Pisati čitko – nečitak odgovor ne donosi bodove.

1. [2] Neki klimatizacijski sustav može upravljati temperaturom i vlagom prostorije. Osmisliti upravljač zasnovan na neizrazitom upravljanju koji će kao ulaz dobivati: postavljene vrijednosti temperature (18°C) i vlage (65%) te očitane vrijednosti za isto. Upravljač na osnovu toga treba generirati naredbe za temperaturu i vlagu (dvije vrijednosti) koje se kreću od -2 do 2. Negativne vrijednosti predstavljaju način rada koji smanjuje traženo svojstvo (temperaturu ili vlagu) a pozitivne povećanje tih vrijednosti. Veća apsolutna vrijednost označava jači način rada. Pokazati rad sustava na primjeru ulaza t=22°C, v=40%.

2. Zadan je sustav zadataka koji se raspoređuje na dvoprocesorskom sustavu. Grafički prikazati izvođenje sustava, počevši od kritičnog slučaja u t=0 ms do t=20 ms ako se koristi:

   a) [2] postupak prema krajnjem trenutku završetka, uz sekundarni kriterij mjere ponavljanja,

   b) [2] postupak prema najmanjoj labavosti, uz sekundarni kriterij prema krajnjem trenutku završetka.

$$\begin{aligned} \mathcal{T}_1: &\quad T_1 = 5 \text{ ms}, \quad C_1 = 2 \text{ ms} \\ \mathcal{T}_2: &\quad T_2 = 7 \text{ ms}, \quad C_2 = 5 \text{ ms} \\ \mathcal{T}_3: &\quad T_3 = 10 \text{ ms}, \quad C_3 = 6 \text{ ms} \\ \mathcal{T}_4: &\quad T_4 = 20 \text{ ms}, \quad C_4 = 6 \text{ ms} \end{aligned}$$

a)

```
    4                                       4
    3                   3                   3
    2               2               2
    1           1           1           1           1
t:  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
P1:  2 2 2 2 2 1 1 2 2 2 2 2 4 4 4 1 1 4 2 2
P2:  1 1 3 3 3 3 3 3 4 4 1 1 3 3 3 3 3 3 - -
```

b)

```
    4                                       4
    3                   3                   3
    2               2               2
    1           1           1           1           1
t:  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
P1:  2 2 2 2 2 1 1 2 2 2 2 2 4 4 2 2 1 2 1 2
P2:  1 1 3 3 3 3 3 3 4 4 1 1 3 3 3 3 3 4 3 4

l1: 3 3 - - - 3 3 - - - 3 3 - - - 3 2 2 1 - 2
l2: 2 2 2 2 2 - - 2 2 2 2 2 - - 2 2 2 1 1 0 -
l3: 4 3 2 2 2 2 2 2 - - 4 3 2 2 2 2 2 2 1 - 3
l4:1413121110 9 8 7 6 6 6 5 4 4 4 3 2 1 1 014
```

LLF je očito efikasniji: zadatak 2 je u svom trećem javljanju obavio 4 jedinice vremena, dok je u kod EDF-a samo dvije, a procesor nema što raditi dvije jedinice

3. [2] Dretve A, B, C i D bude se trenutcima: 1. ms, 3. ms, 5. ms i 7. ms respektivno (A u 1. ms). Dretve se raspoređuju prema prioritetu, a prioriteti dretvi su: p(A)=1, p(B)=3, p(C)=4, p(D)=2. Po buđenju svaka dretva radi nešto 1 ms. Potom dretve traže sredstvo (semafore): A→S1, B→S2, C→S1, D→S2 i s njim rade 2 ms. Nakon toga otpuštaju sredstvo i rade još 1 ms prije nego završe se radom. Prikazati stanje jednoprocesorskog sustava dok sve dretve ne završe s opisanim poslom, ako se za semafore koristi protokol nasljeđivanja prioriteta.

```
ima semafor Sx:      1   2   1 1 1   2     2 2
aktivna dretva:   A A B B C A C C C B B D D D D A
vrijeme:       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
dolaze dretve:   A   B   C   D
```

4. [2] U nekom sustavu koji nema podršku za raspoređivanje prema krajnjem trenutku završetka treba ga ostvariti programski. Svaki posao zadan je identifikatorom id (1,2,...), periodom `T`, relativnim trenutkom krajnjeg završetka u odnosu na vrijeme početka periode `d`. Dovršiti implementaciju u nastavku tako da se riješi problem paralelnog pristupa zajedničkim strukturama podataka te problem radnog čekanja. Koristiti proizvoljne mehanizme, ali koji postoje u POSIX sučeljima (ali ne treba koristiti njihova imena; npr. može `ČekajSem` umjesto `sem_wait` i slično).

```
poslovi[] - uređeno polje s elementima {id, D, status=radi|čeka}
            D je apsolutni trenutak do kada posao mora biti gotov
posao_dretve(p) { //p = {id, T, d}
   t = dohvati_vrijeme()
   ponavljaj {
      t = t + T
      odgodi_do(t)

      dodaj_u_listu(id, t+d, ČEKA)
      dok je p.periodički_posao_nije_gotov() {
         ako je poslovi[0].id == id { //ovaj je prvi u redu
            poslovi[0].status = RADI
            p.odradi_dio_posla()
         }
         inače {
            poslovi[0].status = ČEKA
            //zaustaviti dretvu da ne radi radno čekanje
         }
      }
      makni_iz_liste(id)
   }
}
```

```
posao_dretve(p) { //p = {id, T, d}
   t = dohvati_vrijeme()
   ponavljaj {
      t = t + T
      odgodi_do(t)

      mutex_lock(m)
      dodaj_u_listu(id, t+d, čeka)
      dok je p.periodički_posao_nije_gotov() {
         ako je poslovi[0].id == id {
            poslovi[0].status = radi
            mutex_unlock(m)
            p.odradi_dio_posla()
            mutex_lock(m)
         }
         inače {
            poslovi[0].status = čeka
            cond_wait(red, m)
         }
      }
      makni_iz_liste(id)
      cond_broadcast(red)
      mutex_unlock(m)
   }
}
```

5. [1] Što je sve potrebno napraviti programski i inače da bi se neku dretvu postavilo kao kritičnu koja se raspoređije preko SCHED_RR s prioritetom 77 na Linuxu? Koji su sve koraci u kodu i van njega?

   1. u programu pozvati sučelje da se postavi način raspoređivanje SCHED_RR
   2. u programu pozvati sučelje da se postavi prioritet 77
   2. pokrenuti program s root ovlastima

6. [2] U nekom sustavu (sličnom kao u 5. laboratorijskoj vježbi) postoje dva procesa: *zaprimatelj* i *obrada*. Zaprimatelj prima vanjske zahtjeve sa `z=zaprimi()`, oblikuje poruku sa zahtjevom i šalje je u red sa `pošalji_u_red(id, p)`. Proces *obrada* ima više dretvi koje čekaju na poruke iz tog reda te ih uzimaju sa `p=čekaj_na_poruku(id)` i obrađuju sa `obradi(p)`. Obzirom da je zaprimanje zahtjeva vremenski kritično, proces zaprimatelj koristi raspoređivanje SCHED_RR i pokrenut je od strane privilegiranog korisnika *zaprimatelj*, dok dretve procesa *obrada* su obične dretve (SCHED_OTHER) i sam je proces pokrenut u kontekstu običnog korisnika *radnik*. Skicirati kod programa zaprimatelja (samo jedna dretva) te kod jedne dretve procesa obrade. Naglasiti sve detalje koje treba ispravno napraviti da bi procesi radili prema opisanome kada bi se koristila POSIX sučelja na Linuxu (ili sličnom UNIX sustavu).

   Obzirom da komuniciraju procesi različitih korisnika pri stvaranju reda treba im omogućiti pristup redu. Npr. dati svima dozvole (0666, kao u donjem rješenju) ili bolje stvoriti novu grupu u koju staviti ta dva korisnika, tj. programe označiti da pripadaju toj grupi (tada su dozvole 0660).
   U donjem rješenju zaprimatelj stvara red, pa ga je potrebno prvog pokrenuti (u okviru korisnika zaprimatelj)

```
dretva_zaprimi() {
    id = otvori_red_poruka(id_reda, STVORI_RED | SAMO_ZA_PISANJE, 0666)
    ponavljaj {
      z = zaprimi()
      pošalji_u_red(id,z)
    }
}
dretva_obrada() {
    id = otvori_red_poruka(id_reda, SAMO_ZA_ČITANJE)
    ponavljaj {
        poruka = čekaj_na_poruku(id)
        obradi_poruku(poruka)
    }
}
```

7. [1] U nekom sustavu koji koristi sabirnicu CAN dva čvora istovremeno započinju sa slanjem svoje poruke, prvi hoće poslati poruku 0x123456789, a drugi 0x1111111111. Što će se dogoditi?

   Poruka drugog će se poslati do kraja, prvi će odustati o daljenjeg slanja nakon što detektira da ono što šalje nije na vodiču.

8. U nekom sustavu čvor A želi sinkronizirati svoj sat sa satom čvora B i šalje mu poruku u koju stavlja svoj trenutni sat. Po zaprimanju poruke čvor B zapisuje trenutnu vrijednost svog sata u poruku i odmah ju vraća čvoru A. Pri primitku povratne poruke čvor A očitava svoj sat. Ako su navedene vrijednosti redom 5555, 5565 i 5595, koju vrijednost će čvor A dodati na svoj sat:

   a) [1] uz pretpostavku simetrične veze

   b) [1] uz pretpostavku asimetrične veze, gdje slanje traje triput duže od primanja?

```
a) 5555 - 5595
   5565 je na pola, barem po satu poslužitelja
   pola po satu klijenta je 5575
   razlika je -10, tu vrijednost treba dodati na sat klijenta
b) trenutak primitka poruke čvora B po satu A je
   5555 + (5595-5555)*3/4 = 5555 + 30 = 5585
   5585 - 5565 = 20 => ide napred 20 jedinica, -20 treba dodati
```

9. [2 boda] U sustavima koji koriste jednostavan RTOS (npr. FreeRTOS) programira se cijeli sustav, od upravljačke petlje ili upravljačkih dretvi do prekidnih rutina. Koje su prednosti korištenja takva sustava, a koja ograničenja?

```
+ mali zahtjevi na resurse (takav OS je jako mali)
+ potpuna kontrola sustava iz programa
- potrebno potpuno poznavanje OS-a, sklopovlja i programa
- zahtjevno za ostvariti (manje API-a, nema dovoljno dobrih programera za to)
```

10. [2 boda] Pri detekciji šuma u nekom nepouzdanom kanalu preko koje komuniciraju dva čvora koristi se ispitni signal koji je opisan s tri vrijednosti `a`, `b` i `c`. Prosječna amplituda signala definirana je funkcijom: `P=a*K1+b*K2+c*K3` gdje su `K1`, `K2` i `K3` konstante. U nastavku je prikazan dio jednog rješenja. Navesti moguće probleme i kako ih popraviti.

```
#define K1 1.23456789012345678901234567890e35
#define K2 2.34567890123456789012345678900
#define K3 3.45678901234567890123456789001e-35
//testni signal:
#define A 55.66
#define B 33.44
#define C 11.11
struct X {
    long double a, b, c;
};
void slanje() {
    struct X x = {A, B, C};
    for (;;)
        posalji(x);
}
long double odredi_gresku() {
    struct X y;
    long double greska = 0, primljeno;
    int i, j;
    for (i = 0; i < 100; i++) {
        int r = gen_rnd(300000,500000); //slučajni broj iz intervala
        for (j = 0; j < r; j++) //za simulaciju kašnjenja
            ;
        y = primi();
        primljeno = y.a * K1 + y.b * K2 + y.c * K3
        greska += primljeno - (A * K1 + B * K2 + C * K3);
    }
    return greska/100;
}
```

```
//dodati L na kraj inače se može broj spremiti kao običan double
>>>#define K1 1.23456789012345678901234567890e35L
>>>#define K2 2.34567890123456789012345678901L
>>>#define K3 3.45678901234567890123456789012e-35L
...
long double odredi_gresku()
{
...
>>>     for (j = 0; i < r; i++) //za simulaciju kašnjenja
>>>     for (j = 0; j < r; j++) //nenamjerno dodana c/p greška, j umjesto i
>>>         memorijska_barijera();//treba dodati da kompajler ovo ne makne
...
>>>     primljeno = y.a * K1 + y.b * K2 + y.c * K3
>>>     greska += primljeno - (A * K1 + B * K2 + C * K3);
###     Ovo je loše zbog nekoliko razloga:
###     - ogromne razlike u K1, K2 i K3 mogu uzrokovati da se greška "izgubi"
###     - greške mogu biti pozitivne i negativne - zbrajanjem se "poništavaju"
###       imalo bi više smisla uzimati apsolutne vrijednosti
###     - jedna mogućnost popravka:
>>>     greska += abs(y.a - A) * K1 + abs(y.b - B) * K2 + abs(y.c - C) * K3
###     druge mogućnosti uključuju zasebno praćenje grešaka pojedinih
###     komponenti ili neka drukčija formula koja ima smisla za problem,
###     uzimajući u obzir ovaj problem preciznosti
    }
    return greska/100;
}
```