Deadlock - rešenja

DL1.

Sistem je u bezbednom stanju - sekvenca P1, P0, P2 doveće do zadovoljenja potreba svih procesa.

Proces P1 najpre uzima još dve instance resursa, a zatim ih vraća:

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	5	10	5	1
P1	4	4	0	
P2	2	9	7	

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	5	10	5	5
P1	završio aktivnosti			
P2	2	9	7	

Proces P0, zatim, uzima još pet instanci resursa, a potom ih vraća:

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	10	10	0	0
P1	završio aktivnosti			
P2	2	9	7	

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	završio aktivnosti			10
P1	završio aktivnosti			
P2	2	9	7	

Na kraju, proces P2 uzima još sedam instanci resursa:

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	završio aktivnosti			3
P1	završio aktivnosti			
P2	9	9	7	

Proces	***Allocation***	***Max***	***Need***	***Available***
P0	završio aktivnosti			12
P1	završio aktivnosti			
P2	završio aktivnosti			

DL2.

Sistem nije u bezbednom stanju. Na primer, za sekvencu P1, P0, P2, proces P1 bi zadovoljio svoje potrebe i vratio resurse, ali se sistem može dovesti u zastoj ukoliko P0 zatraži maksimalan broj instanci resursa.

DL3.

(a) Najpre se određuje matrica potreba - *need* (svaki element matrice *need* se računa kao razlika odgovarajućih elemenata matrica *max* i *allocation*):

	Allocation			*Max*			*Need*			*Available*		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	3	3	2
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Zatim se primenom bankarskog algoritma pronalazi sekvenca P1, P3, P4, P2, P0 koja dokazuje da je sistem u stabilnom stanju [1].

(b) Proces P1 izdaje zahtev za dodelom reursa *request*=(1,0,2). Primenićemo algortiam koji razrešava zahteve za dodelom resursa.

Najpre se proverava da li je *request* ≤ *available*. Kako je (1,0,2) ≤ (3,3,2), uslov je ispunjen, pa se zatim obavlja kvazi-dodela resursa na osnovu zahteva.

	Allocation			*Max*			*Need*			*Available*		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	3	0	2	3	2	2	0	2	0			
P2	3	0	2	9	0	2	6	0	0	2	3	0
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Zatim se primenom bankarskog algoritma pronalazi sekvenca P1, P3, P4, P0, P2 koja zadovoljava uslove stabilnosti. To znači da će sistem ispuniti zahtev P1(1,0,2)

(c) Sistem neće odobriti zahtev procesa P4 za dodelu resursa: *request*=(3,3,0), jer je zahtev veći od raspoloživih resursa.

(d) Sistem neće odobriti zahtev procesa P0 za dodelu resursa: *request*=(1,2,2), jer se posle obavljene kvazi-dodele, sistem ne nalazi u stabilnom stanju.

	Allocation			*Max*			*Need*			*Available*		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	1	3	2	7	5	3	6	2	1			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	2	1	0
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

(e) Sistem će odobriti zahtev procesa P4 za dodelu resursa: *request*=(1,1,0), jer je *request* ≤ *available*, tj. (1,1,0) ≤ (3,3,2), a sistem se posle obavljene kvazi-dodele, nalazi u stabilnom stanju (sekvenca P3, P4, P1, P2, P0).

1 Uvek se polazi od procesa sa manjim zahtevima, a kasnije se rešavaju problemi najzahtevnijih procesa..

	Allocation			*Max*			*Need*			*Available*		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	2	2	2
P3	2	1	1	2	2	2	0	1	1			
P4	1	1	2	4	3	3	3	2	1			

DL4.

(a) Svaki element matrice *need* se računa kao razlika odgovarajućih elemenata matrica *max* i *allocation*:

	Allocation				*Max*				*Need*			
Proces	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	0	0	0	1	7	5	0	0	7	5	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

b) Sistem je u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:

- P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) ⇒ *available*=(1,5,3,2),
- P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) ⇒ *available*=(1,11,6,4),
- P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) ⇒ *available*=(2,14,11,8),
- P1 uzima (0,7,5,0) a zatim vraća (1,7,5,0) ⇒ *available*=(3,14,11,8),
- P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) ⇒ *available*=(3,14,12,12).

c) Najpre se proverava da li je *request* ≤ *available*. Kako je (0,4,2,0) ≤ (1,5,2,0), uslov je ispunjen, pa se zatim obavlja kvazi-dodela resursa na osnovu zahteva. Procesu P1 dodeljujemo (0,4,2,0):

	Allocation				*Max*				*Need*			
Proces	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	4	2	0	1	7	5	0	0	3	3	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

Posle dodele resursa, slobodna je 1 instanca resursa A i 1 instanca resursa B, odnosno *available*=(1,1,0,0).

Sistem je u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:

- P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) ⇒ *available*=(1,1,1,2),
- P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) ⇒ *available*=(2,4,6,6),
- P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) ⇒ *available*=(2,10,9,8),

- P1 uzima (0,3,3,0) a zatim vraća (1,7,5,0) ⇒ *available*=(3,14,11,8),
- P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) ⇒ *available*=(3,14,12,12).

Sistem odobrava zahtev za dodelom resursa.

DL5.

(a) Sistem nije u stanju zastoja. Primenom algoritma za detekciju zastoja može se dokazati da postoji sekvenca P0, P2, P3, P1, P4 nakon koje bi svi procesi završili svoje aktivnosti. Interesantno je da sistem u trenutku t_0 nema raspoloživih resursa, međutim u sistemu postoje dva procesa koji ne traže ni jedan dodatni resurs (procesi P0 i P2). Oni mogu da obave svoje aktivnosti i vrate resurse, koji se zatim mogu dodeliti drugim procesima.

- Kada P0 završi svoje aktivnosti, oslobodiće 1 instancu resursa B. Posle toga je *availible*=(0,1,0).
- Kada P2 završi svoje aktivnosti, oslobodiće po 3 instance resursa A i C. Nakon toga je *availible*=(3,1,3).
- Proces P3 je najmanje zahtevan, tako da mu sistem može dodeliti jednu instancu resursa A. Kada P3 završi svoje aktivnosti, on vraća (3,1,1) instanci resursa sistemu, pa je *availible*=(5,2,4).
- Zatim P1 uzima (2,0,2) i vraća (4,0,2) ⇒ *available*=(7,2,4)
- Na kraju, P4 uzima (0,0,2) i vraća (0,0,4) ⇒ *available*=(7,2,6)

Kada svi procesi završe svoje aktivnosti, na sistemu ostaje 7 instanci resursa A, dve instance resursa B i 6 instanci resursa C.

(b) Sistem je u stanju zastoja. U trenutku t_0, na sistemu nema raspoloživih resursa, međutim proces P0 ne traži ni jedan dodatni resurs. On može da obavi svoje aktivnosti i oslobodi jednu instancu resursa B. Posle toga je *availible*=(0,1,0). Međutim, dalje ni jedan proces ne može da zadovolji svoje potrebe za resursima.

DL6.

Sistem nije u stanju zastoja, jer postoji sekvenca P0, P2, P3, P1, P4 posle koje će svi procesi završiti svoje aktivnosti.

- P0 uzima (1,0,0) a zatim vraća (2,1,0) ⇒ *available*=(2,1,1)
- P2 uzima (0,1,0) a zatim vraća (2,1,2) ⇒ *available*=(4,1,3)
- P3 uzima (1,0,0) a zatim vraća (2,1,1) ⇒ *available*=(5,2,4)
- P1 uzima (2,0,0) a zatim vraća (4,0,2) ⇒ *available*=(7,2,6)
- P4 uzima (0,0,2) a zatim vraća (0,0,4) ⇒ *available*=(7,2,8)

Memorija - rešenja

M1.

(a) overlay 1: GP+A = 60K+20K = 80K

overlay 2: GP+B+D+E= 60+50+40+70 = 220K

overlay 3: GP+C+F = 60+40+70 = 170K

(b) max (80, 220, 170) = 220K

(c) GP+A+B+C+D+E+F = 60+20+50+40+70+40+70 = 350K

M2.

(a) *First fit*:

- 212K se smešta u particiju veličine 500K, pri čemu ostaje prazna particija veličine 500K-212K=288K. Posle dodele, sledeće particije su prazne (redom): 100K, 288K, 200K, 300K, 600K. Algoritam je obavio dva poređenja.
- 417K se smešta u particiju veličine 600K, pri čemu ostaje prazna particija veličine 600K-417K=183K. Posle dodele, sledeće particije su prazne (redom): 100K, 288K, 200K, 300K, 183K. Algoritam je obavio 5 poređenja.
- 112K se smešta u particiju veličine 288K, pri čemu ostaje prazna particija veličine 288K-112K=176K. Posle dodele, sledeće particije su prazne (redom): 100K, 176K, 200K, 300K, 600K. Algoritam je obavio dva poređenja.
- Procesu koji zahteva 426K ne može se dodeliti memorija zbog eksterne fragmentacije. Algoritam je obavio 5 poređenja.

  Proces koji zahteva 426K ne može odmah da se izvrši i mora da sačeka da neki drugi proces oslobodi memoriju. Obavljeno je ukupno 14 poređenja između veličine memorije koju proces zahteva i slobodnih particija.

(b) *Best fit*:

- 212K se smešta u particiju veličine 300K, pri čemu ostaje prazna particija veličine 300K-212K=88K. Posle dodele, sledeće particije su prazne (redom): 100K, 500K, 200K, 88K, 600K.
- 417K se smešta u particiju veličine 500K, pri čemu ostaje prazna particija veličine 500K-417K=83K. Posle dodele, sledeće particije su prazne (redom): 100K, 83K, 200K, 88K, 600K. Algoritam je obavio 5 poređenja.
- 112K se smešta u particiju veličine 200K, pri čemu ostaje prazna particija veličine 200K-112K=88K. Posle dodele, sledeće particije su prazne (redom): 100K, 83K, 88K, 88K, 600K. Algoritam je obavio 5 poređenja.
- 426K se smešta u particiju veličine 600K, pri čemu ostaje prazna particija veličine 600K-426K=174K. Posle dodele, sledeće particije su prazne (redom): 100K, 83K, 88K, 88K, 174K. Algoritam je obavio 5 poređenja.

  Svaki proces dobija zahtevanu memoriju. Obavljeno je ukupno 20 poređenja između veličine memorije koju proces zahteva i slobodnih particija.

(c) *Worst fit*

- 212K se smešta u particiju veličine 600K, pri čemu ostaje prazna particija veličine 600K-212K=388K. Posle dodele, sledeće particije su prazne (redom): 100K, 500K, 200K, 300K, 388K. Algoritam je obavio 5 poređenja.
- 417K se smešta u particiju veličine 500K, pri čemu ostaje prazna particija veličine 500K-417K=83K. Posle dodele, sledeće particije su prazne (redom): 100K, 83K, 200K, 300K, 388K. Algoritam je obavio 5 poređenja.
- 112K se smešta u particiju veličine 388K, pri čemu ostaje prazna particija veličine 388K-112K=278K. Posle dodele, sledeće particije su prazne (redom): 100K, 83K, 200K, 300K, 278K. Algoritam je obavio 5 poređenja.
- Procesu koji zahteva 426K ne može se dodeliti memorija zbog eksterne fragmentacije. Algoritam je obavio 5 poređenja.

  Proces koji zahteva 426K ne može odmah da se izvrši i mora da sačeka da neki drugi proces oslobodi memoriju. Obavljeno je ukupno 20 poređenja između veličine memorije koju proces zahteva i slobodnih particija.

(d) U ovom slučaju *Best fit* algoritam obezbeđuje najbolje iskorišćenje memorije.

M.3.

broj stranice = ceo broj količnika logičke adrese i veličine stranice

ofset = ostatak pri deljenju logičke adrese i veličine stranice

(a) Logička adresa 251

broj stranice=[251/2048]=0 ⇒ okvir=1, ofset=251%2048=251, fizička adresa=1x2048+251=2299

(b) Logička adresa 3129

br.stranice=[3129/2048]=1 ⇒ okvir=4, ofset=3129%2048=1081, fizička adresa=4x2048+1081=9273

(c) Logička adresa 10000

br.stranice=[10000/2048]=4 ⇒ okvir=12, ofset=10000%2048=1808, fiz.adr.=12x2048+1808=26384

(d) Logička adresa 6066

br. stranice=[6066/2048]=2 ⇒ okvir=3, ofset=6066%2048=1970, fizička adresa=3x2048+1970=8114

M.4.

(a) 219 + 430 = 649

(b) 2300 + 10 = 2310

(c) illegalna referenca (premašena vrednost limit registra)

(d) 1327 + 400 = 1727

(e) illegalna referenca (premašena vrednost limit registra)