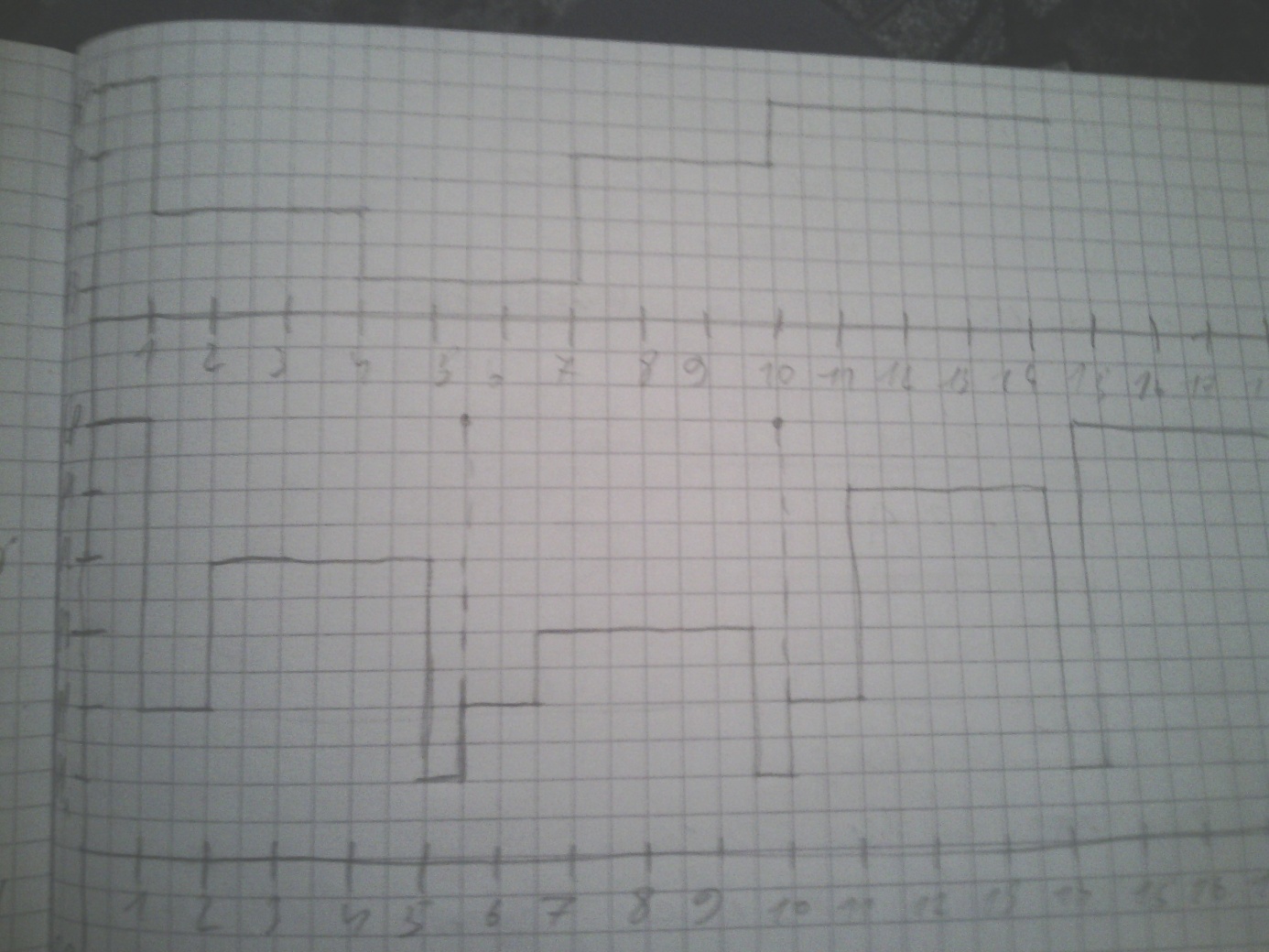
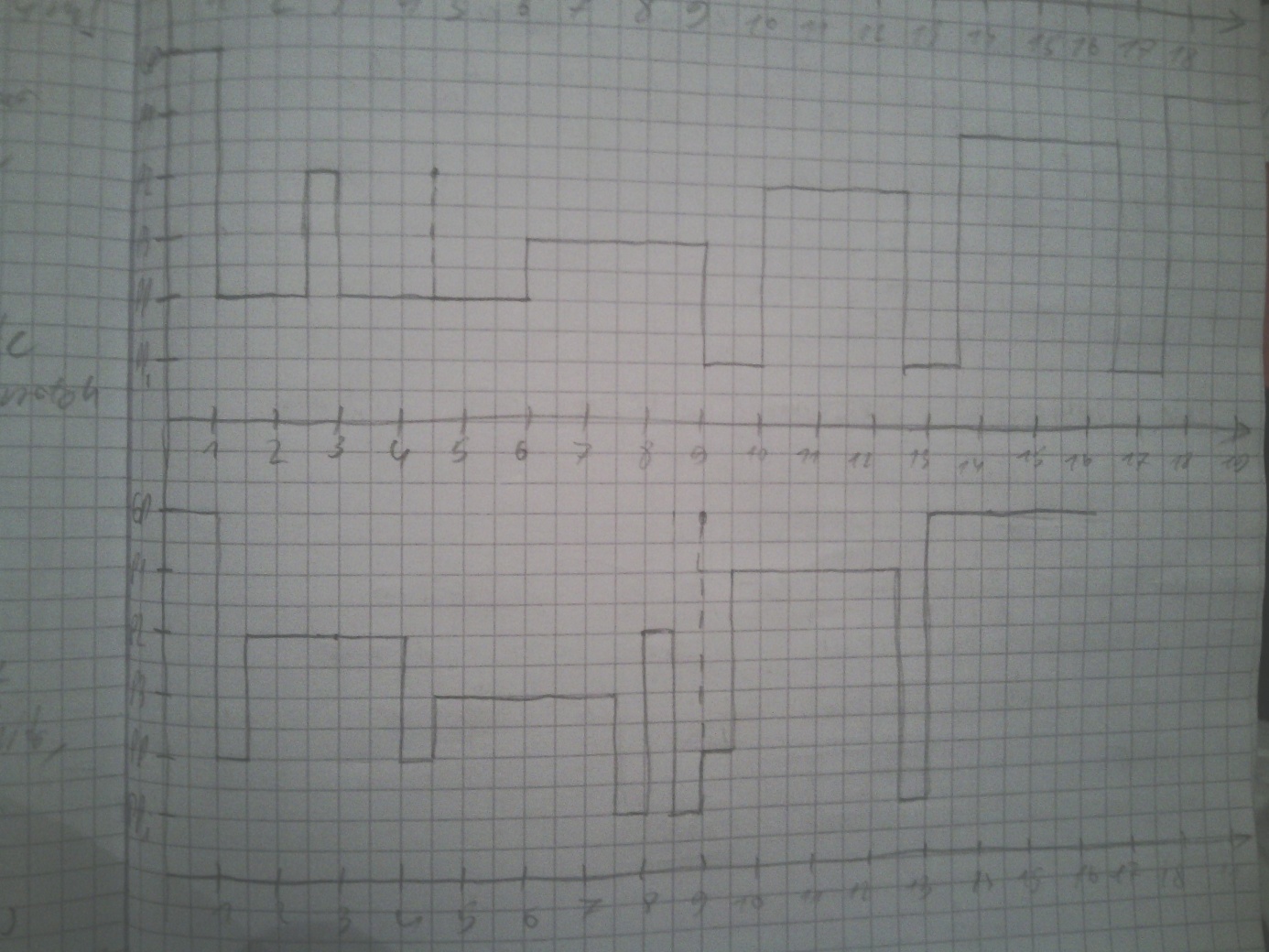
Zadatak 3.1

U nekom sustavu javljaju se prekidi. P1 u 3. ms, P2 u 1. ms, P3 u 4. ms. Prioritet prekida određen je brojem (P3 ima najveći prioritet). Obrada svakog prekida traje 3 ms. Grafički prikazati aktivnosti procesa u glavnom programu (GP), procedurama za obradu prekida Pi te procedurama za prihvat prekida (PP) i povratak iz prekida (PPi).  
 a) U idealnom slučaju  
 b) Bez sklopa za prihvat prekida, obrada uz zabranjeno prekidanje, ako je zadano uz vremena PP = 1 ms (potraga za izvorom prekida, pohranu konteksta) i PPi = 0.5 ms (obnova konteksta)  
 c) Bez sklopa za prihvat prekida uz vrijeme PP = 1.5 ms te PPi = 1 ms.  
 d) Sa sklopom za prihvat prekida, PP = 0.5 ms (izvodi se samo za pohranu konteksta), PPi = 0.5 ms (obnova konteksta)

*Rješenje*

Rješenja su grafovi, slični onima u prezentaciji 03\_UI. Na x osi je vrijeme, na y osi su moguće aktivnosti procesa. Princip crtanja pojedinog grafa je objašnjen dolje, uz priložene slike pojedinih rješenja.  
  
  
Slika 1 - Rješenje slučajeva 'a' i 'b' (oznake na y-osi se baš ne vide, pa ću ih ovdje zapisati odozgo prema dolje - GP, P1, P2, P3, PP te PPi, s time da prvi graf nema PP i PPi jer nisu potrebni)  
  
  
a) Nema kućanskih poslova, prekidi se međusobno ne mogu prekidati, eventualno pripaziti na prioritete (ako na obradu čeka više prekida, prvi se obrađuje onaj većeg prioriteta).  
  
b) Kada se dogodi prekid, skačemo iz GP u PP, iz PP u Px (x je razina prekida), iz Px u PPi, s tim da prekidi koji se dogode za vrijeme obrade, prihvata ili povratka iz prekida bivaju prihvaćeni tek nakon što završi povratak iz prekida koji se trenutno obrađuje. Naznačiti težnju ka povratku u glavni program onda kada završi povratak iz prekida, a postoje prekidi koji još nisu prihvaćeni (to se dogodi u t = 5.5 ms i t = 10 ms). U tim trenutcima se, naime, glavni program postavi kao trenutno izvođeni, ali istog trenutka kreće u prihvat najprioritetnijeg prekida koji čeka na obradu.  
  
  
Slika 2 - Rješenje slučajeva 'c' i 'd'  
  
c) U ovom slučaju je dozvoljeno gniježđenje prekida (obrada trenutnog prekida može biti prekinuta). Ovdje je bitno naznačiti dvije stvari - prihvat (PP) i povratak iz prekida (PPi) ne mogu biti prekinuti. Nadalje, budući da nema sklopa za prihvat prekida, prekid se mora prihvatiti čim se dogodi, neovisno o njegovom prioritetu (i o prioritetu trenutnog prekida). Prekidna rutina, naime, mora odrediti hoće li se prekid staviti na čekanje ili pustiti na obradu, a ta odluka se mora donijeti čim prije.  
  
  
  
  
  
d) Sad kad imamo sklop za prihvat prekida, on djeluje kao neka vrsta "tajnika" - prekide većeg prioriteta od trenutnog će propustiti prema procesoru, a prekide manjeg prioriteta od trenutnog će ostaviti da čekaju sve dok ne završi obrada trenutnog. Zato se prihvat prekida događa neposredno prije početka obrade istog. Trenutak t = 9 ms je malo nezgodan, ali nije neobjašnjiv. Znamo da će sklop propustiti prema procesoru samo prekid većeg prioriteta od trenutnog. To znači da prekid P1 neće biti propušten u procesor sve dok sklop ne detektira da procesor kreće izvoditi GP (koji je najmanjeg mogućeg prioriteta). Upravo se zato u tom trenutku dogodi težnja ka povratku u GP.  
  
Dobar način provjere rješenja je pogledati izvršava li se PP jednaki broj puta koliko PPi. Ako ne, onda ste sigurno pogriješili.

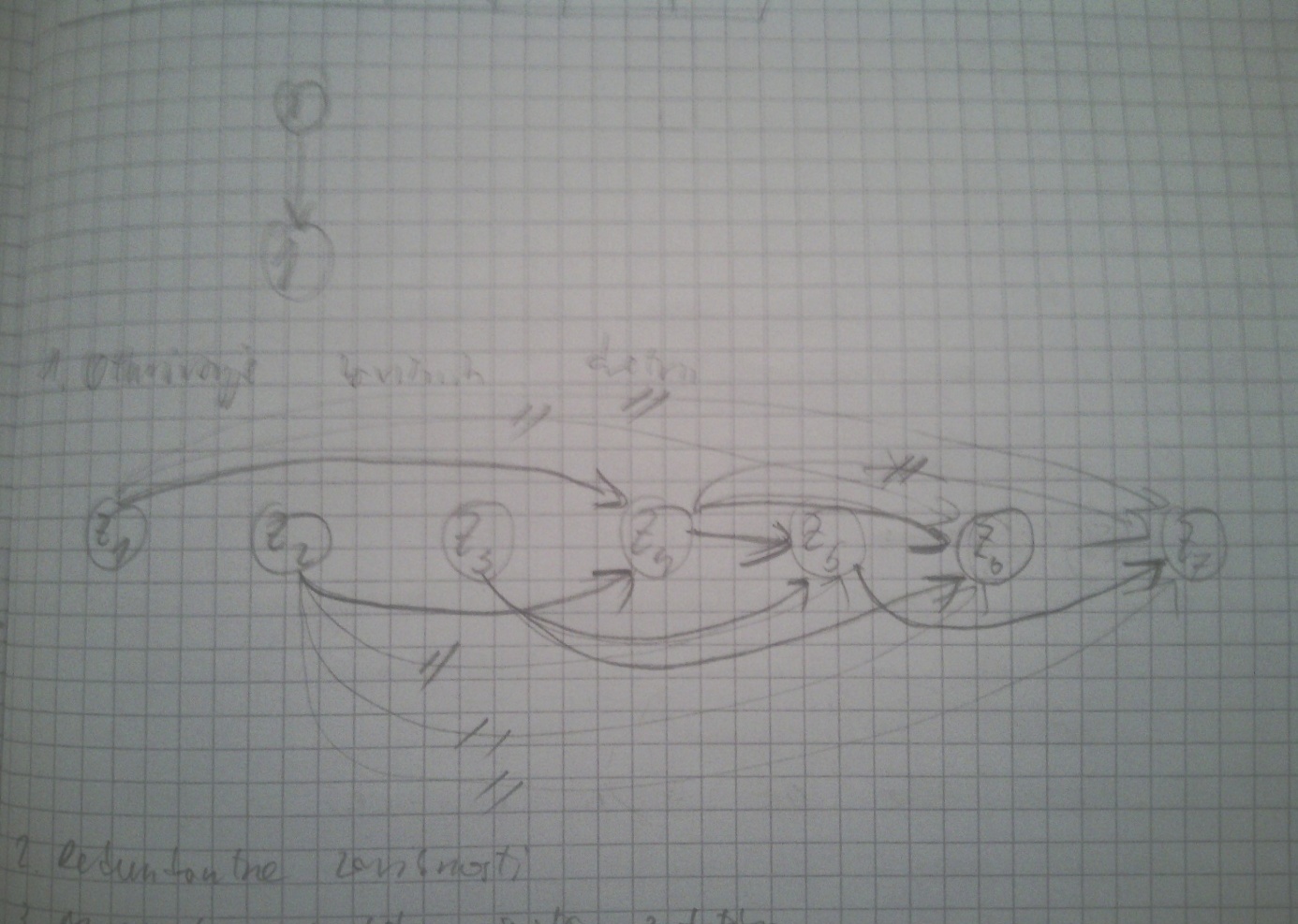
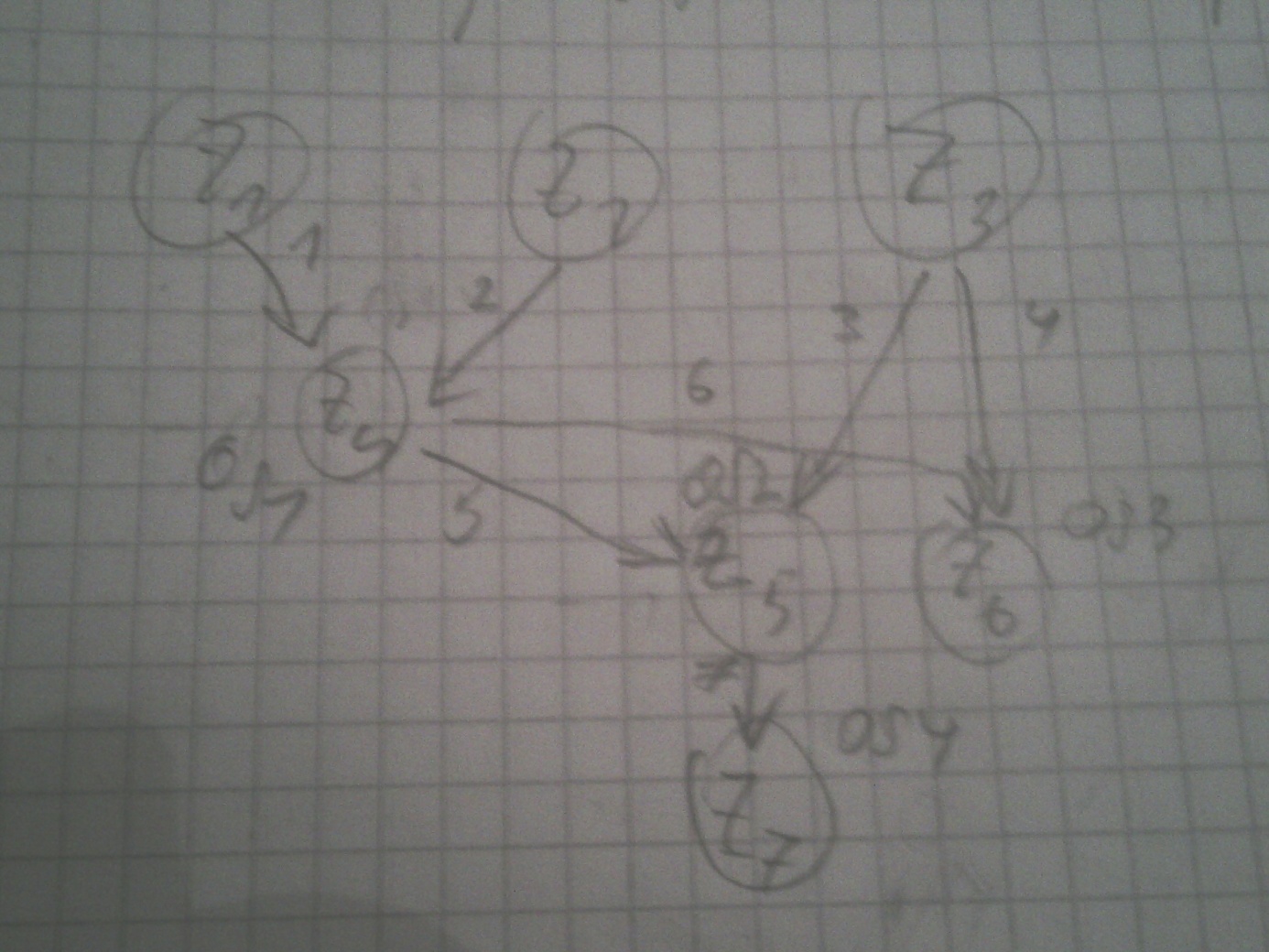
Zadatak 4.1

Dan je sustav zadataka Z1 --> Z2 --> Z3 --> Z4 --> Z5 --> Z6 --> Z7. Zadaci imaju domene i kodomene prema tablici:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Z1 | Z2 | Z3 | Z4 | Z5 | Z6 | Z7 |
| M1 | D | D | D | D |  | K |  |
| M2 | K |  |  | K |  | D | D |
| M3 |  | K |  | D | DK |  | K |
| M4 |  |  | K |  | D |  |  |
| M5 |  |  | K |  |  | K |  |
| M6 |  |  |  |  |  |  |  |

Nacrtati maksimalno paralelni sustav postavljenih zadataka.

*Rješenje*

Bitno je držati na umu da prva rečenica ovog zadatka zapravo daje ispravan redoslijed izvršavanja ovih zadataka. Dakle, ako se izvršavaju redom kojim su navedeni, tada sustav funkcionira. Ako se sjetimo uvjeta nezavisnosti zadataka, tada znamo da zadaci ne smiju istovremeno pisati u zajedničku memorijsku lokaciju, niti može jedan čitati iz nje dok drugi piše u nju. To nas dovodi do prvog koraka u rješavanju ovog zadatka: Pronalaženje međusobno zavisnih zadataka.  
  
 Ovaj korak se svodi na čitanje tablice, i to tako da uspoređujemo svaki zadatak sa onima desno od njega. U biti tražimo redak u kojemu dva zadatka koja uspoređujemo imaju kombinaciju K-D, D-K ili K-K. Tako na primjer Z1 i Z4 imaju K-K u M2, pa zaključujemo da se Z4 mora nužno izvoditi nakon Z1. Dalje Z1 i Z6 imaju D-K u M1 i K-D u M2 pa zaključujemo da Z6 mora doći nakon Z1 itd.  
  
Slijedeći korak u rješavanju ovog zadatka je eliminacija redundantnih zavisnosti.  
  
 Ako na crtežu imamo npr. put od Z1 kroz druge zadatke do Z7, tada nam je strelica iz Z1 u Z7 redundantna (vidi sliku dolje). Također nam je redundantna strelica iz Z1 u Z6 jer Z1 ima strelicu u Z4 koji pak ima strelicu u Z6. Ponavljanjem tog postupka za svaki zadatak dobijemo crtež koji izgleda otprilike ovako:  
  
Naš posljednji korak je crtanje maksimalno paralelnog sustava zadataka, koji izgleda otprilike ovako:  


Zadatak 4.2

Za ostvarenje sustava dretvi prema rješenju zadatka 4.1 koriste se semafori.  
 a) Koliko je potrebno binarnih, a koliko brojačkih za sinkronizaciju tog sustava?  
 b) Koje su početne vrijednosti semafora (BSEM i OSEM)?  
 c) Neka je Ti tekst (kod, slijed naredbi) zadatka i. Proširiti svaki zadatak Ti u Ti' s minimalnim brojem potrebnih procedura čekaj\_semafor, postavi\_semafor.

*Rješenje*

a) Minimalni broj potrebnih semafora za sinkronizaciju paralelnog sustava dretvi određuje se promatrajući sliku maksimalno paralelnog sustava dretvi na slijedeći način:  
broj\_BSEMa = broj\_strelica = 7  
broj\_OSEMa = broj\_dretvi\_u\_koje\_uvire\_bar\_jedna\_strelica = 4  
  
b) Početne vrijednosti svih BSEMa i OSEMa su 0.  
  
c) Promatrat ćemo dretve T2 i T4 kao neke ogledne primjere dretvi (T2 je jedna od dretvi koja nema strelicu koja uvire u nju, a T4 ima strelice koje uviru i strelice koje izviru).  
  
Dretva T2' { Dretva T2' {  
 T2 T2  
 postavi\_BSEM(2); postavi\_OSEM(1);  
} }  
  
Dretva T4' { Dretva T4' {  
 čekaj\_BSEM(1); čekaj\_OSEM(1);  
 čekaj\_BSEM(2); čekaj\_OSEM(1);  
 T4 T4  
 postavi\_BSEM(5); postavi\_OSEM(2);  
 postavi\_BSEM(6); postavi\_OSEM(3);  
} }  
  
Vidimo da dretve koje se prve moraju izvršavati ne čekaju nikakav semafor, jer se ništa ne mora izvršiti prije njih. Isto tako postoji razlika u broju semafora. Budući da za svaku strelicu postoji binarni semafor, onda dretva Ti' mora za svaku strelicu koja iz nje izvire postaviti odgovarajući semafor, a za svaku strelicu koja u nju uvire dočekati odgovarajući semafor. S druge strane, budući da za svaku dretvu u koju uviru strelice postoji jedan opći semafor, onda dretva Ti' mora dočekati svoj semafor onoliko puta koliko strelica uvire u nju, a mora postaviti odgovarajući OSEM za svaku strelicu koja izvire iz nje (iz Z4 idu strelice u Z5 i Z6, pa on postavlja njihove semafore).

Zadatak 4.3

Proces izvršava u glavnoj dretvi kod:  
 .  
 .  
 .  
 fork();  
 fork();  
 pthread\_create(...);  
 pthread\_create(...);  
 .  
 .  
 .  
Koliko će nakon izvršetka odsječka biti procesa, a koliko dretvi?  
  
*Rješenje*  
  
 Svaki proces se sastoji od minimalno jedne dretve, tako da na početku imamo jedan proces i pripadajuću glavnu dretvu. Taj proces izvrši prvi fork, rezultat čega je još jedan proces sa svojom glavnom dretvom. Sada oba procesa izvrše svaki svoj fork, što povećava ukupan broj procesa i dretvi na 4. Zatim, 4 procesa koja imamo ukupno svaki stvori dodatnu dretvu - pridodajemo 4 na broj dretvi - rezultat je 8 dretvi. Zatim svaki proces (broj procesa se nije promijenio, uzimamo da stvorene dretve ne pozivaju fork i pthread\_create) PONOVO izvršava naredbu pthread\_create, što na naših 8 ukupnih dretvi pridodaje 4, pa je rezultat 4 PROCESA I 12 DRETVI. Za bolju vizualizaciju pogledati tablicu dolje.

|  |  |  |
| --- | --- | --- |
| Naredba koja će se izvesti | Broj procesa koji ju izvodi | Broj dretvi u tom trenutku |
| fork(); | 1 | 1 |
| fork(); | 2 | 2 |
| pthread\_create(); | 4 | 4 |
| pthread\_create(); | 4 | 8 |
| - | 4 | 12 |

Zadatak 5.1

Sinkronizirati ulaznu, radnu i izlaznu dretvu pomoću 4 binarna semafora.  
  
*Rješenje*  
Početne vrijednosti: Ulazna\_D {  
BSEM[1].v = 1; dok je (1) {  
BSEM[2].v = 0; dohvati\_podatke; //NKO  
BSEM[3].v = 1; čekaj\_BSEM(1);  
BSEM[4].v = 0; pošalji\_podatke\_radnoj; //KO  
 postavi\_BSEM(2);  
 }  
 }  
Radna\_D { Izlazna\_D {  
 dok je (1) { dok je (1) {  
 čekaj\_BSEM(2); čekaj\_BSEM(4);   
 prihvati\_podatke; //KO primi\_podatke; //KO  
 postavi\_BSEM(1); postavi\_BSEM(3);   
 obrada\_podataka; //NKO ispiši\_rezultat; //NKO  
 čekaj\_BSEM(3); }  
 pošalji\_podatke\_izlaznoj; //KO }  
 postavi\_BSEM(4);  
 }  
}  
  
Čemu služi koji BSEM:  
1 - onemogućuje ulaznoj dretvi da šalje radnoj novi skup podataka dok je ona zauzeta prihvatom trenutnog skupa podataka  
2 - onemogućuje radnoj dretvi da krene prihvaćati podatke koje ulazna još nije završila slati  
3 - onemogućuje radnoj dretvi da šalje novi skup podataka izlaznoj dok izlazna prima trenutni skup podataka  
4 - onemogućuje izlaznoj dretvi da krene primati podatke koje radna još nije završila slati

Zadatak 5.2

Zadana je globalna varijabla A postavljena na vrijednost 0. Proces kojem A pripada vrti 10 dretvi, a svaka dretva radi isti zadatak:  
 Dretva {  
 za i = 1 do 1 000 000  
 A++;  
 }  
Koja je minimalna moguća vrijednost varijable A nakon završetka svih dretvi?  
 *Rješenje*Promatrajmo prvu dretvu, zadnju dretvu i ostalih 8. Kako nema sustava za sinkronizaciju dretvi, onda se može dogoditi da prva dretva pročita 0, poveća ju za 1 i zabloka prije nego stigne zapisati izračunat rezultat u memoriju (taj rezultat je u nekom registru, sjetimo se). Nakon toga se izvrše sve ostale dretve. Zatim, zadnja dretva krene raditi i izvrši se sve do zadnje iteracije. Sad prva dretva zapiše svoj 1 u memoriju. Zadnja ju pročita, poveća i zabloka. Prva se izvrši do kraja, a onda zadnja svoj rezultat (koji je 2) zapiše u memoriju i završi, pa je odgovor **2**. *Ovakav zadatak vjerojatno neće doći na ispit, ali može pomoći da shvatite kako razmišljati kod zadataka gdje treba otkriti grešku u načinu sinkronizacije.*

Zadatak 5.3

Riješiti problem potrošača i proizvođača sa minimalnim brojem brojačkih i binarnih semafora (OSEM i BSEM).  
  
*Rješenje*Proizvođač { Potrošač {  
 dok je (1) { dok je (1) {  
 proizvesti\_poruku(P); čekaj\_OSEM(PRAZAN);  
 čekaj\_OSEM(PUN); čekaj\_BSEM(2);  
 čekaj\_BSEM(1); R = MS[IZ];  
 MS[UL] = P; IZ = (IZ +1) % N;  
 UL = (UL+1) % N postavi\_BSEM(2);  
 postavi\_BSEM(1); postavi\_OSEM(PUN);  
 postavi\_OSEM(PRAZAN); potrošiti\_poruku(R);  
 } }  
} }  
  
početne vrijednosti:  
OSEM[PUN].v = N  
OSEM[PRAZAN].v = 0  
BSEM[1] = 1  
BSEM[2] = 1  
  
MS je međuspremnik, koji radi na principu cirkularnog reda (dva pokazivača IZ i UL, kad su jednaki onda je prazan, kad je IZ za jedno mjesto ispred UL, red je pun). OSEM PUN je neprolazan kad je međuspremnik pun. OSEM PRAZAN je neprolazan kad je međuspremnik prazan. Proizvođač ne smije upisivati u međuspremnik kad je on PUN, pa mora čekati upravo taj OSEM. Kada proizvođač upiše poruku u međuspremnik, on više nije prazan pa proizvođač postavlja upravo taj OSEM. Potrošač ne može čitati iz praznog međuspremnika, pa mora čekati OSEM PRAZAN. Kad potrošač pročita poruku iz međuspremnika, on mora postaviti OSEM PUN (on zapravo broji koliko mjesta je slobodno za pisanje).  
Binarni semafori se koriste za sinkronizaciju više potrošača/proizvođača. Potrošači imaju svoj BSEM ako ih je više od jednoga (nema smisla sinkronizirati ga sa samim sobom), kao i proizvođači. Znači, možemo imati minimalno 0 BSEM-a, maksimalno 2 BSEM-a.

Zadatak 5.4

U jednoprocesorskom računalu pokrenut je sustav dretvi D1, D2, D3 i D4 s prioritetima 1, 2, 3 i 4 respektivno. Najviši prioritet je 4. Svi zadaci koje obavljaju dretve su istog oblika (Dx). Red pripravnih dretvi i red semafora su prioritetni. Aktivna je dretva ona koja je prva u redu pripravnih dretvi (nema posebnog reda aktivnih dretvi). Prije pokretanja sustava dretvi semafor S je bio crven. Nakon nekog vremena sve dretve se nađu u redu semafora S. Ako se tada pozove procedura postavi\_BSEM(S), što će se ispisati na zaslonu?  
  
 Dretva Dx {  
 dok je (1) {  
 čekaj\_BSEM(S); // b - kraj  
 piši (Px); // a - poč  
 postavi\_BSEM(S); // a - kraj  
 piši (Zx); // b - poč  
 }  
 }  
  
*Rješenje*  
  
Podijelimo dretvu Dx na dio a i dio b kako je naznačeno gore. Uzet ćemo da izvršavanje dretve 4 počinje od oznake a-poč jer inače zadatak ne bi imao smisla (vratili bismo se nazad na prvi red tablice). Sada pratimo pravila funkcija čekaj\_BSEM i postavi\_BSEM te gledamo što će se ispisati (prema tablici). Valja naznačiti da se 29. (predzadnji) i 28. (jedan prije predzadnjeg) red u tablici ciklički ponavljaju.

|  |  |  |  |
| --- | --- | --- | --- |
| Red BSEM | V | Red PRIPRAVNE\_D | izlaz |
| 4 3 2 1 | 0 | - | - |
| 3 2 1 | 0 | 4a | P4 |
| 2 1 | 0 | 4b 3a | Z4 |
| 4 2 1 | 0 | 3a | P3 |
| 2 1 | 0 | 4a 3b | P4 |
| 1 | 0 | 4b 3b 2a | Z4 |
| 4 1 | 0 | 3b 2a | Z3 |
| 4 3 1 | 0 | 2a | P2 |
| 3 1 | 0 | 4a 2b | P4 |
| 1 | 0 | 4b 3a 2b | Z4 |
| 4 1 | 0 | 3a 2b | P3 |
| 1 | 0 | 4a 3b 2b | P4 |
| - | 0 | 4b 3b 2b 1a | Z4 |
| 4 | 0 | 3b 2b 1a | Z3 |
| 4 3 | 0 | 2b 1a | Z2 |
| 4 3 2 | 0 | 1a | P1 |
| 3 2 | 0 | 4a 1b | P4 |
| 2 | 0 | 4b 3a 1b | Z4 |
| 4 2 | 0 | 3a 1b | P3 |
| 2 | 0 | 4a 3b 1b | P4 |
| - | 0 | 4b 3b 2a 1b | Z4 |
| 4 | 0 | 3b 2a 1b | Z3 |
| 4 3 | 0 | 2a 1b | P2 |
| 3 | 0 | 4a 2b 1b | P4 |
| - | 0 | 4b 3a 2b 1b | Z4 |
| 4 | 0 | 3a 2b 1b | P3 |
| - | 0 | 4a 3b 2b 1b | P4 |
| - | 1 | 4b 3b 2b 1b | Z4 |
| - | 0 | 4a 3b 2b 1b | P4 |

Zadatak 5.5

U promatranom trenutku stanje sustava je slijedeće: dretva 1 je aktivna, dretve 2 i 3 su u redu BSEM[1], te dretve 4 i 5 su u redu pripravne dretve. Ako tada dretva 1 pozove j\_fju postavi\_BSEM(1), kako će izgledati struktura podataka jezgre nakon poziva? Svi redovi organizirani su po redu prispijeća (FIFO).  
  
*Rješenje*  
  
Aktivna\_D: 1  
Pripravne\_D: 4, 5  
BSEM[1]: 2,3  
 |  
 |  
 V  
Aktivna\_D: -  
Pripravne\_D: 4, 5, 1 //MEĐUKORAK 1  
BSEM[1]: 2, 3  
 |  
 |  
 V  
Aktivna\_D: -  
Pripravne\_D: 4, 5, 1, 2 //MEĐUKORAK 2  
BSEM[1]: 3  
 |  
 |  
 V  
Aktivna\_D: 4  
Pripravne\_D: 5, 1, 2  
BSEM[1]: 3  
  
Samo treba pogledati kako radi funkcija postavi\_bsem.

Zadatak 5.6

Modelirati vrtuljak s dva tipa dretvi: dretvama Posjetitelj (koje predstavljaju posjetitelje koji se žele voziti na vrtuljku) te dretvom Vrtuljak. Dretvama Posjetitelj se ne smije dozvoliti ukrcati na vrtuljak kada više nema praznih mjesta (koristiti varijablu BR\_MJESTA) i pokrenuti vrtuljak kada je pun. Za sinkronizaciju koristiti opće semafore i dodatne varijable.  
  
*Rješenje*

Dretva Posjetitelj { Dretva Vrtuljak {  
 čekaj\_OSEM(vrtuljak); dok je (1) {  
 sjedni; za i = 1 do BR\_MJESTA  
 postavi\_OSEM(sjeo); postavi\_OSEM(vrtuljak);  
 čekaj\_OSEM(kraj); za i = 1 do BR\_MJESTA  
 ustani; čekaj\_OSEM(sjeo);  
 postavi\_OSEM(sišao); pokreni\_vrtuljak;  
} zaustavi\_vrtuljak;  
 za i = 1 do BR\_MJESTA  
početne vrijednosti: postavi\_OSEM(kraj);  
BR\_MJESTA = zadano za i = 1 do BR\_MJESTA  
OSEM[vrtuljak].v = 0 čekaj\_OSEM(sišao);  
OSEM[sjeo].v = 0 }  
OSEM[kraj].v = 0  
OSEM[sišao].v = 0  
  
OSEM vrtuljak daje posjetiteljima do znanja da ima slobodnih mjesta na vrtuljku.  
OSEM sjeo pokazuje dretvi Vrtuljak da je posjetitelj završio sjedati na vrtuljak (ne može se početi vrtiti dok netko sjeda na njega).  
OSEM kraj daje posjetiteljima do znanja da se vrtuljak završio vrtiti (bilo bi jako nezgodno da netko krene ustajati sa vrtuljka dok se još vrti).  
OSEM sišao služi tome da vrtuljak počinje primati novu grupu posjetitelja tek nakon što su svi iz prošle grupe sišli.