Sveučilište J. J. Strossmayera u Osijeku

Fakultet elektrotehnike, računarstva i informacijskih tehnologija

Kneza Trpimira 2b

HR-31000 Osijek

www.ferit.unios.hr

Laboratorijska vježba 5:

Raspoređivanje procesa

Sadržaj

I. UVOD

U operacijskom sustavu koji podržava istovremeni rad više programa (engl. *multiprogramming system*), više procesa se konkurentno natječe za radnu memoriju računala. Svaki proces izmjenjuje stanja između korištenja procesora ili čekanja na određeni događaj. S druge strane, procesor je gotovo uvijek zaposlen izvršavajući procese, dok neki procesi čekaju. Ključna stavka *multiprogramming* koncepta je efikasno i efektivno **raspoređivanje** procesa. Cilj raspoređivanja procesa je pridružiti procesoru procese koje će izvršiti tijekom vremena, na način da se ispune sustavski zahtjevi te efikasno iskoristi procesor.

U ovoj laboratorijskoj vježbi ćemo izložiti teorijsku podlogu raspoređivanja u današnjim operacijskim sustavima. Detaljnije ćemo proučiti poznate algoritme za raspoređivanje te prikazati njihov rad na zadanim skupovima procesa. U konačnici, implementirati ćemo jedan od algoritama u programskom jeziku C. U zadacima za samostalni rad, potrebno je izraditi implementaciju preostalih algoritama na sličan način.

I.1. Tipovi raspoređivanja procesa

U većini operacijskih sustava, raspoređivanje procesa se ostvaruje kroz tri tipa raspoređivanja:

- **Dugoročno raspoređivanje** (engl. *long-term scheduling*) - kontrolira stupanj *multiprogramming*-a; provodi se kada se pojavi novi proces u sustavu te se on postavlja u red i dodjeljuje kratkoročnom raspoređivaču
- **Srednjoročno raspoređivanje** (engl. *medium-term scheduling*) - dio procesa izmjene (engl. *swapping*); raspoređivanje dijela procesa u glavnu memoriju
- **Kratkoročno raspoređivanje** (engl. *short-term scheduling*) - stvarno odlučivanje o tome koji spremni proces se izvršava sljedeći

Dio operacijskog sustava zadužen za raspoređivanje se naziva i raspoređivač (engl. *scheduler*). Odluke o raspoređivanju procesa na procesor donose se u sljedećim okolnostima:

1. Kada proces prelazi iz aktivnog stanja u stanje čekanja (zbog U/I zahtjeva ili poziva za čekanje na završetak nekog podređenog procesa).
2. Kada proces prelazi iz izvršavanja u stanje spremnosti (npr. kada dođe do prekida).
3. Kada proces prelazi iz stanja čekanja u stanje spremnosti (npr. dovršavanje U/I operacije).
4. Kada proces završava.

U slučajevima 1 i 4, nema izbora u pogledu raspoređivanja - novi proces se mora izabrati za izvršavanje. Međutim, u slučajevima 2 i 3, moguće je donijeti razne odluke. Kada se raspoređivanje odvija samo pod okolnostima 1 i 4, kažemo da je shema raspoređivanja neprekidna (engl. *non-preemptive*); inače je shema prekidna (engl. *preemptive*). Kod neprekidnog raspoređivanja, jednom kad se procesor dodijeli procesu, proces ga zadržava sve do završetka ili prelaska u stanje čekanja. Ovakva vrsta raspoređivanja se koristila u operacijskim sustavima Microsoft Windows 3.1 i Apple Macintosh. U slučaju prekidnog raspoređivanja, zadacima se uglavnom dodjeljuju prioriteti. U određenim trenucima, potrebno je izvršiti zadatak većeg prioriteta prije zadatka koji se već izvršava. Zbog toga se tekući zadatak na neko vrijeme prekida i ponovno nastavlja nakon završetka prioriteta zadatka.

1.2. Politike raspoređivanja

1.2.1. Kriteriji kratkoročnog raspoređivanja

Glavni cilj kratkoročnog raspoređivanja je alociranje vremena procesora na način da se optimiziraju aspekti ponašanja sustava. Skup pravila kojima se odabire i raspoređuje proces se naziva politikom raspoređivanja (engl. *scheduling policies*). Općenito, određuje se set kriterija prema kojima se evaluiraju različite politike raspoređivanja. Najčešći kriteriji raspoređivanja se dijele u 2 dimenzije:

- Korisnički-orijentirani - ponašanje sustava s aspekta korisnika (npr. vrijeme odziva)
- Sustavski-orijentirani - efektivno i efikasno iskorištenje procesora

Korisnički-orijentirani kriteriji su važniji od sustavski-orijentiranih, no obje kategorije kriterija se mogu uzeti u obzir. Tablica 1 sažima ključne kriterije raspoređivanja. Svi prikazani kriteriji su međuzavisni i nemoguće ih je sve optimizirati istovremeno.

Tablica 1. Kriteriji kratkoročnog raspoređivanja procesa

	Kriterij	Značenje
Korisnički-orijentirani	Vrijeme obrade (engl. *turnaround time*)	Interval između podnošenja zahtjeva za procesom i njegova izvršavanja
	Vrijeme odziva (engl. *response time*)	Interval između podnošenja zahtjeva za procesom i odgovora sustava
	Vremenski rok (engl. *deadline*)	Specificirano prihvatljivo vrijeme završetka
	Predvidivost	Traženi proces se treba izvršiti u otprilike jednakoj količini i za jednak trošak neovisno o sustavu
Sustavski-orijentirani	Propusnost (engl. *throughput*)	Poželjno je maksimizirati broj završenih procesa po jedinici vremena
	Iskorištenje procesora (engl. *proces utilization*)	Postotak vremena u kojem je procesor zaposlen
	Pravednost (engl. *fairness*)	Procesi se trebaju jednako tretirati
	Provođenje prioriteta	Kada su procesima dodijeljeni prioriteti, politika raspoređivanja ih treba uzeti u obzir
	Balansiranje resursa	Politika raspoređivanja treba podjednako opteretiti računalne resurse

1.2.2. Klasifikacija procesa

Kad je riječ o raspoređivanju procesa, procesi se dijele u tri klase:

- **Serijski procesi (engl. *Batch processes*)** - ne zahtijevaju korisničku interakciju te se često pokreću u pozadini. Pošto nisu vrlo responzivni, raspoređivač ih često penalizira. Tipični *batch* programi su prevoditelji programskih jezika, tražilice baza podataka, znanstveni izračuni i sl.
- **Interaktivni procesi** - oni konstantno komuniciraju sa svojim korisnicima te troše puno vremena čekajući na korisnikove reakcije (npr. pritisak tipke na tipkovnici ili mišu). Kada je dobiven korisnički ulaz, proces se mora brzo probuditi jer će se u suprotnom korisnik imati dojam da sustav ne reagira. U današnjim operacijskim sustavima, prosječno čekanje je između 50 i 150 milisekundi. Tipični interaktivni programi su komandna sučelja, tekstualni editori, grafičke aplikacije i sl.

- **Procesi u stvarnom vremenu** (engl. *real-time processes*) - oni imaju vrlo stroge zahtjeve za izvršavanje. Takvi procesi nikada ne bi trebali biti blokirani procesima nižeg prioriteta i trebaju imati kratko zajamčeno vrijeme odziva s minimalnom varijancom. Tipični programi u stvarnom vremenu su video i zvučni programi, kontroleri robota i programi koji prikupljaju podatke od fizičkih senzora.

1.2.3. Uzroci raspoređivanja

Jedno od glavnih pitanja vezanih za raspoređivanje je kada donijeti odluku o novom rasporedu. Postoji niz situacija u kojima se javlja potreba za raspoređivanjem. Neke od tih situacija su:

- Kreiranje novog procesa - kada se kreira novi proces, donosi se odluka o tome hoće li se prvo izvesti taj proces ili njegov roditelj.
- Završetak procesa - kada je proces završio s izvođenjem, procesor može preuzeti novi proces. Ukoliko niti jedan proces nije spreman, pokreće se proces **idle**.
- Blokiranje procesa u izvršavanju - kada je proces blokiran, novi proces koji je spreman na izvođenje može biti raspoređen na procesor.
- Pojava prekida za blokirani proces - kada je poslan prekid (engl. *interrupt*) od I/O uređaja, proces koji je bio blokiran čekajući na taj prekid, može biti ponovno raspoređen za izvršavanje. Raspoređivač tada mora odlučiti hoće li izvršiti taj proces ili neki drugi u redu čekanja.

1.2.4. Zamjena konteksta

Zamjena konteksta (engl. *Context switch*) je mehanizam za spremanje i vraćanje stanja ili konteksta procesora u kontrolni blok procesa, tako da procesor može nastaviti s izvršavanjem s iste točke u kojoj je bio prekinut. Pomoću ovog mehanizma, više procesa mogu dijeliti isti procesor. Zamjena konteksta je ključni dio višezadaćnog operacijskog sustava.

Kada raspoređivač zamijeni proces koji se izvršava na procesoru, stanje prekinutog procesa se sprema u kontrolni blok procesa. Nakon toga, stanje idućeg procesora se dohvaća sa sabirnice te se postavlja potreban kontekst za njegovo izvršavanje. Kontekst izvršavanja čine podaci:

- Programski brojač
- Informacije o raspoređivanju
- Vrijednosti registara
- Adrese korištenih registara
- Stanje procesa
- Informacije o vezanim I/O uređajima
- Statističke informacije

2. ALGORITMI RASPOREĐIVANJA

Za dizajn algoritama za raspoređivanje, potrebno je imati neku ideju o tome što bi dobar algoritam trebao raditi. Neki ciljevi ovise o okruženju (*batch*, interaktivno, u stvarnom vremenu), dok su neki poželjni u svim slučajevima. Tablica 2 prikazuje kriterije raspoređivanja specifične za spomenute klase procesa.

Tablica 2. Kriterij raspoređivanja prema vrsti okruženja sustava

Okruženje sustava	Kriterij raspoređivanja
***Batch* sustavi**	Maksimizirati propusnost
	Minimizirati vrijeme obrade
	Maksimizirati iskorištenje procesora
Interaktivni sustavi	Smanjiti vrijeme odziva
	Ispuniti zahtjeve korisnika
Sustavi u stvarnom vremenu	Spriječiti gubitak podataka
	Povećati predvidivost
Svi sustavi	Održavati pravednost
	Osigurati provođenje prioriteta
	Osigurati balansiranje resursa

2.1. Raspoređivanje u serijskim (*Batch*) sustavima

2.1.1. *First-Come, First-Served*

Jedan od najjednostavnijih algoritama za raspoređivanje je neprekidni *First-Come, First-Served* (FCFS) algoritam. Prema ovom algoritmu, procesi se raspoređuju na procesor u redoslijedu njihovih zahtjeva. Postoji jedinstven red spremnih procesa. Algoritam je neprekidan, a novi procesi se smještaju na kraj reda. Također, kada blokirani proces postaje spreman za izvođenje, smješta se na kraj reda iza ostalih procesa.

Primjer 1: Raspoređivanje po FCFS algoritmu
Zadan je sustav od 5 procesa s vremenima pojavljivanja: $T_{p1..5}$ = (0, 2, 5, 7, 12) i vremenima trajanja procesa: $T_{t1...5}$ = (2, 5, 7, 2, 3). Primjenom FCFS algoritma skicirajte vremensku os zadanog sustava.

2.1.2. *Shortest Job First*

Drugi neprekidni algoritam raspoređivanja za *Batch* sustave je *Shortest Job First* (SJF). Ovaj algoritam pretpostavlja da su vremena izvođenja poznata unaprijed. Kada postoji nekoliko procesa jednake važnosti, algoritam izabire prvo najkraći od njih. Algoritam je neprekidan.

Primjer 2: Raspoređivanje po SJF algoritmu
Zadan je sustav od 5 procesa s vremenima pojavljivanja: $T_{p1..5}$ = (0, 2, 5, 7, 12) i vremenima trajanja procesa: $T_{t1...5}$ = (2, 5, 7, 2, 3). Primjenom SJF algoritma skicirajte vremensku os zadanog sustava.

P1	P1	P2	P2	P2	P2	P2	P4	P4	P3	P3	P3	P3	P3	P3	P3	P5	P5	P5

2.1.3. *Shortest Remaining Time Next*

Prekidna verzija *SJF* algoritma je *Shortest Remaining Time Next* (SRTN) algoritam. Prema ovom algoritmu, raspoređivač uvijek izabire proces čije je preostalo vrijeme izvršavanje najkraće. On također pretpostavlja da su vremena izvođenja poznata unaprijed. Pri dolasku novog procesa, njegovo ukupno vrijeme izvršavanja se uspoređuje s preostalim vremenima izvršavanja drugih procesa. Ako je ono manje, trenutni proces se suspendira a novi raspoređuje za izvršavanje. *SRTN* algoritam je prekidan.

Primjer 3: Raspoređivanje po SRTN algoritmu
Zadan je sustav od 5 procesa s vremenima pojavljivanja: $T_{p1..5}$ = (0, 2, 5, 7, 12) i vremenima trajanja procesa: $T_{t1...5}$ = (2, 5, 7, 2, 3). Primjenom SRTN algoritma skicirajte vremensku os zadanog sustava.

P1	P1	P2	P2	P2	P2	P2	P4	P4	P3	P3	P3	P5	P5	P5	P3	P3	P3	P3

2.2. Raspoređivanje u interaktivnim sustavima

2.2.1. *Round Robin*

Jedan od najstarijih, najpravednijih i najčešće korištenih algoritama za raspoređivanje je *Round Robin* (RR) algoritam. Prema tom algoritmu, svakom procesu je dodijeljen vremenski interval, zvan **kvant**, tijekom kojeg proces ima pravo na izvršavanje. Ako se proces i dalje izvršava na kraju kvanta, oduzima mu se pravo na procesor i pridaje se drugom procesu. Ako je proces blokiran ili završen prije nego što je istekao njegov kvant, razmjena procesa se događa u tom trenutku. Vrlo je bitno napomenuti da kvant nije prekidan, što znači da dolazak procesa većeg prioriteta ne prekida izvođenje procesa kojemu je trenutno dodijeljen kvant. Raspoređivač mora održavati listu procesa koji su spremni na izvršavanje. Kada proces iskoristi svoj kvant, stavlja se na kraj liste. Ključna postavka ovog algoritma je duljina kvanta. Razmjena procesa zahtijeva određenu količinu vremena i resursa, pa je poželjno postaviti veći kvant. S druge strane, poželjno je što prije rasporediti što više procesa kako se ne bi ostavio dojam tromosti sustava, a to podrazumijeva manju veličinu kvanta.

Primjer 4: Raspoređivanje po RR algoritmu
Zadan je sustav od 5 procesa s vremenima pojavljivanja: $T_{p1..5}$ = (0, 2, 5, 7, 12) i vremenima trajanja procesa: $T_{t1...5}$ = (2, 5, 7, 2, 3). Primjenom RR algoritma s kvantom 2 skicirajte vremensku os zadanog sustava.

P1	P1	P2	P2	P2	P2	P3	P3	P2	P4	P4	P3	P3	P5	P5	P3	P3	P5	P3

2.2.2. *Priority Scheduling*

RR algoritam čini implicitnu pretpostavku da su svi procesi jednako važni. Međutim, u stvarnosti to najčešće nije tako, nego određeni procesi imaju veći prioritet. Uvrštavanje

prioriteta u RR algoritam čini novi algoritam - *Priority Scheduling* (PS). Osnovna ideja je da je svakom procesu dodijeljen prioritet i procesi najvećeg prioriteta se izvršavaju odmah pri dolasku. Ukoliko više procesa jednakih prioriteta čeka na izvršavanje, oni se raspoređuju prema RR algoritmu. U tom slučaju, vrlo je bitno napomenuti da kvant nije prekidan, što znači da dolazak procesa većeg prioriteta ne prekida izvođenje procesa kojemu je trenutno dodijeljen kvant. Moguće je i raspoređivanje procesa jednakih prioriteta prema nekim drugim algoritmima (npr. FCFS).

Primjer 5: Raspoređivanje po PS algoritmu
Zadan je sustav od 5 procesa s vremenima pojavljivanja: $T_{p1..5}$ = (0, 2, 5, 7, 12), vremenima trajanja procesa: $T_{t1...5}$ = (2, 5, 7, 2, 3) i prioritetima: $P_{PR1...5}$ = (1, 1, 2, 3, 2), gdje je 1 najviša razina prvenstva, a 4 najniža. Primjenom PS algoritma skicirajte vremensku os zadanog sustava. Po pojedinim razinama se vrši RR razdioba sa kvantom 1.

P1	P1	P2	P2	P2	P2	P2	P3	P3	P3	P3	P3	P5	P3	P5	P3	P5	P4	P4

2.3. Analiza vremenskih svojstva računalnog sustava

Upravitelji velikih računalnih centara unutar kojih se izvodi mnoštvo serijskih poslova obično promatraju tri metrike kako bi stekli uvid u performanse i ponašanje njihovog sustava: propusnost (engl. *throughput*), vrijeme obrade (engl. *turnaround time*) te iskorištenje procesora (engl. *CPU utilization*). **Propusnost** sustava se definira brojem poslova kojeg taj sustav izvrši u nekoj jedinici vremena (npr. satu). **Vrijeme obrade** procesa označava vrijeme koje je proteklo od trenutka njegova pojavljivanja u sustavu sve do trenutka kada je završio s izvođenjem. Algoritam raspoređivanja koji nastoji maksimizirati propusnost ne mora nužno minimizirati vrijeme obrade. Primjerice, ako sustav ima zadaću rasporediti kombinaciju vrlo dugih i vrlo kratkih poslova, raspoređivač koji uvijek pokreće samo kratke poslove, a nikada duge, može ostvariti visoku propusnost sustava nauštrb velikog vremena obrade za duge poslove.

Kod interaktivnih sustava, metrike za evaluaciju performansi i ponašanja su nešto drugačije. Najvažnija metrika je **vrijeme odziva** (engl. *response time*) koje predstavlja vrijeme koje je proteklo od početka do kraja izvođenja procesa. Prilikom raspoređivanja interaktivnih procesa, jedan od glavnih ciljeva je minimizirati vrijeme odziva. Primjerice, korisnik osobnog računala vrlo vjerojatno smatra pokret miša ili klik tipkovnice procesima većeg prioriteta od nekog pozadinskog procesa (npr. provjera e-maila na serveru). Stoga je cilj minimizirati vrijeme odziva za taj interaktivni proces, kako bi sustav pružio bolje korisničko iskustvo.

Prilikom kreiranje politike raspoređivanja procesa nekog sustava, bitno je razumjeti postupke analize dinamičkog ponašanja takvog sustava kako bi se on mogao prikladno evaluirati. Analizu dinamičkog ponašanja računalnog sustava moguće je provesti na tri načina:

- Simulacijom
- Praćenjem stvarnog sustava
- Korištenjem modela

U ovoj laboratorijskoj vježbi, sažeto ćemo prikazati načine analize računalnog sustava korištenjem modela. Pri tome, računalni sustav možemo smatrati **determinističkim** ili **nedeterminističkim**. U determinističkom sustavu, svi događaji su poznati ili predvidljivi. Neki proces u sustavu se pojavljuje u trenutku t_d, a iz njega odlazi u trenutku t_n. Vrijeme obrade tog procesa u sustavu je:

$$T = t_n - t_d$$

Pri dolasku novog procesa u sustav, on može odmah doći na red ili čeka u redu ako je poslužitelj zauzet. Vrijeme čekanja u redu se označava sa T_r, a vrijeme odziva sa T_p. Za vrijeme obrade također vrijedi i:

$$T = T_r + T_P$$

Kod nedeterminističkih sustava, analiza ponašanja se temelji na pretpostavkama o dolascima i trajanju procesa. Pri tome se te vrijednosti najčešće modeliraju pomoću funkcija razdiobe (Poissonova, eksponencijalna, …).

2.4. Implementacija algoritma raspoređivanja u programskom jeziku C

U ovom poglavlju implementirati ćemo jedan od prikazanih algoritama raspoređivanja u programskom jeziku C. Odabrani algoritam je *Shortest Remaining Time Next* (SRTN). Iako se svi algoritmi razlikuju u logici, većina koristi jednake strukture podataka te ispisuje jednak vremenski dijagram.

2.4.1. Strukture podataka

Algoritmi za raspoređivanje prvo primaju podatke o procesima za izvođenje, a zatim primjenjuju logiku raspoređivanja. Od ulaznih podataka, potrebno je pohraniti sljedeće za svaki od *n* procesa:

- Vrijeme pojavljivanja (engl. *arrival time*)
- Vrijeme izvršavanja (engl. *burst time*)

Izlaz svakog algoritma je u obliku vremenskog dijagrama koji ispisuje sljedeće podatke:

- Vrijeme čekanja (engl. *waiting time*)
- Vrijeme obrade (engl. *turnaround time*)
- Trenutak završetka (engl. *completion time*)

Primjer 6: Strukture podataka za algoritam SRTN

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int n, i, time;
 printf("Unesite broj procesa:\n");
 scanf("%d", &n);

 int *arrival_times = (int*) calloc (n, sizeof(int));
 int *burst_times = (int*) calloc (n, sizeof(int));
 int *waiting = (int*) calloc (n, sizeof(int));
 int *turnaround = (int*) calloc (n, sizeof(int));
 int *completion = (int*) calloc (n, sizeof(int));


 for (i =0; i<n; i++)
  scanf("%d", &arrival_times[i]);

 for (i =0; i<n; i++)
  scanf("%d", &burst_times[i]);
```

2.4.2. Vremenski dijagram

Vremenski dijagram prikazan u Primjeru 7, nije klasična vremenska crta kao u prethodnim primjerima. Za svaki proces se ispisuju njegov pid, ulazni podaci te izračunati podaci o vremenima raspoređivanja. Slika 1 prikazuje izgled ovakvog oblika vremenskog dijagrama.

```
Unesite broj procesa:
pid      burst   arrival    waiting     turnaround  completion
 1         5         1          2           7           8
 2         6         3          5           11          14
 3         3         0          0           3           3
 Total waiting time: 7.000000
Total turnaround time: 21.000000
```

Slika 1. Vremenski dijagram algoritma SRTN

Primjer 7: Vremenski dijagram za algoritam SRTN

```
 double waiting_time = 0, turnaround_time=0;

 printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");
 for (i=0; i<n; i++) {
  printf("\n %d \t   %d \t %d\t\t%d
\t\t%d\t\t%d",i+1,burst_times[i],arrival_times[i],waiting[i],turnaround[i],completion[i]);

  waiting_time = waiting_time + waiting[i];
  turnaround_time = turnaround_time + turnaround[i];
 }
 printf("\n Total waiting time: %lf\n", waiting_time);
 printf("Total turnaround time: %lf\n", turnaround_time);

 printf("Average waiting time: %lf\n", waiting_time/n);
 printf("Average turnaround time: %lf\n", turnaround_time/n);
```

2.4.3. Logika algoritma

Izvođenje algoritma SRTN prikazano je u primjeru 3. Algoritam raspoređuje proces s najkraćim preostalim vremenom izvršavanja. U implementaciji prikazanoj u primjeru 8, stvorena je *for* petlja koja inkrementira brojač vremena (*time*), sve dok se ne rasporede svi procesi. Prije izvođenja procesa, traži se proces s najkraćim preostalim vremenom izvršavanja, a njegov indeks se pohranjuje u varijablu *min*. Nakon što je takav proces nađen, smanjuje se njegovo preostalo vrijeme izvršavanja. Ukoliko je ono nakon smanjenja jednako 0, povećava se brojač raspoređenih procesa (*count*), te se računaju podaci za vremenski dijagram o izvršenom procesu. Postupak se ponavlja sve dok brojač raspoređenih procesa ne postane jednak broju procesa.

Primjer 8: Logika algoritma SRTN

```
 int count = 0; // Broj izvršenih procesa

 //Kopiranje burst_times u novo polje remaining
 int *remaining = (int*) calloc (n, sizeof(int));
 for (i=0; i<n; i++)
  remaining[i] = burst_times[i];

 // Logika - Traženje procesa s najkraćim preostalim vremenom izvršavanja
 for (time=0; count!=n; time++) {

  int min = 0; // Pretpostavimo da je to prvi proces
  // Ako je prvi proces već izvršen (remaining ==0), tražimo sljedeći koji još nije izvršen
  for (i=0;i<n;i++){
   if (remaining[i] !=0) {
    min = i;
    break;
   }
  }

  // Tražimo proces s najkraćim preostalim vremenom izvršavanja, a uspoređujemo s
procesom na indexu min
  for(i=0; i<n; i++) {
    // Ako neki drugi proces, koji je već stigao na red, ima kraće vrijeme izvršavanja (a
veće od 0), onda je on idući na redu (min)
    if (arrival_times[i] <= time && remaining[i] <= remaining[min]  && remaining[i]>0)
     min = i;
   }

// Nađeni proces se izvršava
  remaining[min] = remaining[min] - 1;

  // Provjeravamo je li proces izvršen do kraja i povecavamo count
  if (remaining[min] == 0){
   count++;
   int end = time +1; // Vrijeme završetka tog procesa
   completion[min] = end;
   waiting[min] = end - arrival_times[min] - burst_times[min]; // Vrijeme čekanja tog
procesa
   turnaround[min] = end - arrival_times[min];  //Vrijeme proteklo od pojavljivanja do
kraja izvršavanja procesa
  }
 }
```

3. ZADACI ZA SAMOSTALNI RAD

Pristupiti poslužitelju linux.etfos.hr pomoću Putty SSH klijenta. Sljedeće zadatke riješite koristeći uređivač teksta nano, sintaksu programskog jezika C i Bash ljusku operacijskog sustava Linux. Za kompajliranje programa napisanih u C jeziku koristite GCC kompajler koji je već instaliran na poslužitelju. Za svaki zadatak kopirajte programski kod, naredbe ljuske i dobiveni izlaz u Zadaću. Ako je potrebno, napišite dodatno pojašnjenje Vašeg rješenja.

1. Napišite program koji simulira *First-Come, First-Served* (FCFS) algoritam za raspoređivanje.
2. Napišite program koji simulira *Shortest Job First* (SJF) algoritam za raspoređivanje.
3. Napišite program koji simulira *Priority Scheduling* (PS) algoritam za raspoređivanje. Pretpostavite da se procesi jednakog prioriteta raspoređuju prema FCFS algoritmu koji je već implementiran u prvom zadatku.

3.1. Zadaci za vježbu

Ove zadatke nije potrebno riješiti tijekom laboratorijske vježbe. Oni služe isključivo za vježbanje za pismeni ispit, a svoja rješenja možete provjeriti na konzultacijama.

1. Napišite program koji simulira *Shortest Remaining Time Next* (SRTN) algoritam za raspoređivanje.
2. Napišite program koji simulira *Round Robin* (RR) algoritam za raspoređivanje.
3. Zadan je sustav od 8 procesa s vremenima pojavljivanja: $Tp_{1..8}$ = (0, 2, 3, 5, 7, 9, 12, 15), vremenima trajanja procesa: $Tt_{1..8}$ = (2, 5, 3, 7, 4, 2, 3, 3) i prioritetima: $P_{P1..8}$ = (1, 2, 1, 3, 2, 1, 3, 2), gdje je 1 najviša razina prvenstva, a 4 najniža. Primjenom PS algoritma skicirajte vremensku os zadanog sustava. Po pojedinim razinama se vrši RR razdioba sa kvantom 2.
4. Zadan je sustav od 8 procesa s vremenima pojavljivanja: $Tp_{1..8}$ = (0, 2, 3, 5, 7, 9, 12, 15), vremenima trajanja procesa: $Tt_{1..8}$ = (2, 5, 3, 7, 4, 2, 3, 3) i prioritetima: $P_{P1..8}$ = (1, 2, 1, 3, 2, 1, 3, 2), gdje je 1 najviša razina prvenstva, a 4 najniža. Primjenom PS algoritma skicirajte vremensku os zadanog sustava. Procesi istog prioriteta se raspoređuju pomoću SRTN algoritma.
5. U primjerima 1-5 te zadacima za vježbu 3 i 4, analizirajte dinamičko ponašanje sustava tako da odredite:
    a. propusnost sustava po 10 vremenskih jedinica
    b. prosječno vrijeme obrade procesa u sustavu
    c. prosječno vrijeme odziva procesa u sustavu
6. S obzirom na rezultate iz zadatka 5, usporedite i komentirajte performanse svih 5 razmatranih algoritama. Koji od algoritama su prikladniji za interaktivne, a koji za *Batch* procese?
7. Za primjere 1-5 te zadatke za vježbu 3 i 4, skicirajte graf tijeka prosječnog vremena odziva po jedinici vremena te graf tijeka prosječnog vremena po jedinici vremena.

4. LITERATURA

[1] Stallings, W., 2012. Operating systems: internals and design principles. Boston: Prentice Hall,.

[2] Tanenbaum, A.S. and Bos, H., 2015. Modern operating systems. Pearson.

[3] Bovet, D.P. and Cesati, M., 2005. Understanding the Linux Kernel: from I/O ports to process management. " O'Reilly Media, Inc.".

[4] Love, R., 2010. Linux kernel development. Pearson Education.

[5] Love, R., 2013. Linux system programming: talking directly to the kernel and C library. " O'Reilly Media, Inc.".

[6] Jelenković L., 2019. Operacijski sustavi: Interni material za predavanja iz predmeta. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva (http://www.zemris.fer.hr/~leonardo/os/fer/_OS-skripta.pdf)

[7] http://programmingearth.com/post.php?pageid=69&title=C%20program%20for%20shortest%20remaining%20time%20first%20scheduling%20algorithm.