



Operativni sistemi 1

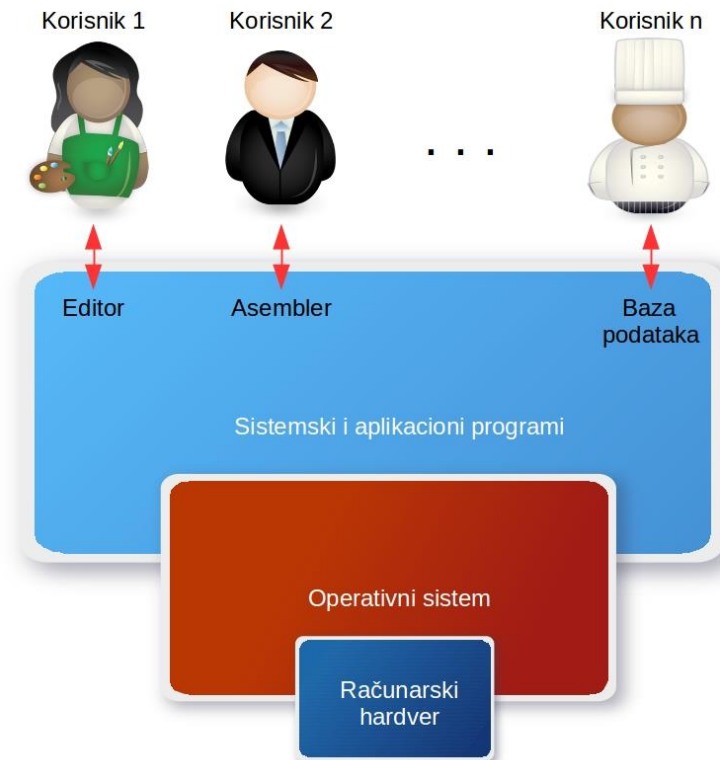
# Konkurentno programiranje - uvod

Predmetni asistenti:

Mihailo Obrenović, Jelica Vasiljević

# Operativni sistem

- Šta je operativni sistem?
  - program koji služi kao posrednik između korisnika, odnosno njegovih programa i računarskog hardvera
- Čemu služi operativni sistem?
  - apstrahuje hardver
    - Aplikativni programi dele niz zajedničkih operacija npr. I/O
    - OS obuhvata upravljanje hardverskim resursima: CPU, memorija, I/O itd.
  - kontroliše i upravlja izvršavanjem svih poslova



# Sistemske pozivi

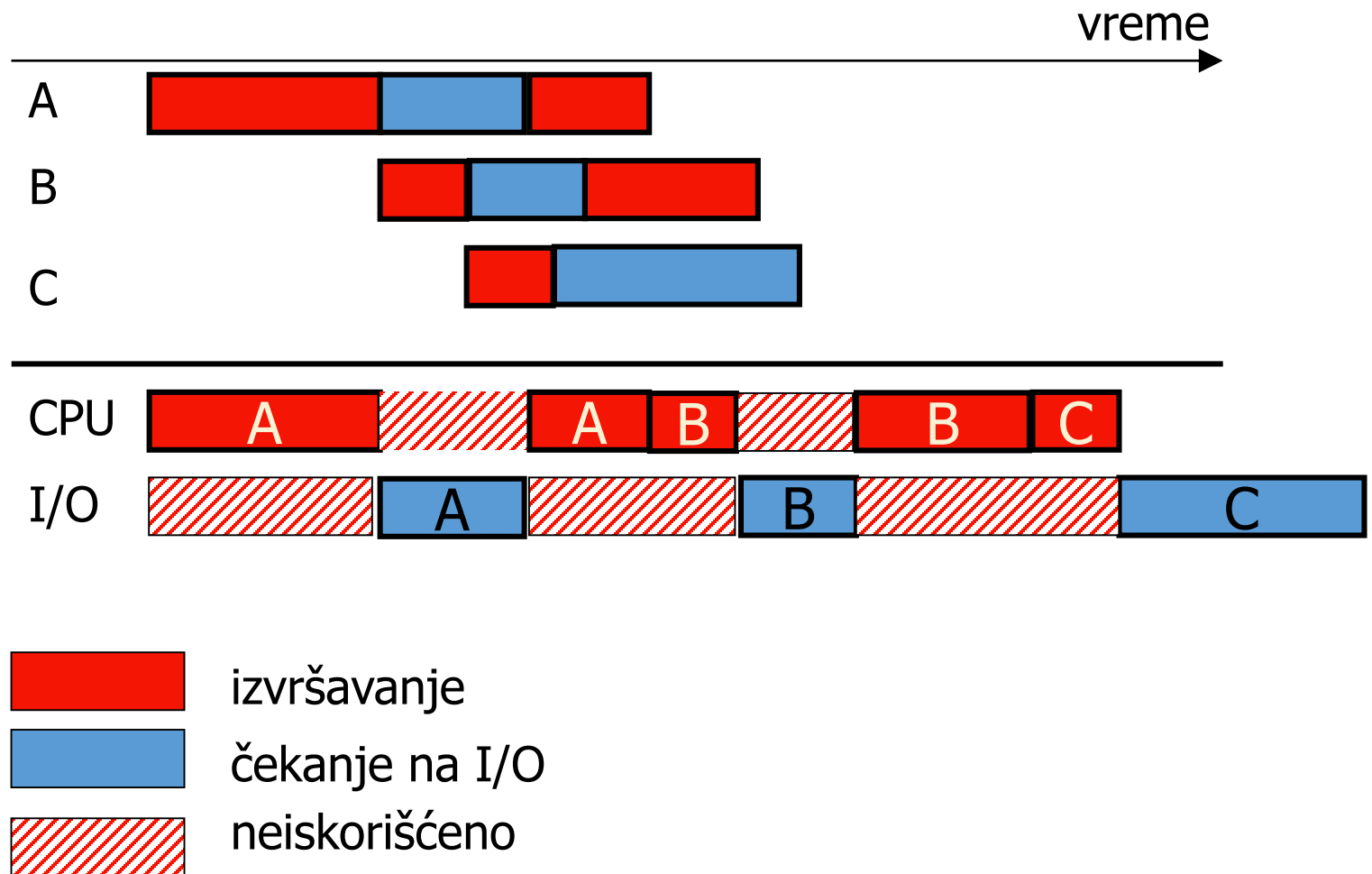
- Način komunikacije sa OS-om
- Računarski poziv – program zahteva uslugu od jezgra OS-a, npr:
  - Hardverske usluge: pristup uređajima, memoriji itd.
  - Poslovi : pokretanje, zaustavljanje, pauziranje itd.
- Linux ima preko 300 različitih poziva

# Razvoj OS

- Uniprogramski
  - Jedan korisnički program u jednom trenutku
- Multiprogramski
  - Više korisničkih programa istovremeno
  - Problem - zajednički resursi: CPU, memorija, I/O uređaji
- Sistemi sa vremenskom raspodelom
  - Multiprogramski u osnovi
  - Omogućavaju interaktivnost (kratko vreme odziva)

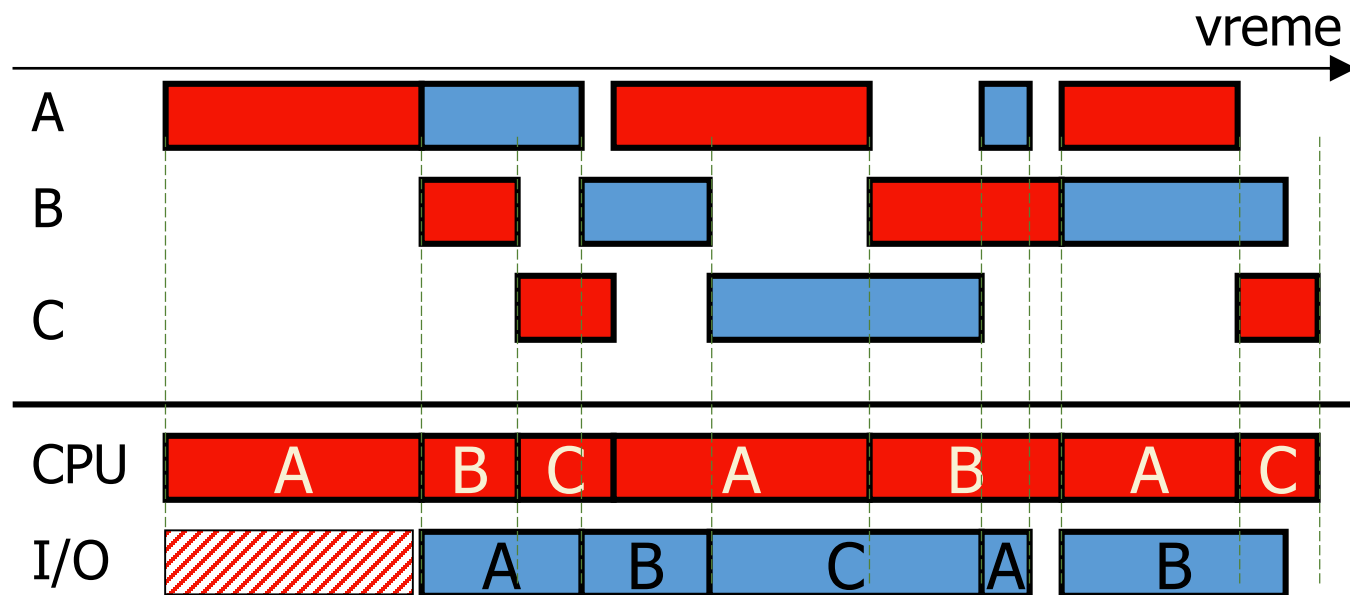
# Uniprogramski sistemi

Jedan korisnički program u jednom trenutku



# Multiprogramski sistemi

- Više korisničkih programa simultano
- Programi nezavisni, resursi zajednički
- Resursi – procesor, memorija, I/O uređaji

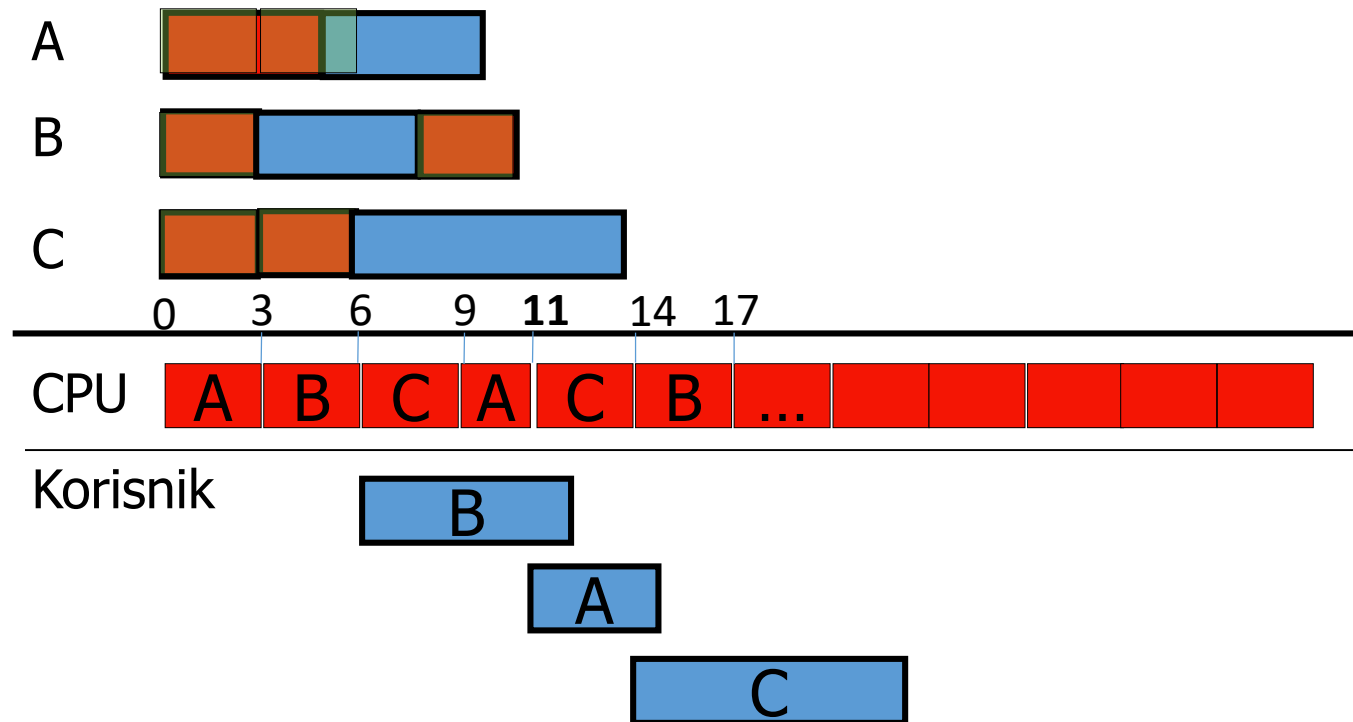


# Multiprogramski sistemi

- Izazovi:
  - Raspoređivanje poslova : kako odabrati koji je sledeći posao koji se izvršava
  - Upravljanje memorijom
  - Šta se dešava kada posao zatraži I/O operaciju
    - Kako zapamtiti da posao izvršava ili čeka na izvršenje I/O
    - Kako odabrati sledeći posao za CPU
    - Kako rasporediti poslove kada imaju istovremeno zahtev za istim resursima (npr. I/O)
  - Kako sprečiti da poslovi ne ugrožavaju jedan drugog
  - Šta raditi sa poslovima koji beskonačno traju

# Vremenska podela procesora

- Preduslov : multiprogramiranje
- Više korisnika komunicira (interaktivno) sa sistemom
  - Svaki korisnik dobije procesor na određeno vreme
  - Promena je toliko brza da svaki korisnik stiče utisak da sam koristi ceo sistem
- **Potrebni mehanizmi za sinhronizaciju i komunikaciju**
- Strategija raspoređivanja





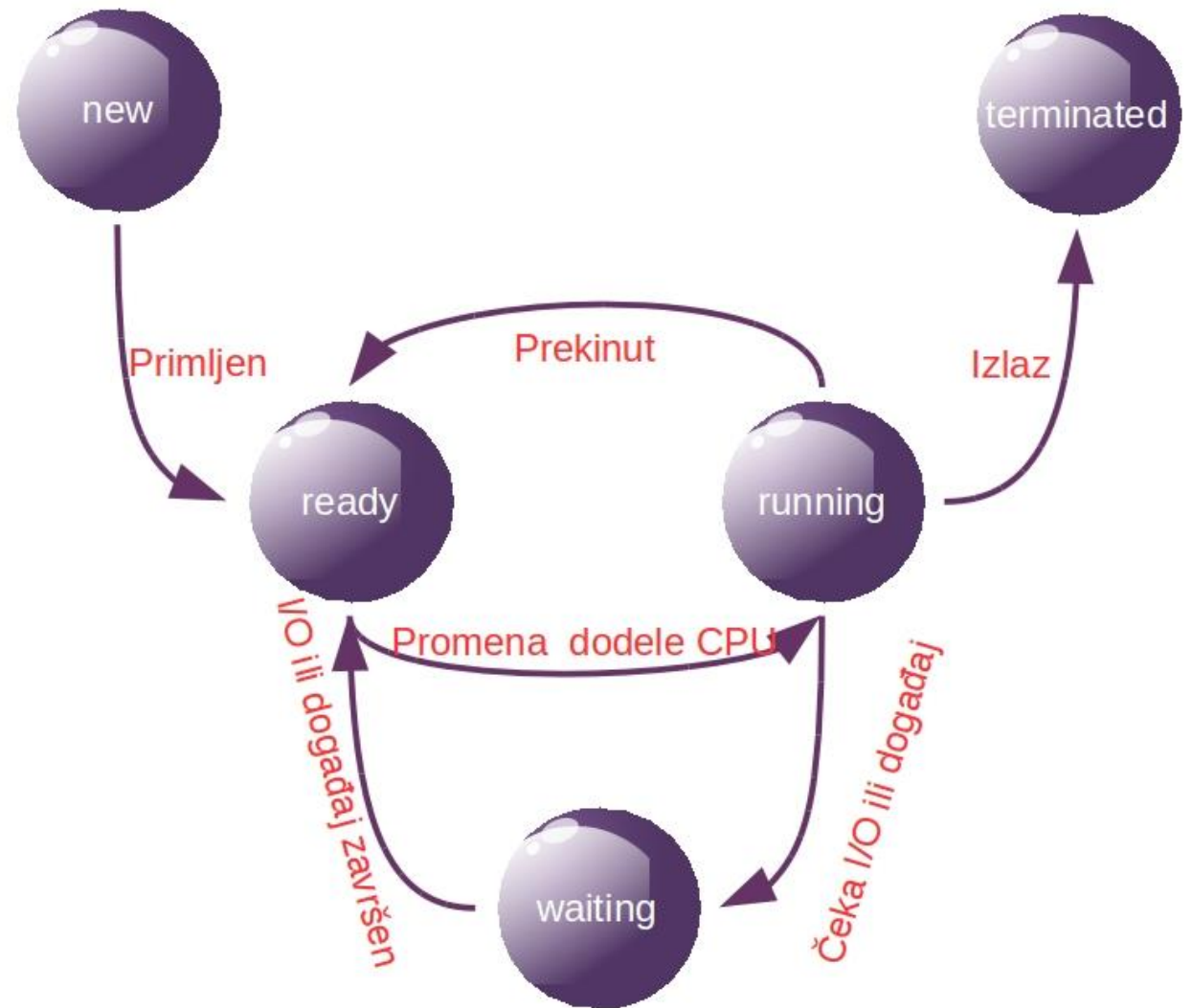
# Proces

- Program
  - Fajl na HDD-u, spisak instrukcija
  - Statičan, pasivan entitet
- Proces
  - Instanca programa, izvršavanje instrukcija
  - Dinamički, aktivni entitet
  - Nastanak procesa – program se učitava u glavnu memoriju
  - Više procesa može izvršavati isti kod
  - Svaki process radi nad svojim podacima – svaki ima svoj adresni prostor
  - Kreiranje, izvršavanje, prekid

```
int main()
{
    int i, prod = 1;
    for (i = 0; i < 100; i++)
        prod = prod * i;
    return 0;
}
```

# Stanje procesa

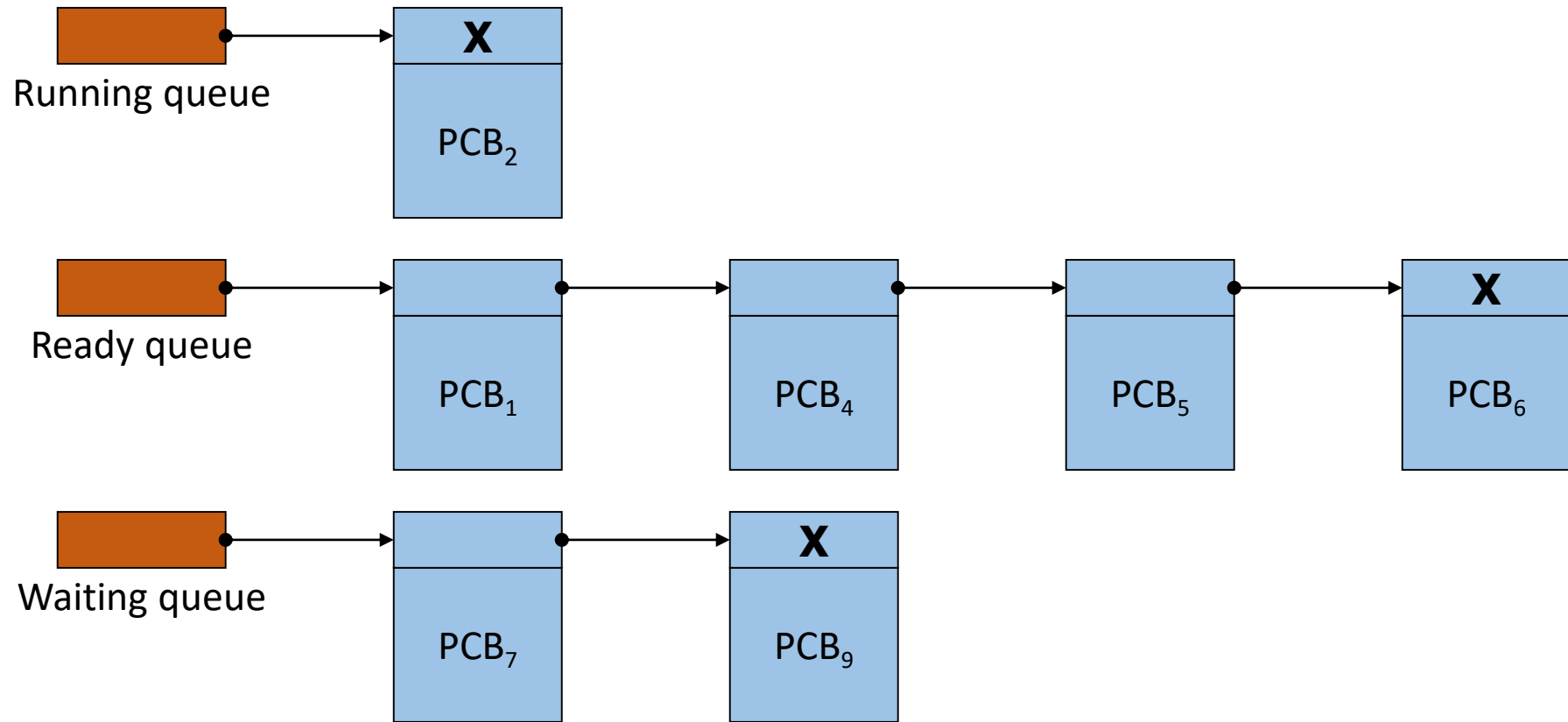
- New – proces kreiran
- Running – instrukcije se izvršavaju
- Waiting – Proces čeka da primi signal, da I/O uređaj završi posao i sl.
- Ready – proces čeka dodeljivanje procesoru
- Terminated – proces završio sa radom

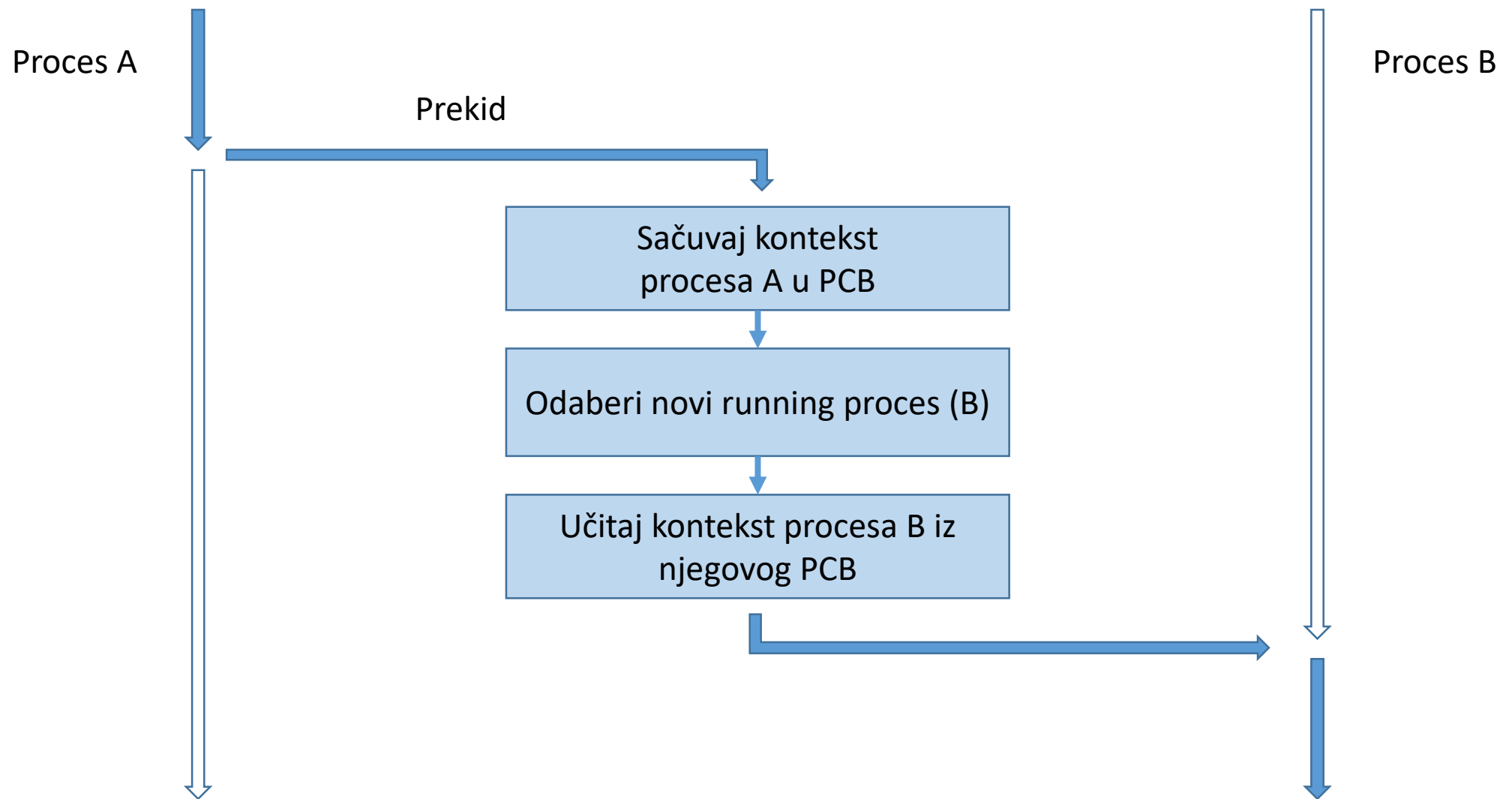


# Kontrolni blok proces (PCB)

- Kontekst procesa – sve informacije potrebne da OS nastavi izvršavanje procesa
  - Identifikator – PID
  - Stanje – izvršavanje, blokiranje, čekanje
  - Prioritet – u odnosu na ostale procese
  - Brojač instrukcija
  - Memorijski pokazivači – ka programskom kodu, ka podacima procesa, ka deljenim podacima
  - Kontekstni podaci – sadržaj registara
  - I/O statusi – zahtevi, uređaji dodeljeni procesu, lista fajlova
  - Informacije naloga – upotrebljeno procesorsko vreme, vremenska ograničenja
- Struktura podataka u kojoj se čuvaju ove informacije za svaki proces unutar OS – *Process Control Block* (PCB)







# Procesi

- Kreiranje procesa
  - OS pri pokretanju kreira nekoliko procesa
  - Aktivan proces može kreirati nove procese
  - Proces roditelj, procesi deca
  - Stablo procesa
- Prednji procesi – komunikacija sa korisnikom
- Pozadinski procesi (daemons) – specifične funkcije

# Fork

- Sistemski poziv za kreiranje novih procesa (jedini)
- Klonira pozivajući proces
- U programskom jeziku C, biblioteka *unistd*:
  - *pid\_t fork(void)*
  - *pid\_t* – tip procesa, *short* na Unix-u
  - Roditelj i dete proces – iste kopije podataka, isti kod
- Funkcija *fork* vraća:
  - Roditelju – PID procesa deteta
  - Detetu – 0
  - Neuspešno kreiranje deteta – -1

# Fork

- Proces dete ostavlja za sobom određene podatke nakon kraja
- Proces roditelj čeka i kupi te podatke
- *void exit(int status)* – proces dete završava
- *pid\_t wait(int \*status)* – roditelj čeka bilo koje dete
- *pid\_t waitpid(pid\_t pid, int \*status, int options)* – roditelj čeka konkretno dete
- Čekanje dozvoljava da ostatak procesa deteta nestane



# Fork

- Proces siroče (*orphan*):
  - Proces roditelj se završi bez čekanja deteta
  - Proces dete se još uvek izvršava – ostaje bez roditelja
  - Proces dete se dodeljuje *init (systemd)* procesu
- Proces zombi, ili *defunct* proces
  - Proces dete se završi i izađe
  - Proces roditelj se još uvek izvršava
  - Roditelj još uvek nije pozvao *wait()*
- Ignorisanje SIGCHLD signala
  - *signal(SIGCHLD, SIG\_IGN)* – pozvati pre *fork* naredbe
  - Sada roditelj ne mora da čeka – nema zombi procesa