Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, decembar 2014.
Datum: 6.12.2014.

Drugi kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 1,5 sat. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____%

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Ocenjivanje unutar potpitanja je po sistemu "sve ili ništa", odnosno nema parcijalnih poena. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**.

1. (10 poena) Mrtva blokada

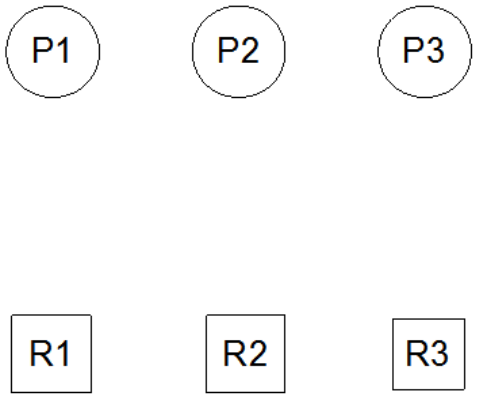
U nekom sistemu primenjuje se algoritam izbegavanja mrtve blokade zasnovan na grafu alokacije. Posmatraju se tri procesa, $P1$, $P2$ i $P3$ i tri resursa, $R1$, $R2$ i $R3$. Proces $P1$ najavio je korišćenje resursa $R1$ i $R2$, proces $P2$ korišćenje sva tri resursa, a proces $P3$ korišćenje resursa $R2$ i $R3$.

a)(5) Procesi su izvršili sledeće akcije zahteva za resursima, tim redom: $P2 - R2$, $P1 - R1$, $P3 - R3$. Prikazati izgled grafa alokacije nakon ova tri zahteva.

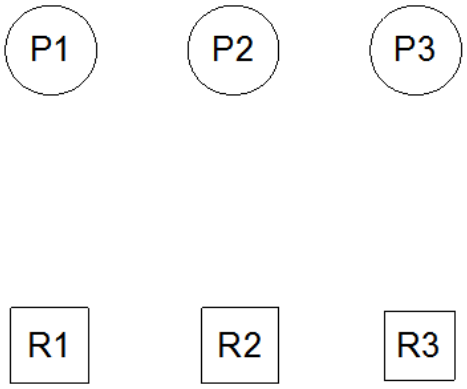
b)(5) Nakon sekvence zahteva iz tačke a), u nastavku proces $P2$ oslobađa resurs $R2$. Prikazati izgled grafa alokacije nakon ove akcije.

Rešenje:

a)



b)



2. (10 poena) Upravljanje memorijom

Za izbor stranice za zamenu u nekom sistemu koristi se prošireni algoritam „davanja nove šanse“ (*second-chance algorithm*), tako što se pored bita referenciranja uzima u obzir i „zaprljanost“ stranice, odnosno posmatra se par (*reference bit, modify bit*). Primenjuje se lokalna politika zamene stranice (samo za stranice istog procesa). U deskriptoru stranice u PMT, koji je tipa `unsigned long int`, najniži bit je bit modifikacije, a pored njega je bit referenciranja. U posebnom nizu `pagefifo` svaki element odgovara jednoj stranici i sadrži indeks ulaza u tom nizu koji predstavlja sledeću stranicu u kružnoj FIFO listi stranica uređenih po redosledu učitavanja.

```
struct PCB {
  ...
  unsigned int clockHand;
  unsigned long* pmt; // Pointer to the PMT
  unsigned int* pagefifo; // Pointer to the FIFO page table
};
```

Funkcija `getVictimPage(PCB*)` implementira ovaj algoritam i vraća broj stranice koju treba zameniti po ovom algoritmu zamene, za proces sa datim PCB. Ova funkcija u četiri prolaza poziva funkciju `findBestCandidate()` koja obilazi samo one stranice kojima je *modify* bit postavljen na datu vrednost i pronalazi takvu kojoj je *reference* bit obrisan, dok ostalim na koje naiđe pre nje „daje novu šansu“ i briše *reference* bit. Ukoliko takvu traženu ne nađe u jednom prolazu kroz celu listu, vraća 0, kako bi naredni pozivi mogli da obiđu neku drugu klasu stranica (sa drugom vrednošću *modify* bita). Implementirati funkciju `findBestCandidate()`.

```
#define refmod(x) pcb->pmt[x]
#define next(x) pcb->pagefifo[x]
#define modmask 1UL
#define refmask 2UL

int findBestCandidate (PCB* pcb, unsigned int modifyBit);

unsigned int getVictimPage(PCB* pcb) {
  if (pcb==0) return -1; // Exception!
  unsigned int victim = -1;
  if (findBestCandidate(pcb,0)) {
    victim = pcb->clockHand;
    pcb->clockHand = next(victim);
    return victim;
  }
  if (findBestCandidate(pcb,1)) {
    victim = pcb->clockHand;
    pcb->clockHand = next(victim);
    return victim;
  }
  if (findBestCandidate(pcb,0)) {
    victim = pcb->clockHand;
    pcb->clockHand = next(victim);
    return victim;
  }
  if (findBestCandidate(pcb,1)) {
    victim = pcb->clockHand;
    pcb->clockHand = next(victim);
    return victim;
  }
  // Should never fall through here:
  return -1;
}
```

Rešenje:

3. (10 poena) Upravljanje memorijom

Neki sistem primenjuje alokator ploča (*slab*) za alokaciju memorije. U svakom kešu, ploče su iste veličine i povezane su u dvostruo ulančanu listu. U strukturi `Cache` polja `head` i `tail` ukazuju na prvu i poslednju ploču u kešu. Svaka ploča zauzima samo jednu stranicu, poravnata je na početak stranice i na samom početku sadrži zaglavlje predstavljeno strukturom `Slab`, iza kog sledi tačno *N* slotova za smeštanje objekata istog tipa *T*. Slobodni slotovi unutar iste ploče ulančani su u dvostruko ulančanu listu čija su glava i rep u poljima `head` i `tail` strukture `Slab`, preko pokazivača `Slot::next` i `Slot::prev` koji se upisuju u sam prostor slota koji je slobodan. Broj trentutno slobodnih slotova u jednoj ploči zapisan je u polju `freeSlots` strukture `Slab`. Kada se cela ploča isprazni (svi slotovi su prazni), ovaj alokator oslobađa celu ploču pozivom funkcije `free`.
Implementirati funkciju `slab_free_slot` koja, unutar datog keša, oslobađa slot na koga ukazuje drugi argument. Makro `page_align` vraća pokazivač poravnat na početak stranice u kojoj je adresa zadata argumentom.

```
const int N = ...;
#define page_align(ptr) ...
extern void free(void*);

struct Slot {          // Header stored in a free slot
  Slot *prev, *next;  // Next and previous free slot in the same slab
};

struct Slab {          // Header of each slab
  unsigned int freeSlots;  // Number of free slots in this slab
  Slab *prev, *next; // Next and previous slab in the same cache
  Slot *head, *tail; // Free slots in this slab
};

struct Cache {
  Slab *head, *tail; // List of all slabs in this cache
};

int slab_free_slot (Cache* cache, void* slot);
```

Rešenje: