JEDNO moguće ispravno rješenje za:

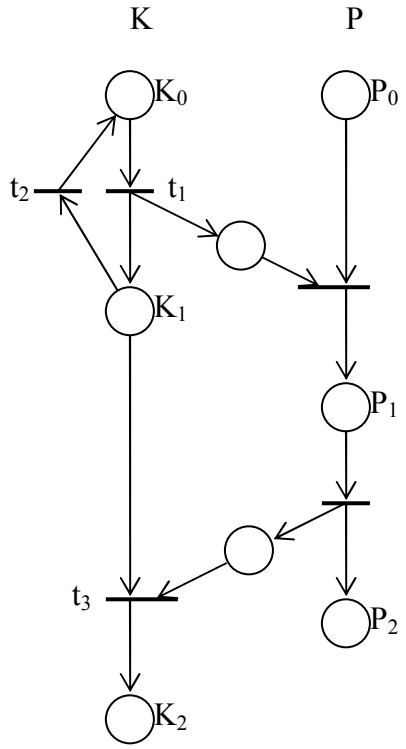
Prvi međuispit iz predmeta *SUSTAVI ZA RAD U STVARNOM VREMENU* 3.4.2009.

1. U nekom komunikacijskom sustavu postoje dvije strane: klijent (K) i poslužitelj (P). Uspostava komunikacijskog kanala obavlja se tako da klijent pošalje zahtjev (REQ) na koji poslužitelj odgovara (ACK).
   (4) Korištenjem petrijevih mreža modelirati postupak uspostavljanja kanala.
   Pretpostaviti da se klijent može naći u stanjima:
   - K_0 – prije slanja zahtjeva REQ (početno stanje),
   - K_1 – nakon slanja zahtjeva REQ, čekanje na odgovor,
   - K_2 – nakon primitka odgovora ACK – veza uspostavljena.

   Poslužitelj se može naći u stanjima:
   - P_0 – prije primitka zahtjeva REQ (početno stanje),
   - P_1 – nakon primitka zahtjeva REQ, a prije slanja odgovora ACK,
   - P_2 – nakon slanja odgovora ACK – veza uspostavljena.

   Odgovor na klijentov zahtjev ne smije doći prije T_1=10 μs (jer za ovaj sustav protok poruka sigurno traje dulje), ali ni kasnije od T_2=500 ms („timeout“). U oba slučaja klijent treba ponoviti slanje zahtjeva REQ (vratiti se u stanje K_0).
   (2) Za klijentsku stranu petrijeve mreže pokazati kada (vremenski gledano) pojedine tranzicije mogu „okinuti“ (napisati tablicu tranzicija koja proširuje petrijevu mrežu u vremensku petrijevu mrežu, ali samo za klijenta).

   Rj.
   Iduće rješenje je najjednostavnije, sa samo navedenim stanjima za klijenta i poslužitelja te dva „prijenosna“ stanja.



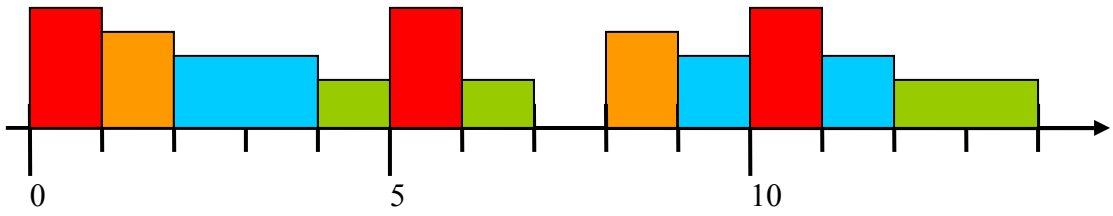
	vrem.	stanja	vremena tranzicija
t1	-	K_0	$(time(K_0), \infty)$
t2	T_1, T_2	K_1	$(time(K_1), T_1] \cup [time(K_1)+T_2, \infty)$
t3	T_1, T_2	K_1	$(time(K_1)+T_1, time(K_1)+T_2)$

Osim navedenog, problem je moguće riješiti i bez „prijenosnih“ stanja. Npr. iz t_1 token može ići u P_0, ali tada veza P_0 do iduće tranzicije mora biti težine 2; također t_2 bi trebao „pokupiti“ jedan token iz P_0 kada se tranzicija dogodi.
Moguće da ima još dobrih rješenja…

2. Zadan je sustav zadataka τ_1 do τ_4. Zadaci su zadani s vremenima ponavljanja (oznake T_i) te s vremenima izračunavanja (oznake c_i).
   $T_1 = 5$ $c_1 = 1$
   $T_2 = 8$ $c_2 = 1$
   $T_3 = 9$ $c_3 = 2$
   $T_4 = 10$ $c_4 = 2$
   Ako se za raspoređivanje koristi RMPA metoda odrediti:
   a) (3) je li sustav rasporedljiv (pokazati računski – nužni uvjet, te grafički – za kritični slučaj);
   b) (2) implicitne trenutke krajnjeg završetaka zadataka (implicitni „deadline“, računski i/ili grafički);
   c) (2) da li zadani skup zadataka u potpunosti iskorištava procesor; ako ne, koji se zadaci mogu dulje izračunavati i koliko.

   Rj.

   a)
   U = 1/5 + 1/8 + 2/9 + 2/10 = 0,74 ➔ procesorska iskoristivost manja od 1!



   Prema slici sustav je rasporedljiv i u kritičnom slučaju!

   b) Implicitni deadline (iz slike):
   - za τ_1 → $d_1 = 5$
   - za τ_2 → $d_2 = 8$
   - za τ_3 → $d_3 = 8$
   - za τ_4 → $d_4 = 8$

   c) može „još“ (da se iskoristi i „rupa“): $c_{1,max} = 1,5$, ili $c_{2,max} = 2$, ili $c_{3,max} = 3$, ili $c_{4,max} = 3$

3. (1) Za neki RT sustav kaže se: svako dodatno kašnjenje u izračunu (nečega) značajno povećava cijenu (nečeg drugog). U koju kategoriju RT sustava spada ovaj? Obrazložiti.

Rj.
- spada u sustave s ublaženim vremenskim ograničenjima (soft RT);
- zato jer se dozvoljava kašnjenje – iako ono ima cijenu, kašnjenje ne rezultira „ispadom“ sustava, tj. nepopravljivom greškom

4. (2) Pri oblikovanju programa često se koristi podjela na segmente (podzadatke). Navesti nekoliko prednosti i nedostataka korištenja podjele, posebice gledano s aspekta arhitekta RT sustava.

Rj.
- neke prednosti:
  (dobra) podjela rezultira odvajanjem „složenog“ problema u jednostavnije, mogućnost grešaka se smanjuje, (lakše se testira, održava);
  ako se segmenti (zadaci) mogu izvoditi (bar dijelom) nezavisno može se bolje iskoristiti i višeprocesorsko računalo (ili tek na njemu sustav može raditi);

(višezadaćni program može biti razumljiviji i jednostavniji za ostvarenje)

- neki nedostaci:
  svaka podjela unosi potrebu za prijenosom podataka među segmentima – problem performansi, dodatne (nepotrebne) provjere parametara, komunikacija i sinkronizacija između zadataka može znatno usporiti rad sustava i uvesti nedeterminizam; (loše izvedena) sinkronizacija može dovesti do potpunog zastoja!

5. (1) U nekom jednostavnom sustavu periodički zadaci se obrađuju onim redoslijedom kojim se javljaju. U koju kategoriju raspoređivanja zadataka (statičko/dinamičko) možemo svrstati ovakvo raspoređivanje? Obrazložiti!

Rj.

- ako razmatramo sa stanovišta prioriteta, onda može biti dinamičko: zadatak „n" koji se javlja periodički, svaki puta najprije ide u red (jer se obrada obavlja po redu prispijeća), i to na kraj reda – „raspoređivač" mu daje najmanji prioritet; kada se idući put pojavi isti zadatak, opet će dobiti najmanji prioritet, ali će taj biti i manji od prethodnog puta, jer svi zadaci koji su se u međuvremenu pojavili također imaju manji prioritet, ali veći od novog pojavljivanja zadataka;
  iako na predavanju nije bilo napomenuto, različiti zadaci mogu imati isti prioritet – ako svi zadaci imaju isti prioritet, raspoređivač može uz prioritet koristiti FIFO (za zadatke istog prioriteta) – tj. mogli bi reći da je tada raspoređivanje statičko
- ako bi razmatrali sa stanovišta ponašanja raspoređivača, tj. treba li nekakva posebna logika koja će dinamički određivati idući zadatak, onda je statičko raspoređivanje odgovor, jer posluživanje po redu prispijeća to ne zahtijeva (to je najjednostavniji oblik raspoređivanja)

6. (1) Najniža gornja granica faktora iskorištenja procesora (kada se koristi RMPA) za skup od m zadataka je lub=X, a za skup od m+1 zadataka lub=Y. U kakvom su odnosu X i Y? Obrazložiti (bez formule)!

Rj.

- $Y < X$, zato jer je m+1 zadatak teže rasporediti od m zadataka, odnosno, moramo ostaviti više procesorskog vremena za eventualne komplikacije u raspoređivanju (procesor treba biti manje opterećen)

7. (2) Koje su prednosti, a koji nedostaci u korištenju RT operacijskog sustava (naspram rješenja bez Osa) u ostvarenju nekog RT sustava?

Rj.

- neke prednosti: OS olakšava korištenje računala – ima API-je kojima jednostavnije napravimo gotovo sve (ne moramo se brinuti oko npr. načina rada s nekim sklopovljem, organizacijom memorije i sl.),
- neki nedostaci: OS unosi nedeterminizam – trajanje izvođenja API-ja (jezgrinih funkcija) može ovisiti o mnogo faktora – ne mora biti isto svaki puta, može predugo trajati i eventualno predugo zadržati obradu nekog bitnog prekida; zato pravi RTOS-evi pokušavaju sve riješiti determinističkim metodama (iako one ne moraju biti optimalne sa stanovišta učinkovitosti)