Uvod u procese

1. Proces, definicija.Razlika između procesa i programa. Šta sve sadrži proces?

Proces predstavlja jedan od najvažnijih koncepata operativnih sistema. Proces je program ili deo programa u stanju izvršenja, zajedno sa svim resursima koji su potrebni za rad programa.
Jednostavno rečeno, program je pasivan objekat, to jest datoteka na disku. Kada se ta datoteka ucita u memoriju, ona postaje proces, tj. Aktivan objekat koji ima svoje resurse, poput registra i memorije.
Svaki proces ima tri fundamentalna memorijska dela, to jest sekcije:

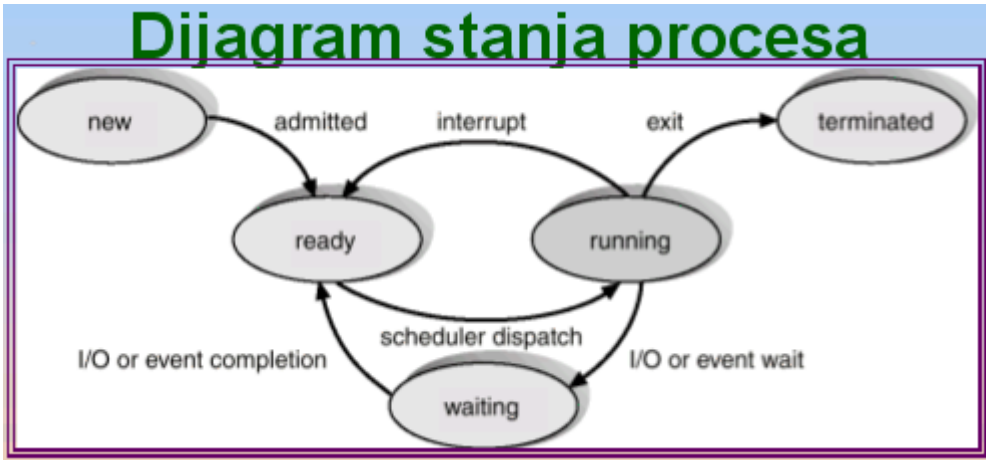
- **Programska ili tekstualna sekcija, koja se ne menja (engl. *Read only*) i koja sadrzi programski kod;**
- **Stek sekcija (engl. *Stack section*), koja sadrzi privremene podatke (parametre za procedure, povratne adrese, lokalne promenljive);**
- **Sekcija podataka (engl. *Data section*), koja sadrži globalne promenljive.**

Osim memorijskih sekcija, proces obuhvata i vrednost programskog brojaca (engl. *Program counter*), kao i vrednost ostalih važnih registara procesora. Proces obuhvata i ulazno-izlazne resurse koje eventualno koristi, kao sto su datoteke i razne vrste I/O uređaja.

2. Koja su stanja procesa, nacrtati dijagram stanja procesa?

Proces se moze naći u nekoliko stanja (pet ili sedam, zavisno od konačnog automata koji je korišćen za opisivanje automata) a stanja su sledeća:

- **Kreiranje procesa (NEW, START) . Proces je kreiran.**
- **Stanje cekanja na procesor (READY, RUNNABLE). Proces je dobio sve potrebne resurse osim prcesora, spreman je za red i ceka da mu se dodeli procesor.**
- **Stanje izvrsenja (RUN, RUNNING). Proces izvrsava instrukcije ovog procesa.**
- **Stanje cekanja na resurs (WAIT, UNRUNNABLE). Proces čeka na neki događaj (npr. Da se izvrši štampanje), jer su za dalje izvršavanje procesa potrebni resursi koji trenutno nisu na raspolaganju. Takođe, proces može ući u stanje WAIT ukoliko čeka završetak neke I/O operacije ili rezultat nekog drugog procesa.**
- **Kraj izvršenja procesa (TERMINATED, STOP).**



3. Šta je to PCB i koje informacije sadrži

PCB (*Process Control Block*) je kontrolni blok koji je deo radne memorije, to jest memorijska struktura sa osnovnim informacijama o procesu, koje operativni sistem koristi za upravljanjem tim procesom.
Zahvaljujuci kontrolnom bloku, izvršavanje programa se moše prekinuti i nastavljati više puta.
Informacije vezane za svaki proces:

- **Stanje procesa**
- **Programski brojač**
- **Sadržaj CPU registara**
- **Informacije o memoriji procesa**
- **Lista otvorenih datoteka**
- **Statističke informacije**
- **Status I/O resursa**

4. Objasniti pojam job queue, ready queue, device queue?

Nakon nastanka, proces se ubacuje u red čekanja za poslove (engl. *Job queue*), koji obuhvata sve postojeće procese na sistemu. Procesi se mogu nalaziti u raznim stanjima i na raznim lokacijama (u memoriji, na disku, delimično u memoriji, delimično na disku).

Svi procesi koji su spremni za rad i nalaze se u radnioj memoriji čuvaju se u redu čekanja na procesor, to jest u redu čekanja spremnih procesa (engl. *Ready queue*).

Operativni sistem uvodi i poseban red čekanja za svaki ulazno-izlazni uređaj (engl. *I/O queue, Device queues*). Svaki red čekanja na uređaj sadrži povezanu listu kontrolnih blokova procesa koji taj uređaj zahtevaju.

5. Šta su schedulers programi?

Programi za raspoređivanje (engl. *Schedulers*) odlučuju o tome kada će proces ući u neki red čekanja ili napustiti taj red. Postoje:

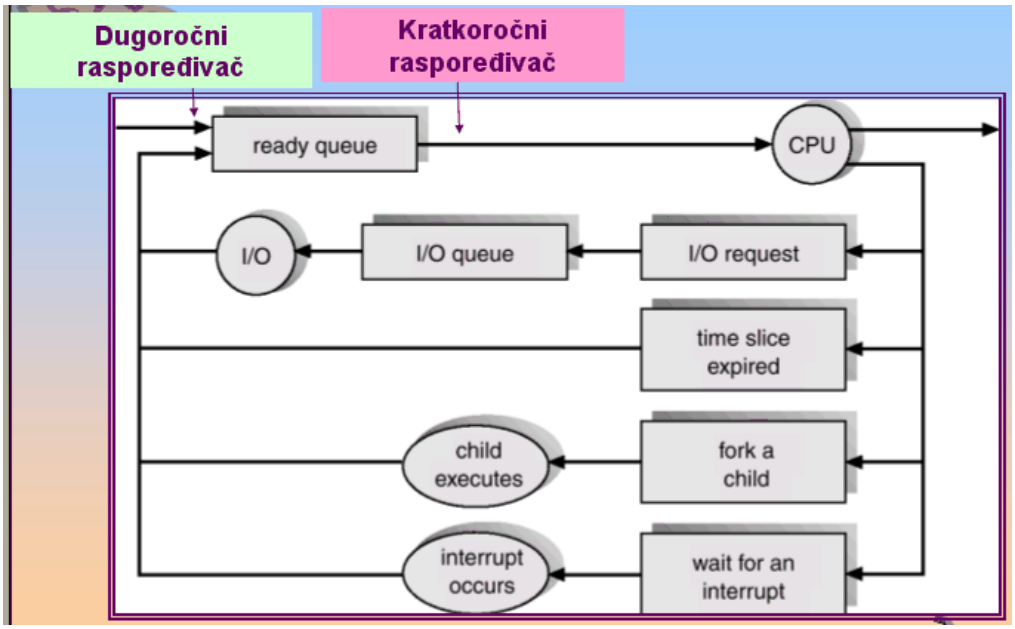
- **Planer poslova (engl. j*ob scheduler, long-term scheduler, high-level scheduler*);**
- **Dispačer (engl. d*ispatcher, short-term scheduler, low-level scheduler*).**

6. Definisati Long-term scheduler (job scheduler), short therm sheduler, medium-term scheduler, skicirati

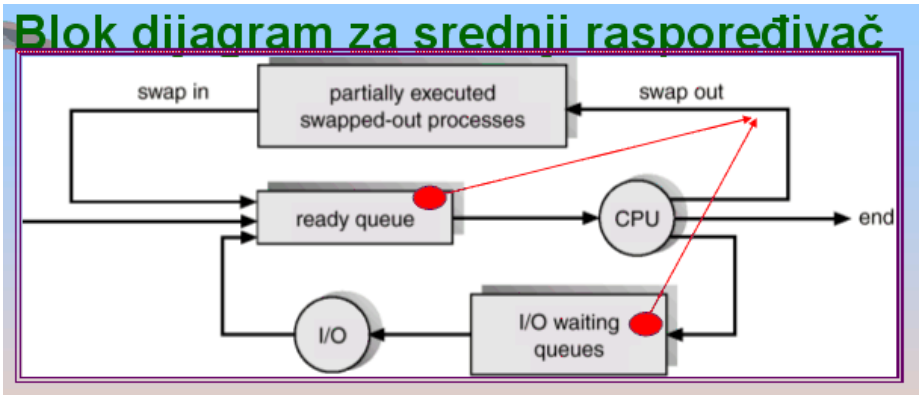
Long-term scheduler (planer poslova) koji se u hijerarhijskom modelu nalazi iznad jezgra, obavlja sledeće funkcije:

- **deli poslove na procese;**
- **na osnovu određenih algoritama dodeljuje prioritete procesima;**
- **dovodi procese u red čekanja na processor.**

Short-term scheduler (dispečer) – Zadatak dispečera sistema je da dodeljuje processor procesima koji se nalaze u procesorskom redu. Dispečer dodeljuje CPU kad god tekući process pređe iz stanja RUN u stanje WAIT ili READY. Dispečer utvrđuje kom je procesu najpovoljnije dodeliti processor, to jest koji je process u stanju READY najvišeg prioriteta.



Medium-term scheduler (srednji raspoređivač) – svaki novostvoreni process odlazi u red čekanja, nakon čega dispečer odlučuje kome će dodeliti processor. Neki procesi mogu biti suspendovani, to jest privremeno prekinuti i upisani na disk (engl. *Swap space*), čim se oslobađa memorija za druge procese. Kada se sa diska premeste u memoriju, suspendovani procesi se vraćaju u red čekanja na procesor. Funkciju suspenzije procesa (engl. *Swap-out*), funkciju povratka procesa u stanje spremnosti (eng. *Swap-in*) i izbor procesa za obe funkcije obavlja srednji raspoređivač.



7. Procesi se dele na CPU-bound i I/O bound (objasniti)

- **I/O intenzivne procese *(bound process)* –**
  - **najveći deo njihovog vremena izvršavanja otpada na I/O cikluse, relativno mala upotreba CPU.**

- **CPU intenzivne procese *(bound process)* –**
  - **najveći deo svog vremena korsite na CPU izračunavanja, veoma velika upotreba CPU.**

8. Šta je to context-switch, dispatcher?

Prilikom dodele procesora drugom procesu, to jest zamene procesa koji se trenutno izvšava, dispečer :

- **Pamti stanje procesa koji se prekida, kako bi se proces mogao kasnije nastaviti;**
- **Puni memoriju stanjem novog procesa kome se dodeljuje procesor.**

Navedene operacije su poznatije pod nazivom zamena konteksta procesa (eng. *Context switch*). Kontekst procesa čine podaci koji se čuvaju prilikom oduzimanja procesora, a koji omogućuju nastavak izvršenja procesa, kao što su registri procesora, memorijske sekcije i lista otvorenih datoteka.
Prebacivanje konteksta je čist gubitak vremena i predstavlja premašenje sistema (engl. *overhead*), ali je neophodno radi omogućavanja multiprogramiranja. Premašenje zavisi od hardveskih performansi.

9. Osnovne opracije nad procesima?

- **Izrada novog procesa (pravljenjem kontrolnog bloka i ažuriranjem procesne liste);**
- **Izrada veza procesa – proces roditelj;**
- **Uništenje procesa (brisanjem kontrolnog bloka iz procesne liste);**
- **Promena stanja procesa (obavljanje tranzicija u dijagramu stanja, kao što je dodela procesora);**
- **Promena prioriteta procesa.**

10. Kreiranje procesa, veza roditelj/dete po pitanju deobe adresnog prostora

Proces u toku svog izvšenja može, odgovarajućim sistemskim pozivom, praviti nove procese. Proces koji pravi nove procese jeste roditeljski proces (engl. *Parent process*), dok se proces koji je roditelj napravio naziva dete proces (engl. *Child process*). Svaki proces koji je dete drugog procesa može dalje stvarati nove procese – na taj način se formira hijerarhijska struktura procesa, tj. Stablo aktivnih procesa.
Odnos proces – proces roditelj može se opisati na osnovu načina deljenja resursa i načina izvršenja.
Prema deljenju resursa, procesi roditelj i dete mogu se naći u sledećim relacijama :

- **Procesi roditelj i dete dele sve resurse;**
- **Procesi roditelj i dete dele podskup resursa roditeljskog procesa;**
- **Procesi roditelj i dete ne dele resurse.**

Posebno je osetljiv memorijski adresni prostor, za čije se korišćenje primenjuju sledeće tehnike :

- **Proces dete duplicira adresni prostor roditelja;**
- **Adresni prostor deteta generiše se prema programu kojim se taj adresni prostor puni.**

Prema načinu izvršavanja, proces se sa svojim roditeljem moše naći u sledećim relacijama :

- **Proces roditelj nastavlja da se izvršava nezavisno i konkuretno sa detetom;**
- **Proces roditelj se blokira i čeka dok proces dete ne završi svoje aktivnosti.**

11. Prikazan je code za UNIX fork u C kodu, objasniti?

```
/* fork another process*/
pid = fork ();

if (pid = =  0 )
 {                                  /*child process*/
    execlp("/bin/ls", "ls", NULL);
 }

else /*parent process*/
```

```
/*parent will wait for child to complete*/
  wait(NULL);
   printf("child complete");
   exit (0);
   }
}
```

Zadati program na jeziku C ilustruje način funkcionisanja procesa pod UNIX sistemom, a primer odgovara slučaju komandnog interpretera /bin/sh koji izvršava komandu ls . Isti kod izvršavaće i proces roditelj i proces dete. Roditelj pravi nov proces pomoću sistemskog poziva fork , koji roditelju i detetu dodeljuje različit PID. Proces dete čiji je PID 0, izvršava sistemski poziv exec koji će u dodeljeni adresni prostor učitati program ls i izvršiti ga. Proces roditelj čiji je PID različit od 0, čeka da se komanda ls završi (sistemski poziv wait). Tek kada proces dete završi aktivnosti, roditelj će izaći iz sistemskog poziva wait.

12. Kooperacija procesa, kompeticija, sinhronizacija

U odnosu na međusobnu zavisnost toka izvršavanja, procesi se mogu podeliti na:

- **Nezavisne procese;**
- **Kooperativne procese.**

Proces je nezavisan ukoliko ne utiče direktno na izvršenje drugih procesa i ukoiko na njegovo izvršenje direktno ne utiču drugi procesi. Nezavisni su oni procesi koji ne dele nikakve podatke sa drugim procesima.
Kooperativni procesi su oni koji jedni na drugi ne utiču, a to su svi procesi koji dele podatke ili bilo kakve resurse. Kooperativni procesi zahtevaju specijalan tretman i okruženje iz sledećih razloga:

- **Deljenje informacija**
- **Ubrzavanje rada**
- **Modularnost**
- **Pogodnost**

Konkuretno izvršavanje kooperativnih procesa zahteva mehanizam koji će dozvoliti procesima da komuniciraju između sebe i da sinhronizuju svoje akcije. Problem sinhronizacije se može objasniti na čuvenom odnosu proizvođač – potrošač (engl. *Producer-cunsumer*) koji predstavlja zajedničku paradigmu kooperativnih procesa.

- **Sinhronizacija procesa**
  - **turn**
  - **flags**
  - **semaphores**
  - **monitor**
  - **IPC (sistem poruka)**

13. Završetak procesa, diskutovati 2 vrste završetka procesa

Proces završava svoje aktivnosti onda kada završi poslednju naredbu programa koji je učitan u adresni prostor tog procesa.
Posle toga, koristeći sistemski poziv exit, proces traži od operativnog sistema da ga uništi, to jest obriše. Pri tome, proces dete može vratiti izlazne podatke roditeljskom procesu pomoću sistemskog poziva wait. Potom se oslobađaju svi resursi koji su pripadali procesu, kao što su fizička i virtuelna memorija, datoteke i I/O baferi.
Proces može izazvati i nasilan završetak rada drugog procesa pomoću specijanog sistemskog poziva abort. Po pravilu, proces roditelj može prekinuti izvršavanje procesa koje je on napravio, a to čini iz sledećih razloga:

- **Proces dete je prilikom korišćenja nekog resursa (memorija, sistemskog vremena) premašio kvotu;**
- **Aktivnost koju proces dete obavlja više nije potrebna;**
- **Proces roditelj je završio aktivnosti pre dece, što se smatra nenormalnom situacijom koju operativni sistemi ne dozvoljavaju – u tom slučaju se svi procesi „deca" moraju nasilno uništiti (engl. *Cascade termination*).**

14. Primer kooperacije na slučaju producer-consumer

- **Paradigma za kooperativne procese, proizvođač proces proizvodi informacije koje troši proces potrošač;**
- **Ta dva procesa mogu da rade konkuretno, ukoliko postoji bafer koji će puniti proizvođač, a prazniti potrošač.**

Pravila sinhronizacije su sama po sebi jasna : potrošač ne može uzeti ništa iz bafera ukoliko proizvođač to prethodno nije stavio u bafer.
Bafer takođe nameće svoja pravila:

- **bafer beskonačnog kapaciteta *(unbounded-buffer)* nema ograničenje u veličini bafera.**
- **ograničeni bafer *(bounded-buffer)* ima ograničenu veličinu bafera.**

15. IPC, šta je to i koje sve metode postoje

IPC (Inter Process Communication) – Interprocesna komunikacija
Mehanizam za procese koji omogućava da komuniciraju i da sinhronizuju njihove akcije.
Tri metode:

- **signali (UNIX) , semafor**
- **sistem poruka**
- **deljena memorija**

16. IPC messages systems

Da bi procesi mogli da komuniciraju slanjem poruka, neophodno je u jezgro implementirati sledeće dve osnovne operacije (engl. *primitives*):

- **Slanje poruke (eng. *Send message, post message*);**
- **Prijem poruke (engl. *Receive m*essage).**

Poruke koje se razmenjuju između procesa mogu biti fiksne ili promenljive veličine. Metod slanja poruka fiksne veličine jednostavnije se implementira ali je manje fleksibilna, dok se slanje poruka promenljive veličine teže implementira ali je fleksibilnije.

Ako P i Q žele da komuniciraju, oni trebaju da:

- **uspostave komunikacioni link između njih**
- **poruke razmenjuju preko send/receive operacija**

Implementacija komunikacionog linka:

- **fizička (npr. deljena memorija, hardverska magistrala)**
- **logička (npr. logičke osobine)**

17. IPC messages-direct communications

Procesi koji žele da komuniciraju moraju da imaju mogućnost imenovanja kako bi mogli međusobno jednoznačno da se identifikuju. U direktnoj šemi, procedure za slanje I primanje moraju eksplicitno da sadrže ime procesa s kojim žele da komuniciraju. Postoje i asimetrična varijanta adresiranja, u kojoj pošiljalac mora da navede ime primaoca, dok primalac ne mora da navede ime pošiljaoca.

send (P, poruka) – poslaće poruku procesu P
receive (Q, poruka) – primiće poruku od procesa Q // simetrično adresiranje
receive (id, poruka) //asimetrično adresiranje

Pravila direktne komunikacije su sledeća:

- **veza se automatski uspostavlja između para procesa koji žele da komuniciraju;**
- **jedna veza se uspostavlja samo za dva procesa;**
- **veza može biti unidirectional, ali se koristi i bi-directional.**

18. IPC messages –indirect communications

Pri inirektnoj komunikaciji, poruke se šalju ili primaju preko poštanskih sandučića (*mailbox*) ili priključaka (engl. *ports*). Poštansko sanduče se može posmatrati kao objekat u kome operativni sistem ostavlja poruke drugih procesa. Svako sanduče ima jedinstvenu identifikaciju. Dva procesa mogu komunicirati samo ako dele sanduče. U indirektnoj šemi, proces može komunicirati s procesom preko više različitih sandučića.

Naredbe za slanje poruka :

- **send (A, poruka) – poslaće poruku u sanduče A**
- **receive (A, poruka) – primiće poruku iz sandučeta A**

19. Sinhronizacija poruka u IPC message sistemu

Postoje različite šeme za realizaciju osnovnih operacija send i receive :

- **sinhrono ili blokirajuće (engl. *blocking*) slanje i primanje poruka;**
- **asinhrono ili neblokirajucće (engl. *nonblocking*) slanje i primanje poruka.**

20. IPC in client/server okruženju, socket

Jedan od najčešće korišćenih mehanizama jeste mehanizam utičnica (engl. *sockets*), patent proizvođača BSD UNIX sistema, koji je prihvaćen na svim operativnim sistemima sa mrežnom podrškom. Utičnica se definiše kao krajnja tačka komunikacije. Par procesa koji želi da komunicira preko mreže formira par priključaka, po jedan za svaki proces, na svakoj strani mreže. Utičnica se definiše pomoću parametra – IP: priključak (engl. *port*), tj. pomoću IP adrese računara na kome se priključak formira i odgovarajućeg broja priključka.

21. Objasniti RPC

U klijent/server okruženju uvodi se apstrakcija klasičnog poziva procedure pozivom udaljene procedure (eng. *Remote Procedure Call - RPC*). RPC omogućava procesu koji se izvršava na jednom računaru da pozove proceduru na drugom računaru. RPC funkcioniše na sledeći način : najpre se na serverskoj i na klijentskoj strani definišu strukture poznate pod imenom vezivna funkcija (engl. *stub*) između kojih se preciznim slanjem poruka i parametara (engl. *marshalling parameters*) ostvaruje komunikacija.

- **Stub klijentske strane :**
  - **pronađe server i šalje parametre.**
- **Stub serverske strane**
  - **prima ovu poruku,**
  - **raspakuje parametre, i**
  - **izvršava proceduru na serveru.**