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, novembar 2013.
Datum: 30.11.2013.

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 svaki proces i svaki resurs ima svoj jedinstveni identifikator (tipa `unsigned int`), a zauzeće resursa od strane procesa prati se u matrici `resourceAlloc` u kojoj vrste označavaju procese, a kolone resurse. U toj matrici vrednost -1 u ćeliji (*p*, *r*) označava da je proces sa identifikatorom *p* zauzeo resurs sa identifikatorom *r*, vrednost 1 označava da je proces *p* tražio, ali nije dobio resurs *r* (i čeka da ga dobije), a vrednost 0 označava da proces *p* nije ni zauzeo ni tražio resurs *r*. Operacije `allocate()` i `release()` procesi pozivaju kada traže, odnosno oslobađaju resurse. Kako jedan proces može u jednom zahtevu zatražiti samo jedan resurs, proces može biti blokiran čekajući samo na jednom resursu. Naravno, neki resurs može biti zauzet od strane najviše jednog procesa.

U sistemu se primenjuje tehnika sprečavanja mrtve blokade (*deadlock*) na sledeći način: kada proces *p* zatraži resurs *r* koji je već zauzet, proverava se da li bi blokiranje procesa uzrokovalo mrtvu blokadu; ukoliko bi, procesu se vraća greška, tako da se on ne blokira, već mora da preduzme alternativnu akciju zbog neuspeha alokacije resursa. Zbog toga je garantovano da u sistemu ni u jednom trenutku ne postoji mrtva blokada (ne postoji petlja u grafu alokacije predstavljenom datom matricom).

Za potrebe detekcije potencijalne mrtve blokade postoji operacija `wouldMakeDeadlock()` koja se poziva iz operacije `allocate()`. Ona treba da ispita da li bi zahtev za alokaciju datog resursa od strane datog procesa napravio petlju u grafu aklokacije predstavljenom matricom. Ukoliko bi je napravio, ova operacija treba da vrati 1 (*true*), inače treba da vrati 0 (*false*).

```
const unsigned MAXPROC = ...; // Maximum number of processes
const unsigned MAXRES = ...;  // Maximum number of resources
extern unsigned numOfProc;    // Actual number of processes
extern unsigned numOfRes;     // Actual number of resources

int resourceAlloc[MAXPROC][MAXRES];

int allocate (unsigned pid, unsigned rid);
int release  (unsigned pid, unsigned rid);

int wouldMakeDeadlock(unsigned pid, unsigned rid);
```

Implementirati operaciju `wouldMakeDeadlock()`.

Rešenje:

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. Ona treba da u više prolaza poziva pomoćnu postojeću funkciju `findBestCandidate()`. Ova pomoćna funkcija obavlja najviše jedan ceo prolaz kroz listu stranica i pri tom obilazi samo one stranice kojima je *modify* bit postavljen na datu vrednost. Ona vraća prvu takvu stranicu kojoj je *reference* bit obrisan (i tada prekida dalji obilazak liste), 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 -1, kako bi naredni pozivi mogli da obiđu neku drugu klasu stranica (sa drugom vrednošću *modify* bita). Implementirati funkciju `getVictimPage()`.

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

Rešenje:

3. (10 poena) Upravljanje memorijom

Neki sistem primenjuje algoritam parnjaka (*buddy*) za alokaciju memorije. Ceo prostor za alokaciju zauzima 256 susednih okvira, inicijalno je ceo slobodan, a najmanja jedinica alokacije je jedan okvir. Izvršene su redom operacije alokacije blokova datih veličina (u jedinicama, odnosno okvirima):
1, 16, 64, 16, 64.
Nakon toga je oslobođen prvi alocirani blok (veličine 1).
Napisati kako izgleda stanje memorije nakon svake od ovih operacija: navesti redom blokove koji su alocirani (A) ili slobodni (F) i njihove veličine (u jedinicama, odnosno okvirima). Odgovor (nakon svake izvršene operacije) je sekvenca oblika na pimer: A16, F32, A1, F64 itd.

Rešenje: