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, septembar 2014.
Datum: 19.8.2014.

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 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, pri čemu vrednost u elementu `alloc[p][r]` ima sledeće značenje: -1 – proces *p* je zauzeo resurs *r* (grana zauzeća); 0 – proces *p* ne koristi i ne traži resurs *r*, niti je najavio njegovo korišćenje; 1 – proces *p* je najavio korišćenje resursa *r*, ali ga nije tražio (grana najave); 2 – proces *p* je tražio resurs *r*, ali ga nije dobio (grana potražnje);
- 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*; ako mu se resurs može dedeliti, on mu se i dodeljuje, a operacija vraća 1; ukoliko se resurs ne može dodeliti iz bilo kog razloga (već je zauzet ili zbog izbegavanja mrtve blokade), operacija vraća 0;
- operacija `release()` se poziva kada proces *p* oslobađa resurs *r*;
- kada neki proces oslobodi neki resurs, a nakon poziva operacije `release()`, okruženje će ponovo pozvati operaciju `request()` za svaki proces *p* i svaki resurs *r* na koji čeka taj proces *p*; tada se taj resurs možda može (a ne mora) dodeliti tom procesu u novonastaloj situaciji.

Implementirati preostale tri javne operacije ove klase.

```
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:
  int** alloc;
  int np, nr;
};

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

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem primenjuje aproksimaciju LRU algoritma zamene stranica u okviru istog procesa pomoću dodatnih bita referenciranja.

PMT procesa je organizovana u jednom nivou. Polje `pmt` u strukturi PCB ukazuje na PMT. PMT ima `PMTSIZE` ulaza tipa `unsigned int`. Jedan ulaz u PMT jednak je 0 ako se stranica ne može preslikati u okvir. U suprotnom, ulaz u PMT sadrži broj okvira (različit od 0), bite prava pristupa, kao i bit referenciranja koji je u najnižem razredu.

Dodatne bite referenciranja za svaki proces sistem čuva u posebnoj strukturi koja je organizovana kao *hash* tabela. Ova struktura dostupna je kao polje `pageRefHash` u PCB svakog procesa. Ova tabela preslikava ključ (broj stranice) u vrednost (registar dodatnih bita referenciranja). Dostupne su dve operacije ove strukture: `getValue(unsigned key)` koja vraća vrednost tipa `unsigned` za dati ključ, i operacija `setValue(unsigned key, unsigned value)` koja postavlja novu vrednost za dati ključ u *hash* tabeli. Konstanta `MAXINT` predstavlja najveću pozitivnu vrednost tipa `int`.

Realizovati operaciju:

```
void updateReferenceRegs (PCB* pcb);
```

koja treba da ažurira registre dodatnih bita referenciranja za proces sa datim PCB-om na osnovu bita referenciranja u PMT.

Rešenje:

3. (10 poena) Alokacija memorije za potrebe kernela

Kernel nekog operativnog sistema koristi *slab* alokator memorije za potrebe alokacije svojih struktura podataka. Ovaj alokator realizovan je šablonskom klasom `SlabAllocator` čija je delimična implementacija data dole. Ova šablonska klasa alocira strukture tipa `T`. *Slab* alokator koristi usluge *buddy* alokatora za alokaciju ploča veličine date parametrom šablona (`slabSize`). Ploče su u kešu alokatora uvezane u dvostruko ulančanu listu preko pokazivača `next` i `prev`. Glava i rep liste ploča su u `cacheHead` i `cacheTail`.

Implementirati operaciju `alloc()` klase `SlabAllocator` koja treba da alocira prostor za jednu instancu tipa `T` i vrati pokazivač na taj prostor.

```
template <class T, int slabSize>
class SlabAllocator {
public:
  SlabAllocator ();

  void* alloc ();
  void  free (void*);

private:
  struct Slot {
    char slot[sizeof(T)];
    int isFree;
    Slot () : isFree(1) {}
  };

  static const int numOfSlots = (slabSize-2*sizeof(Slab*))/sizeof(Slot);

  struct Slab {
    Slot slots[numOfSlots];
    Slab *next, *prev;
    void* operator new (size_t) { return BuddyAllocator::alloc(slabSize); }
    void  operator delete (void* p) { BuddyAllocator::free(p,slabSize); }
  };

  Slab *cacheHead, *cacheTail;
};

template<class T, int slabSize>
SlabAllocator<T,slabSize>::SlabAllocator () {
  cacheHead = cacheTail = new Slab();
}
```

Rešenje: