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 2015.
Datum: 6.12.2015.

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 izbegavanje mrtve blokade (engl. *deadlock avoidance*) korišćenjem grafa alokacije. Evidenciju zauzeća i potražnje resursa implementira klasa `ResourceAllocator` čija je delimična implementacija data dole. Važe sledeće pretpostavke:

- sve operacije u slučaju greške vraćaju negativnu vrednost sa kodom greške;
- polje `alloc` implementira graf zauzeća kao matricu;
- operacija `hasCycle()` proverava da li u ovakvom grafu ima petlje (1) ili je nema (0); ova operacija je na raspolaganju i ne treba je implementirati;
- operacija `announce()` se poziva kada proces *p* najavljuje korišćenje resursa *r*;
- operacija `request()` se poziva kada proces *p* traži resurs *r*;
- operacija `release()` se poziva kada proces *p* oslobađa resurs *r*; ova operacija treba da oslobodi resurs, ali i da potom ispita da li se u novonastaloj situaciji nekim drugim procesima mogu dodeliti resursi koje su oni zahtevali, a nisu mogli da ih dobiju iz bilo kog razloga; takvim procesima treba dodeliti resurse, a za svaki takav proces *p* u element *p* niza `deblock` treba upisati 1, kako bi sistem nakon povratka iz ove operacije znao da treba da deblokira taj proces, dok za ostale procese element niza `deblock` treba da bude 0.

Implementirati operaciju `release()`.

Rešenje:

```
class ResourceAllocator {
public:
  ResourceAllocator (int numOfProcesses, int numOfResources);

  int announce (int proc, int res);
  int request (int proc, int res);  // 1 – acquired, 0 – cannot get it
  int release (int proc, int res);

protected:
  int hasCycle ();
private:
  enum EdgeKind { unused, announced, requested, acquired };
  EdgeKind** alloc;
  int*  deblock;
  int np, nr;
};

ResourceAllocator::ResourceAllocator (int p, int r) : np(n), nr(r) {
  if (np==0 || nr==0) return; // Exception
  alloc = new (Edgekind*)[np];
  deblock = new int[np];
  if ((alloc==0) || (deblock==0)) return; // Exception
  for (int i=0; i<np; i++) {
    deblock[i] = 0;
    alloc[i] = new EdgeKind[nr];
    if (alloc[i]==0) return; // Exception
    for (int j=0; j<nr; j++) alloc[i][j] = unused;
  }
}

int ResourceAllocator::announce (int p, int r) {
  if (p<0 || p>=np || r<0 || r>=nr) return -1; // Exception
  if (alloc[p][r]!=unused) return 0; // No effect if already used
  alloc[p][r] = announced;  // Announcement edge
  return 1;
}

int ResourceAllocator::request (int p, int r) {
  if (p<0 || p>=np || r<0 || r>=nr) return -1; // Exception
  if (alloc[p][r]==unused) return -2;// Error: Request without announcement
  if (alloc[p][r]==acquired) return -3; // Error: Resource already acquired
  alloc[p][r] = requested;  // Request edge
  for (int j=0; j<np; j++)
    if (alloc[j][r]==acquired) return 0;  // Resource already occupied
  alloc[p][r] = acquired; // Try to allocate it and check for a cycle
  if (hasCycle()) {   // Does
    alloc[p][r] = requested;
    return 0;  // Cannot be acquired due to a possible deadlock
  } else
    return 1;  // Resource acquired
}
```

2. (10 poena) Upravljanje memorijom

Za izbor stranice za zamenu u nekom sistemu koristi se modifikovan algoritam „davanja nove šanse“ (*second-chance algorithm*), tako što se umesto samo bita referenciranja, svakoj stranici pridružuje brojač korišćenja koji se postavlja na 0 prilikom učitavanja stranice. Ovaj brojač sistem ažurira periodično, na osnovu vrednosti bita referenciranja, inkrementirajući brojač ukoliko je u datoj periodi stranica bila korišćena, odnosno bit referenciranja postavljen. Pri izboru stranice-žrtve, kada kazaljka dođe do neke stranice, ako je ovaj brojač jednak nuli, stranica se bira za izbacivanje; u suprotnom se stranici daje nova šansa, a brojač dekrementira. Primenjuje se lokalna politika zamene stranice (samo za stranice istog procesa).

U PCB procesa, 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. Niz `pageCounters` sadrži brojače pridružene stranicama.

```
struct PCB {
  ...
  unsigned int clockHand;
  unsigned int* pageCounters;  // Page usage counters
  unsigned int* pageFifo; // Pointer to the FIFO page table
};
```

Implementirati funkciju `getVictimPage(PCB*)` koja implementira ovaj algoritam i vraća broj stranice koju treba zameniti po ovom algoritmu zamene, za proces sa datim PCB.

Rešenje:

3. (10 poena) Alokacija memorije za potrebe kernela

Kernel nekog operativnog sistema koristi *buddy* alokator memorije za potrebe alokacije svojih struktura podataka. Najmanja jedinica alokacije je jedna stranica veličine 4KB, a adresibilna jedinica je bajt. Na početku je ceo segment memorije kojom rukuje ovaj alokator slobodan, a stanje strukture podataka koju koristi ovaj alokator prikazana je tabelarno. U prvoj koloni je stepen *n*, a u drugoj koloni su početne adrese slobodnih blokova veličine 2^n stranica.

n	*Početne adrese (hex) slobodnih blokova veličine 2^n*
4	A 00 00
3	-
2	-
1	-
0	-

a)(5) Prikazati na isti način stanje ove strukture podataka nakon sledeće sekvence alokacija blokova date veličine: 4KB, 16KB, 8KB, 4KB, 8KB.

b)(5) Nakon sekvence pod a), oslobođeni su blokovi na adresama A0000h, A1000h, A4000h i A2000h. Prikazati na isti način stanje ove strukture podataka nakon završetka svih ovih operacija dealokacije.

Rešenje:
a)

n	*Početne adrese (hex) slobodnih blokova veličine 2^n*
4	
3	
2	
1	
0	

b)

n	*Početne adrese (hex) slobodnih blokova veličine 2^n*
4	
3	
2	
1	
0	