(n)** i **semWait(s)** -> smrtni zagrljaj

**Problem: proizvođač/potrošač sa ograničenim (kružnim) bufferom**

* Korektno rešenje problema proizvođač/potrošač sa ograničenim kružnim buffer-om uz upotrebu binarnog i dva brojačka semafora
  + Proizvođač blokira kad hoće da upiše element u pun buffer
  + Potrošač se blokira kad hoće da uzme element iz praznog buffer-a

**Problemi sa semaforima**

* Semafori nude moćan alat za obezbeđenje uzajamnog isključivanja i za koordinaciju procesa
* Ali **semWait(S)** i **semSignal(S)** su razbacani među procesima, pa je teško razumeti njihove efekte
* Korišćenje mora biti korektno u svim procesima
* Jedan loš proces može uticati na zastoj čitave kolekcije procesa

**Monitor**

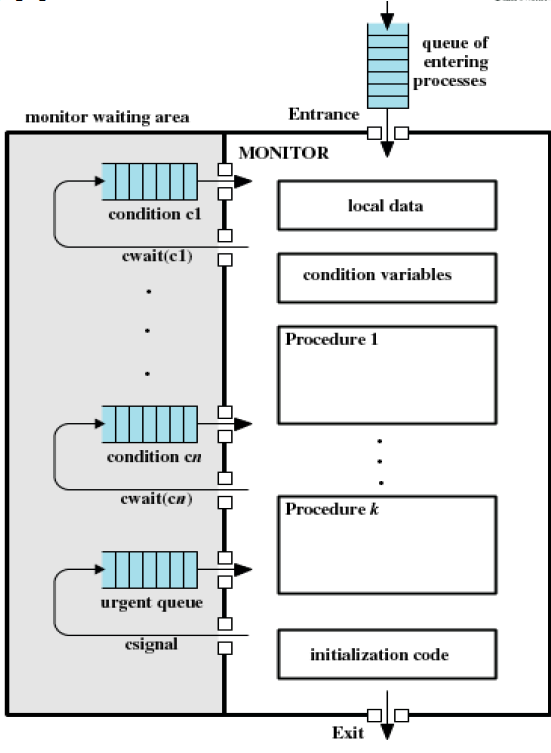
* Monitor je konstrukcija viših programskih jezika koja nudi funkcionalnost ekvivalentnu semaforima, ali je lakša za upravljanje
* Prvi put ga je formalno definisao Hoare, 1974
* Nalazi se u mnogim programskim jezicima
  + Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3, Java...
  + Takođe se može implementirati kao programska biblioteka
* Monitor je SW modul, koji sadrži :
  + **Jednu ili više procesura/funkcija**
  + **Inicijalizacionu sekvencu**
  + **Lokalne promenljive**
* Karakteristike monitora:

1. **Lokalnim promenljivama može se pristupati samo iz procedura monitora i nijednom spoljnom procedurom**
2. **Proces “ulazi” u monitor pozivom neke od njegovih procedura**
3. **U jednom trenutku samo jedan proces se može izvršavati u monitoru; svaki drugi proces koji poziva proceduru monitora se blokira i čeka da monitor postane dostupan**

* Prve dve karakteristike podsećaju na karakteristike objekata u OO jezicima
  + Monitor se može implementirati kao objekat
* Treća karakteristika obezbeđuje uzajamno isključivanje
* Monitor osigurava uzajamno isključivanje
  + Nije potrebno programirati ovo ograničenje
* U jednom trenutku deljivim podacima u monitoru može pristupiti samo jedan proces
  + Stoga su deljivi podaci zaštićeni njihovim smeštanjem u monitor
* Monitor takođe poseduje sinhronizacioni alat
  + To su **uslovne promenljive (condition)**
  + Sinhronizaciju procesa vrši programer korišćenjem uslovnih promenljivih

**Uslovne promenljive monitora**

* Uslovne promenljive su lokalne u monitoru
  + Može im se pristupati jedino unutar monitora
* Uslovne promenljive su specijalnog tipa podataka u monitoru kojima se može pristupati jedino pomoću dve funkcije:
  + **cwait(A):** blokira izvršavanje procesa koji je pozvao ovu funckiju na uslovnoj promenljivoj **A**; monitor je sada na raspolaganju za bilo koji drugi proces
  + **csignal(A):** budi proces koji je blokiran na uslovnoj promenljivoj **A**
    - Ako postoji više takvih procesa – bira jedan
    - Ako nema takvih procesa – ne radi ništa
  + Treba uočiti da su operacije cwait i csignal monitora različite od istih operacija kod semafora
    - Ako proces u monitoru izda signal i ako nijedan proces ne čeka na toj uslovnoj promenljivoj, signal je izgubljen

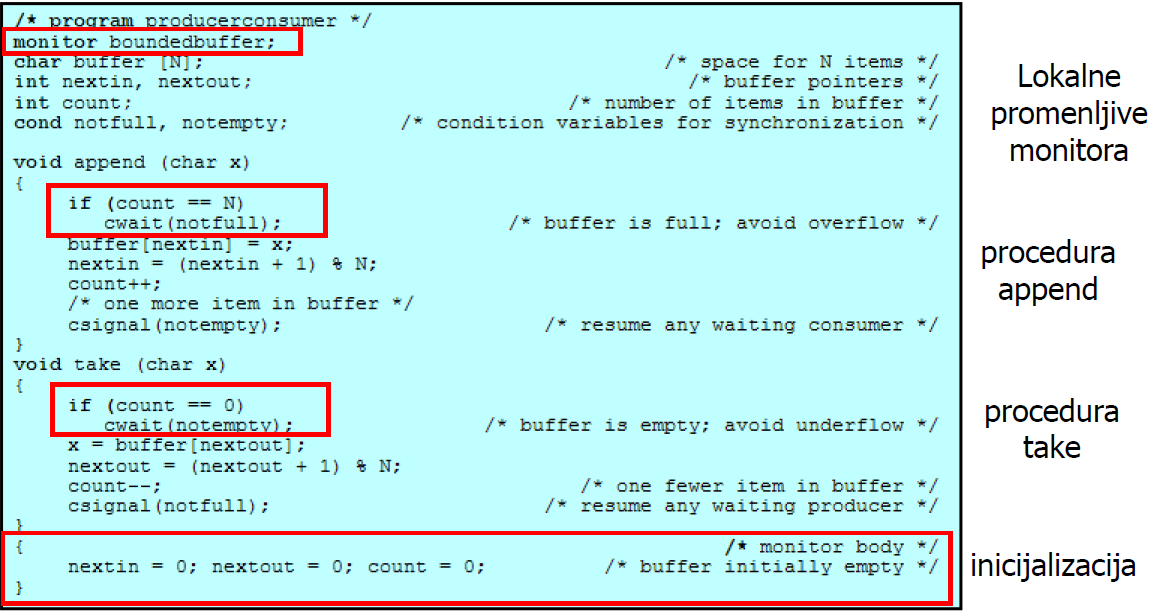
**Struktura monitora**

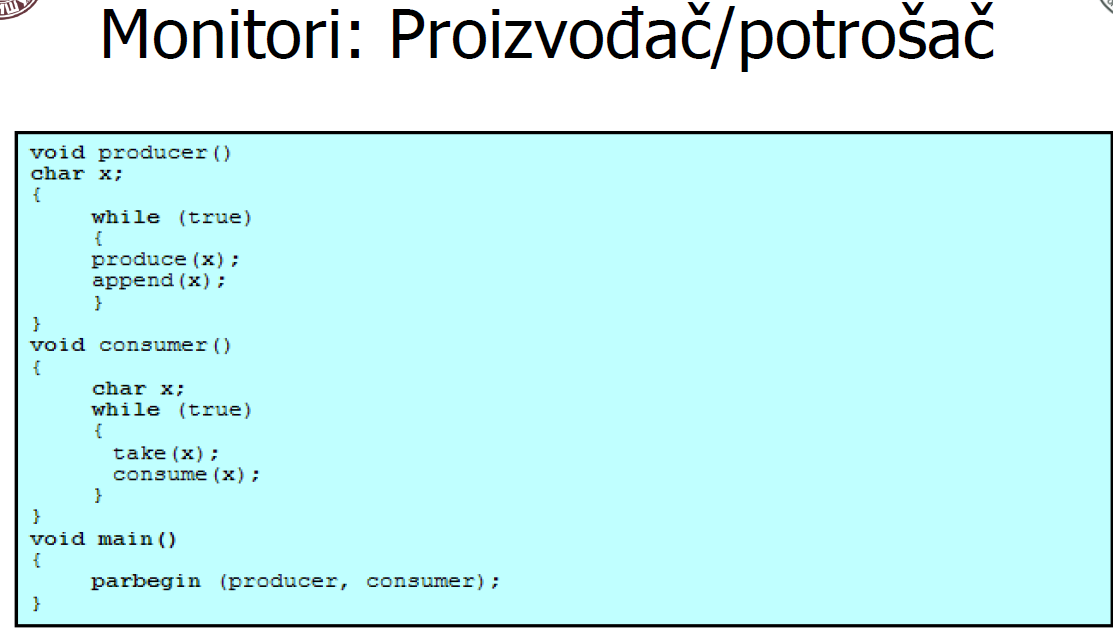
* Monitor ima jednu ulaznu i jednu izlaznu tačku
* Proces koji je u monitoru može privremeno sam sebe blokirati na uslovnoj promenljivoj **A**, korišćenjem operacije **cwait(A)**
* On se tada stavlja u red procesa koji pokušavaju da ponovo uđu u monitor kada se uslov promeni i tada nastavljaju saizvršavanjem sa naredbom koja sledi iza **cwait(A)**
* Ako proces koji se izvršava u monitoru detektuje ispunjenje uslova na kojem su blokirani procesi, poziva **csignal(A)** koja budi (deblokira) proes iz reda procesa te uslovne promenljive **A**

**Primena monitora za rešavanje problema proizvođač-potrošač**

* Monitor **boundedbuffer** upravlja buffer-om, koji se koristi za smeštanje i preuzimanje znakova
* Monitor sadži dve uslovne promenljive **notfull** (kada postoji mesto u buffer-u da se doda barem jedan znak) i **notempty** (kada u buffer-u postoji barem jedan znak)
* Proizvođač može dodati podatke u buffer samo preko procedure monitora **append()**
  + Proizvođač nema direktan pristup buffer-u
* Potrošač može uzeti znak iz buffer-a samo preko procedure monitora **take()**
  + Potrošač nema direktan pristup buffer-u

**Monitori: Proizvođač/potrošač**





**Prenos poruka (Message Passing)**

* To je metoda komunikacije među procesima
* Koriste se dve primitive:
  + **Send(odredište, poruka)**
  + **Receive(izvor, poruka)**
* **Send()** – proces šalje poruku drugom procesu koji je označen kao odredište
* **Receive()** – proces prima poruku od procesa koji je označen kao izvor
* Problemi u prenosu poruka:
  + Format poruke
  + Sinhronizacija – procese koji komuniciraju treba na neki način sinhronizovati
  + Protokol za komunikaciju
    - Primalac i pošiljalac moraju dogovoriti neki protorokol za komunikaciju
    - Npr. primalac po prijemu poruke šalje potvrdu, a pošiljalac ako potvrda ne stigne za neko vreme ponovo šalje poruku
    - Poruke se numerišu, čime se izbegava zabuna ako poruka kasni
  + Imenovanje (adresiranje) procesa
    - Procesi koji komuniciraju moraju imati način da se međusobno referenciraju
  + Autentikacija – da pošiljalac i primalac znaju da komuniciraju sa pravim procesom

|  |  |
| --- | --- |
| Opšti format poruke promenljive dužine | |
| Zaglavlje poruke | Tip poruke |
| ID odredišta |
| ID izvora |
| Dužina poruke |
| Upravljačke informacije |
| Telo  poruke | Sadržaj poruke |

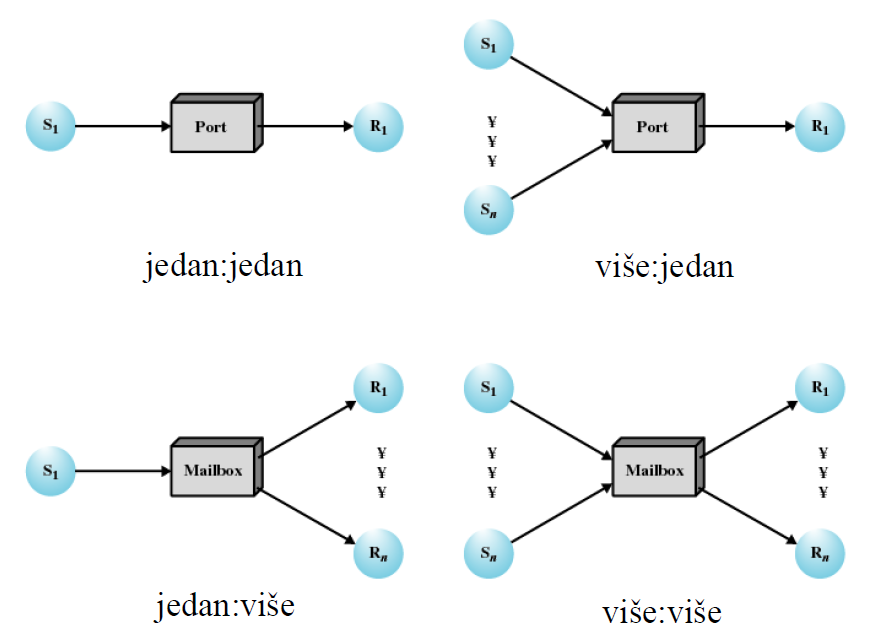
**Format poruke**

* Format poruke zavisi:
  + Od cilja sistema za prenos poruka
  + Od toga da li se sistem izvršava na jednom računaru ili na distribuiranom sistemu
* Poruke mogu biti fiksirane i promenljive družine

**Sinhronizacija u prenosu poruka (1)**

* Komunikacija dva procesa porukama podrazumeva neki nivo sinhronizacije između njih
  + Primalac ne može primiti poruku pre nego što je neki proces ne pošalje
* Važno je da se definiše šta se događa sa procesom nakon što pokrene send ili receive primitivu
* Proces primalac i proces pošiljalac mogu biti
  + **Blokirani**
  + **Neblokirani**
* Moguće su tri kombinacije :
  + **Blokirajući send, blokirajući receive**
    - Pošiljalac i primalac se blokiraju dok poruka ne bude isporučena
    - Naziva se rendezvous
  + **Neblokirajući send, blokirajući receive**
    - Pošiljalac nastavlja da se izvršava, a primalac se blokira dk ne stigne poruka
    - Omogućava se pošiljaocu da pošalje 1 ili više poruka na jednu ili više destinacija
    - Primeri su serverski procesi
  + **Neblokirajući send, neblokirajući receive**
    - Ni pošiljalac, ni primalac ne moraju da budu blokirani
* **Za pošiljaoca : prirodno je da ne bude blokiran posle slanja poruke pozivom funkcije send**
  + Može poslati nekoliko poruka na više destinacija
  + ALi pošiljalac obično očekuje potvrdu prijema poruke (za slučaj da je primalac neispravan)
* **Za primaoca : prirodnije je da bude blokiran, dok ne primi poruku pozivom funkcije receive**
  + Primalac obično treba neku informaciju pre nego što nastavi obradu
  + Postoji opasnost da bude neograničeno blokiran ako proces pošiljalac završi pre nego što pošalje send

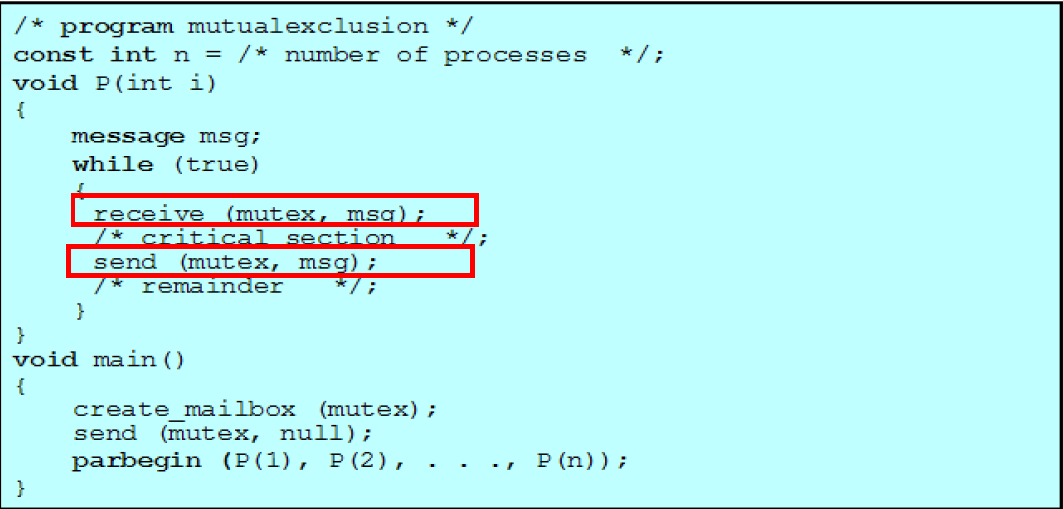
**Adresiranje procesa**

* **Direktno adresiranje:**
  + Jedan proces direktno šalje poruku drugom procesu
  + Pošiljalac eksplicitno specificira kome šalje poruku preko argumenta **odredište**
  + Primalac eksplicitno specificira od koga prima poruku preko argumenta **izvor**
* **Indirektno adresiranje**
  + Poruke se ne šalju direktno od pošiljaoca primaocu, već se šalju deljenim strukturama podataka koje se sastoje od redova koji mogu privremeno da čuvaju poruke
  + Ovi redovi nazivaju se **poštaniski sandučići (mailboxes)**
  + Proces pošiljalac šalje poruku u određeni mailbox, a drugi proces primalac uzima poruku iz tog mailbox-a
* Kod indirektnog adresiranja odnos između pošiljaoca i primaoca može biti :
  + **1 :1 (jedan-prema-jedan)**
    - Privatni komunikacioni link između 2 procesa
  + **N:1 (više-prema-jedan)**
    - Koristan za klijent-server interakciju kada jedan proces prima poruke od većeg broja drugih procesa
    - U ovom slučaju se mailbox naziva **port**
  + **1 :N (jedan-prema-više)**
    - Postoji 1 pošiljalac i više primaoca
    - Pogoda je u slučaju kada jedan proces emituje prouke većem broju procesa
  + **N:M (više-prema-više)**
    - Dopušta da više serverskih procesa konkuretntno nudi servis većem broju klijenata.

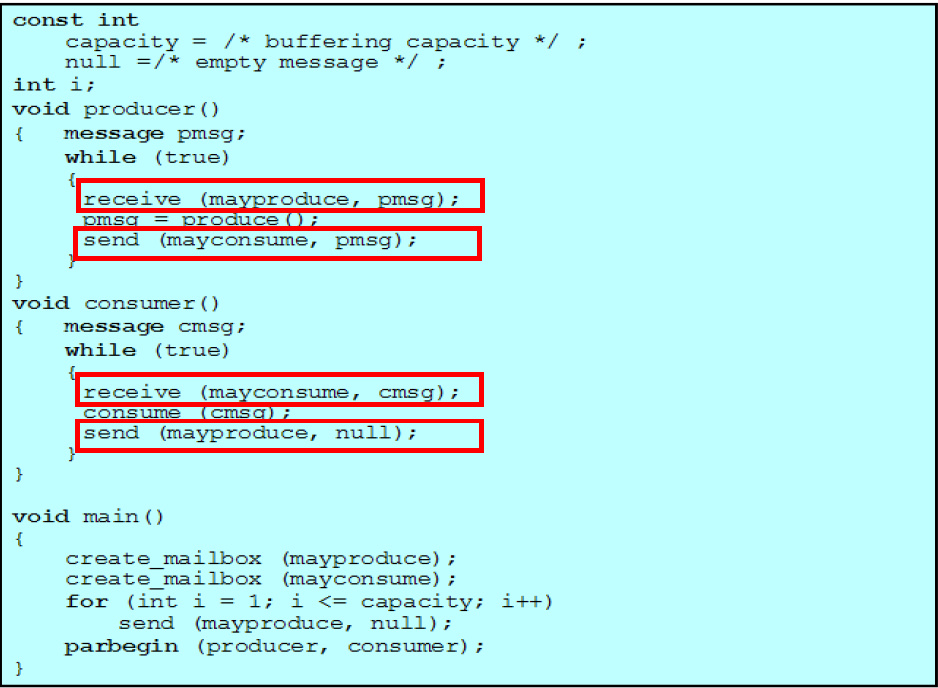
**Poštansko sanduče i port**

* Poštansko sanduče (mailbox) može biti
  + Privatno za par pošiljalac-primalac
  + Isto poštansko sanduče može deliti više pošiljaoca i primaoca
    - OS mora obezbediti tip podatka – message
* **Port je poštansko sanduče povezano sa 1 primaocem i više pošiljaoca**
  + Koristi se za klijent-server aplikacije
* Vlasnici portova i mailbox-ova
  + **Port obično kreira proces primalac i on je njegov vlasnik**
  + **Port se uništava kad se proces primalac terminira**
  + **Mailbox kreira OS u ime nekog procesa koji postaje njegov vlasnik**
  + **Takav mailbox se uništava na zahtev vlasnika ili kada se vlasnik terminira**

**Uzajamno isključivanje prenosom poruka**



* Kreira se **mailbox mutex deljiv za N konkurentnih procesa**
* **Send() je neblokirajući**
* **Receive() – proces se blokira kada je poštansko sanduče mutex prazno**
* **Inicijalizacija : šalje se prazna poruka send(mutex, null)**
  + Null je poruka bez sadržaja
* Proces Pi koji želi da uđe u KS pokušava da primi poruku
  + Ukoliko je mailbox mutex prazan, proces se blokira (JER ČEKA DA PRIMI xD)
  + Inače proces Pi ulazi u KS
  + Po izlasku iz KS vraća poruku u mailbox
* Prvi proces Pi koji izvrši **receive()** će ući u KS. Ostali će biti blokirani dok Pi ponovo ne pošalje poruku

**Proizvođač-potrošač sa prenosom poruka**

* Koriste se dva mailbox-a : **mayproduce** i **mayconsume**
  + **Proizvođač generiše podatke i šalje ih u mailbox mayconsume**
  + **Potrošač uzima poruke iz mailbox-a mayconsume dok ima bar jedna poruka**
* Mailbox mayconsume je buffer
  + Buffer je kapaciteta K poruka (globalna promenljiva capacity)
* Mailbox **mayproduce** se inicijalno puni sa K praznih (null) poruka
  + **Broj poruka u mailbox-u mayproduce se smanjuje sa svakom proizvodnjom, a povećava se svakom potrošnjom**
* Može podržavati više proizvođača i potrošača koji imaju pristup do oba mailbox-a
* Sistem može biti distribuiran :
  + Proizvođač i mailbox **mayproduce** na jednom mestu, a potrošač i mailbox **mayconsume** na drugom

**Klasični sinhronizacioni problemi**

* [***Producer – consumer***](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem)
* [***Readers – writers problem***](https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem)
* [***Dining philosophers problem***](https://en.wikipedia.org/wiki/Dining_philosophers_problem)
* [***Slipping barber problem***](https://en.wikipedia.org/wiki/Sleeping_barber_problem#:~:text=In%20computer%20science%2C%20the%20sleeping,so%20in%20an%20orderly%20manner.)