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 2016.
Datum: 24.12.2016.

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

Tri procesa koriste resurse na način koji je dat pseudokodom dole. U tom pseudokodu označeni su zahtevi za alokacijom resursa koje dati proces koristi zajedno; na primer, `request(A,B,C)` znači da dati proces traži resurse *A*, *B* i *C*, jer ih koristi zajedno u delu svog izvršavanja. Međutim, sistem podržava alokaciju samo jednog resursa jednim zahtevom, pa procese treba rekonfigurisati tako da traže jedan po jedan resurs: dati primer zahteva treba pretvoriti u neku permutaciju sekvence zahteva `request(A)`, `request(B)`, `request(C)`.

Prekonfigurisati ove procese kako je opisano, s tim da mrtva blokada bude sigurno sprečena. Detaljno i precizno obrazložiti postupak.

```
Process X:              Process Y:              Process Z:
      request(B,D);           request(C,D);           request(D);
      request(C);             request(A);             request(B,C);
      release(C);             release(A);             release(B,C);
      release(B,D);           release(C,D);           release(D);
```

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem koristi rezervoar (engl. *pool*) slobodnih okvira radi ubrzanja obrade straničnih grešaka. Pri tom se za svaki oslobođeni okvir koji se smešta u rezervoar čuva informacija o tome kom procesu i kojoj njegovoj stranici je taj okvir pripadao. Prilikom alokacije okvira za traženu stranicu, sistem najpre pokušava da nađe okvir u kome je upravo tražena stranica tog procesa bila smeštena, i ako se takav okvir u rezervoaru pronađe, on se alocira. U suprotnom, alocira se bilo koji drugi slobodan okvir iz rezervoara.

Evidencija slobodnih okcira u rezervoaru vodi se u nizu `freeFramePool` veličine `FreeFramePoolSize` (to je maksimalna broj okvira u rezervoaru) deskriptora `FreeFrameDescr`. Element ovog niza može biti „prazan“ (`isSlotFree==true`) ili „zauzet“, u kom slučaju informacije o njemu govore kom procesu i kojoj njegovoj stranici je pripadao dati okvir referenciran tim deskriptorom.

```
typedef uint unsigned int;
struct FreeFrameDescr {
  ushort isSlotFree; // Is this slot in the pool's array free?
  uint frame; // The frame referenced by this descriptor
  uint pid;   // PID of the process that owned (released) this frame
  uint page;  // The page of the process that owned this frame
};
FreeFrameDescr freeFramePool[FreeFramePoolSize]; // The pool

int getFreeFrame (uint pid, uint page, uint* frame);
```

Implementirati funkciju `getFreeFrame` koja treba da pronađe slobodan okvir za traženu stranicu `page` datog procesa `pid` i njegov broj upiše u lokaciju na koju ukazuje argument `frame`. Ako nađe baš okvir koji je pripadao toj stranici, ova funkcija treba da vrati 1, ako nađe neki drugi slobodan okvir, treba da vrati 0, a ako slobodnih okvira u rezervoaru nema, treba da vrati -1.

Rešenje:

3. (10 poena) Upravljanje memorijom

U cilju sprečavanja pojave zvane *trashing*, sistem procenjuje veličinu radnog skupa svakog procesa na sledeći način. Za svaku stranicu koja pripada ločikom segmentu virtuelne memorije koji je proces alocirao, sistem vodi registar istorije bita referenciranja, koje periodično ažuirira pomeranjem udesno i upisivanjem sleva bita referenciranja. Struktura podataka koja čuva ove registre organizovana je u dva nivoa, poput dvonivoske tabele preslikavanja stranica, radi uštede prostora. U PCB strukturi procesa polje `refBits` tipa `RefBitsTable` predstavlja tabelu prvog nivoa, veličine `RefTableSize0` pokazivača na tabele drugog nivoa. Ove tabele drugog nivoa su tipa `RefBitsTable1` i veličine `RefTableSize1`, a sadrže registre istorije bita referenciranja tipa `RefBitReg`. Ukoliko neka tabela drugog nivoa ne postoji, jer nije alocirana, odgovarajući ulaz tabele prvog nivoa je *null*. Za stranice koje nisu korišćene ili nisu validne, u tabelama drugog nivoa odgovarajući ulazi imaju vrednost nula.

Kada izračunava veličinu radnog skupa procesa, sistem posmatra najviših `NumOfHistoryBits` bita u registrima istorije bita referenciranja – ako (i samo ako) je neki od njih jednak 1, ta stranica se smatra elementom radnog skupa. Implementirati funkciju `getWorkingSetSize` koja za dati proces izračunava i vraća veličinu radnog skupa na opisani način.

```
typedef unsigned long ulong;
typedef unsigned int RefBitReg;
const long RefBitTableSize0 = ..., RefBitTableSize1 = ...;
const unsigned short NumOfHistoryBits = ...;  // A small positive value
typedef RefBitReg RefBitsTable1[RefBitTableSize1];
typedef RefBitsTable1* RefBitsTable[RefBitTableSize0];

ulong getWorkingSetSize (PCB* pcb);
```

Rešenje: