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 2022.
*Datum:* 4. 12. 2022.

*Drugi kolokvijum iz Operativnih sistema 2*

*Kandidat:* _____

*Broj indeksa:* _____ *E-mail:*_____

*Kolokvijum traje 90 minuta. 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

Ispravnost metode izbegavanja mrtve blokade Bankarevim algoritmom zasniva se, između ostalog, i na sledećim činjenicama:

a)(5) Početno stanje sistema u kom su svi resursi slobodni i svi procesi samo najavili svoje maksimalno korišćenje resursa mora biti bezbedno (*safe*). Definisati jednostavan uslov za to da je početno stanje bezbedno i dokazati da je taj uslov potreban i dovoljan.

b)(5) Kada u bezbednom stanju neki proces oslobodi neke resurse, sistem uvek prelazi u bezbedno stanje. Dokazati.

Rešenje:

## 2. (10 poena) Upravljanje memorijom

Za datu sekvencu referenciranja stranica i četiri raspoloživa okvira, odrediti broj straničnih grešaka (*page fault*) za dati algoritam zamene sgtranica. U prazna polja date tabele upisati brojeve stranica koje se nalaze u svakom okviru nakon obrade referencirane stranice, a posebno naglasiti (npr. zaokruživanjem broja stranice u sekvenci) ona referenciranja koja izazivaju straničnu grešku.

a)(5) LRU

| 1 | 2 | 3 | 4 | 5 | 6 | 3 | 4 | 2 | 5 | 6 | 1 | 2 | 3 | 5 | 6 | 5 | 6 | 4 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

Broj straničnih grešaka: _____

b)(5) Algoritam časovnika (*clock*). Kazaljka ukazuje na stranicu koja je sledeća na redu za zamenu. Iza broja stranice u datom polju upisati tačku (.) ako je njen bit referenciranja postavljen, a poziciju kazaljke označiti simbolom ↑ u polju za dati okvir (prikazati poziciju nakon obrade referenciranja stranice u sekvenci).

| 1 | 2 | 3 | 4 | 5 | 6 | 3 | 4 | 2 | 5 | 6 | 1 | 2 | 3 | 5 | 6 | 5 | 6 | 4 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

Broj straničnih grešaka: _____

## 3. (10 poena) Upravljanje memorijom

Neki sistem ima alokator paranjaka (*buddy*) koji realizuje klasa `Buddy` čija je delimična implementacija data dole. Alociraju se blokovi veličine $2^i$ stranica, gde je $0 \leq i <$ `BUCKET_SIZE`. Svaki ulaz *i* niza `bucket` sadrži `numOfBlocks[i]` elemenata koji čuvaju evidenciju o tome da li su blokovi veličine $2^i$ slobodni (`FREE`) ili nisu (`ALLOC`). Blok broj `j` veličine $2^i$ u ulazu `bucket[i]` deli se na parnjake `2*j` i `2*j+1` veličine $2^{i-1}$ u ulazu `bucket[i-1]`. Inicijalno je samo (jedini najveći) blok broj 0 u ulazu `bucket[BUCKET_SIZE-1]` označen kao slobodan, svi ostali nisu. Implementirane su sledeće operacije:

- `getFreeBlock(int size)`: vraća redni broj prvog slobodnog bloka u ulazu `bucket[size]`, ako takvog ima; u suprotnom, vraća -1;
- `setBlock(int size, int block, Alloc a)`: postavlja evidenciju za blok sa rednim brojem `block` u ulazu `bucket[size]` na slobodan ili ne;
- `getBlockAddr(int size, int block)`: vraća adresu bloka sa rednim brojem `block` u ulazu `bucket[size]`.

Implementirati operaciju

```
void* Buddy::alloc (int size);
```

koja alocira segment veličine `2^size`; ako nema slobodnog mesta, treba da vrati 0.

```
class Buddy {
public:
  Buddy (void* memory);
  void* alloc (int size);
  ...
protected:
  enum Alloc {FREE, ALLOC};
  int   getFreeBlock (int size) const;
  void  setBlock (int size, int block, Alloc a) { bucket[size][block] = a; }
  void* getBlockAddr(int size, int block) const;

private:
  void* mem;
  Alloc bucket[BUCKET_SIZE][MAX_BLOCKS];
  int numOfBlocks[BUCKET_SIZE];
};

Buddy::Buddy (void* p) : mem(p) {
  for (int i=BUCKET_SIZE-1, nblks=1; i>=0; i--, nblks*=2) {
    numOfBlocks[i] = nblks;
    for (int j=0; j<nblks; j++)
      bucket[i][j] = ALLOC;
  }
  bucket[BUCKET_SIZE-1][0] = FREE;
}

inline int Buddy::getFreeBlock (int size) const {
  for (int i=0; i<numOfBlocks[size]; i++)
    if (bucket[size][i]==FREE) return i;
  return -1;
}
```

Rešenje: