1. [4] U nekom proizvodnom procesu sudjeluju strojevi A, B i C. Kad se proces pokrene, najprije stroj A radi svoj prvi dio posla koji traje dvije minute. Potom s priključuju strojevi B i C na idućih pet minuta. Stroj B se zaustavlja, a A i C nastavljaju idućih 10 minuta. Nakon toga tri minute su svi strojevi neaktivni te se postupak ponavlja. Opisati navedeni proces najprikladnijim UML dijagramom.

Sekvencijski dijagram

t	A	B	C
1	X	|	|
2	X	|	|
3	X	X	X
4	X	X	X
5	X	X	X
6	X	X	X
7	X	X	X
8	X	|	X
9	X	|	X
10	X	|	X
11	X	|	X
12	X	|	X
13	X	|	X
14	X	|	X
15	X	|	X
16	X	|	X
17	X	|	X
18	|	|	|
19	|	|	|
20	|	|	|

2. [4] Neko upravljačko računalo treba upravljati aktivnostima X, Y i Z. X i Y su periodički poslovi, za X treba svakih 200 ms pozvati `X()`, za Y svakih 250 pozvati `Y()`. Trajanja tih funkcija su do 20 ms. Aktivnost Z se upravlja tako da se prati stanje ulaza `ULAZ` i kada se ono promijeni (0=>1, 1=>0) treba pozvati funkciju `Z(staro_stanje, novo_stanje)` koja traje vrlo kratko. Reakcija na `ULAZ` ne bi smjela kasniti više od 30 ms. Ostvariti upravljački program ako na raspolaganju (pored navedenih funkcija) stoji i `dohvati_ulaz()` koja dohvaća stanje `ULAZ`-a te `dohvati_sat_ms()` koja dohvaća stanje brojila koje odbrojava u milisekundama.

```
tx = ty = dohvati_sat_ms()
staro_stanje = dohvati_ulaz()
ponavljaj {
    novo_stanje = dohvati_ulaz()
    ako je (novo_stanje != staro_stanje) {
        Z(staro_stanje, novo_stanje)
        staro_stanje = novo_stanje
    }
    inače ako je (dohvati_sat_ms() >= tx + 200) {
        tx += 200
        X()
    }
    inače ako je (dohvati_sat_ms() >= ty + 250) {
        ty += 250
        Y()
    }
}
```

3. [4] Neki PID regulator zadan je parametrima $K_P = 0,5$, $K_I = 0,2$ i $K_D = 0,05$. Ako je stanje sustava u nekom trenutku $y_k = 100$, željeno stanje $y_P = 200$ izračunati r_k (vrijednost s kojom se utječe na sustav)? Prethodno stanje sustava je $y_{k-1} = 80$, uz $I_{k-1} = 20$. Korak integracije je $T = 0,1$ s.

```
Kp=0,5
Ki=0,2
Kd=0,05
yk=100
yp=200      => ek=yp-yk=100
yk-1=80     => ek-1=120
               Dk = (ek-ek-1)/T = (100-120)/0,1 = -200
Ik-1=20     => Ik = Ik-1 + ek*T = 20 + 100*0,1 = 30
T = 0,1     rk = Kp*ek + Ki*Ik + Kd * Dk
               = 0,5*100 + 0,2*30 - 0,05*200 = 50+6-10 = 46
```

4. [4] Zadan je sustav zadataka koji se raspoređuje na jednoprocesorskom sustavu. Grafički prikazati izvođenje sustava, počevši od kritičnog slučaja u t=0 ms do t=20 ms ili do prekoračenja ograničenja, ako se koristi raspoređivanje mjerom ponavljanja (RMPA).

$$\begin{aligned} \mathcal{T}_1: &\quad T_1 = 5 \text{ ms}, &\quad C_1 = 1 \text{ ms} \\ \mathcal{T}_2: &\quad T_2 = 7 \text{ ms}, &\quad C_2 = 2 \text{ ms} \\ \mathcal{T}_3: &\quad T_3 = 10 \text{ ms}, &\quad C_3 = 3 \text{ ms} \\ \mathcal{T}_4: &\quad T_4 = 20 \text{ ms}, &\quad C_4 = 3 \text{ ms} \end{aligned}$$

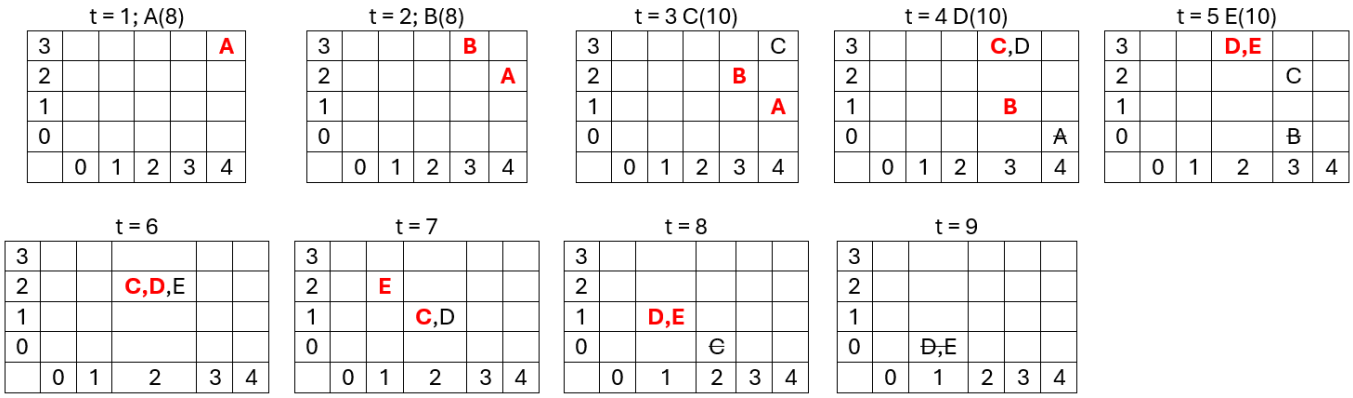
```
    4                                   4
    3                   3               3
    2               2               2
    1           1           1           1           1
t:  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
P1:  1 2 2 3 3 1 3 2 2 4 1 3 3 3 2 1 2 4 4 -
-----------------------------------------------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

5. [4] 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 postupak prema rokovima završetka, uz sekundarni kriterij mjere ponavljanja.

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

```
    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   (T2 ne stiže do 21!)
P2:  1 1 3 3 3 3 3 3 4 4 1 1 3 3 3 3 3 3 - -   (ali to nije traženo)
```

6. [4] Zadatci A, B, C, D i E bude se trenutcima: 1. ms, 2. ms, 3. ms, 4. ms i 5. ms respektivno (A u 1. ms) i moraju biti gotovi sa svojim poslovima u 8. ms, 8. ms, 10. ms, 10. ms i 10. ms respektivno (A u 8. ms). Svaki zadatak treba 3 ms procesorskog vremena. Zadatci se raspoređuju na dvoprocesorskom računalu prema labavosti kao primarnom, rokovima kao sekundarnom kriteriju te redu prispijeća kao tercijalnom. Korištenjem ℓ-c grafa pokazati rad raspoređivača nad dotičnim zadatcima.



7. [4] Dretve A, B, C i D bude se trenutcima: 1. ms, 2. ms, 3. ms i 4. ms respektivno (A u 1. ms). Dretva A se raspoređuje prema SCHED_OTHER s nice=5, dretva B sa SCHED_RR i prioritetom 20, dretva C sa SCHED_FIFO i prioritetom 30 te dretva D sa SCHED_FIFO i prioritetom 20. Po buđenju svaka dretva treba 4 ms procesorskog vremena. Pokazati kako će se izvoditi dretve na jednoprocesorskom računalu.

```
P:              - A B C C C C B D D D D B B A A 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
G:                           C         D   B     A

priznato i D od 7. ms umjesto B.
```

8. [4] Dretve A, B, C i D bude se trenutcima: 1. ms, 3. ms, 5. ms i 8. 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 2 ms. Potom dretve traže sredstvo (semafore): A→S1, B→S2, C→S1, D→S2 i s njim rade 4 ms. Nakon toga otpuštaju sredstvo i rade još 2 ms prije nego završe se radom. Prikazati stanje dvoprocesorskog sustava dok sve dretve ne završe s opisanim poslom, ako se za semafore koristi protokol nasljeđivanja prioriteta.

```
ima semafor S1:        A A a a A A C C C C - - - -
ima semafor S2:            B B B B - - - - D D D D
P1:             - A A A A C C A A C C C C C C A A
P2:             - - - B B B B B B B B D D D D D D D D
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
G:                                     B       C   A   D
```

9. [2] [1] U nekom sustavu čvor A želi sinkronizirati svoj sat sa satom čvora B. Stoga mu šalje poruku u koju stavlja trenutnu vrijednost svog sata t=5555. Po zaprimanju poruke, čvor B poruku proširuje trenutnom vrijednošću svog sata t=5565 i vraća ju čvoru A. Vrijednost sata u čvoru A pri primitku poruke je t=5595. Koju vrijednost će čvor A dodati na svoj sat uz pretpostavku simetrične veze?

```
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
```

10. [2 boda] Neke funkcije u FreeRTOS-u imaj sufiks `FromISR`. Po čemu je takva funkcija različita od istoimene bez tog sufiksa (npr. `xQueueSend` i `xQueueSendFromISR`)?

```
pripremljena je da se može pozivati iz obrade prekida
```

11. [4 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`. Amplituda signala definira se funkcijom: `P=a*K1+b*K2+c*K3` gdje su `K1`, `K2` i `K3` konstante. U nastavku je prikazan dio jednog rješenja u kojem jedna dretva neprestano šalje ispitni signal (na jednoj strani, poziva `slanje()`), a druga (na drugoj strani) povremeno čita stanje kanala i dekodira pročitani signal. Navesti moguće probleme navedena koda i kako ih popraviti.

```
#define K1 1.23456789012345678901234567890e35
#define K2 2.34567890123456789012345678900
#define K3 3.45678901234567890123456789001e-35
```

dodati L na kraj prethodnih konstanti
inače se može broj spremiti kao običan double i izgubiti preciznost

```
//testni signal:
#define A 55.66
#define B 33.44
#define C 11.11
```

- na prvi pogled ovdje ne bi trebalo dodati L jer su samo s dvije decimale
- ALI u binarnom obliku tu brojevi imaju beskonačno decimala pa je svaka bitna kad je potrebna velika preciznost (a ovdje očito jest)
(ali ovo ipak nije nužno za sve bodove :))

```
struct X {
    long double a, b, c;
};
void slanje() { //poziva se u prvom čvoru
    struct X x = {A, B, C};
    int i;
    for (i = 0; i < 10000000000UL; i++)
        posalji(x);
}
```

- "i" treba biti veće tipa, npr. long long
- dobro bi bilo provjeriti status funkcije "posalji"
- neki su ovdje pisali da umjesto ove duge petlje staviti while(1)
to je zapravo i bolje u duhu zadatka koji koristi "neprestano šalje"

```
long double odredi_gresku() { //poziva se u drugom čvoru
    struct X y;
    long double greska = 0, primljeno;
    int i, j;
    for (i = 0; i < 100; i++) {
        //simulacija vrlo kratkog intervala
        int r = 300000 + random() % RAND_MAX;
```

- neki su ovdje komentirali da je RAND_MAX možda prevelik i ne stane u int
*** obično to nije problem, jer je on vezan uz int tip (najčešće)**
- drugi da random nije "thread safe": ali ovdje nema dretvi

```
        for (j = 0; j < r; j++)
            ;
```

u petlju treba dodati nešto (mem. barijeru) da ju prevoditelj ne bi maknuo

```
        //dohvat i obrada
        y = primi();
        primljeno = y.a * K1 + y.b * K2 + y.c * K3
        greska += primljeno - (A * K1 + B * K2 + C * K3);
```

- 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 (ili kvadrate)
- mogućnosti popravka:
*** greska += abs((y.a - A) * K1 + (y.b - B) * K2 + (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;
}
```