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, januar 2023.
Datum: . 1. 2023.

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

Dva izvrsna programera, Prle i Tihi, analiziraju programski kod dva uporedna procesa *X* i *Y* čija je struktura data dole, a koje je Prle napravio, a Tihi proverava. Ova dva procesa koriste zajednički nedeljiv resurs *A* kako je pokazano. Prijem poruke primitivom `receive` je sinhron. Tihi, kao iskusniji i ozbiljniji programer uočava sledeće: „Prle, zar ne vidiš da ovde *uvek* nastaje mrtva blokada jer ne mogu oba procesa da dođu do tačke susreta (randevua) u razmeni poruka? Promeni ovo sinhrono slanje iz procesa *X* u asinhrono slanje!“

Da li je Tihi u pravu sa tvrdnjom da ovo rešenje ima navedeni problem? Ako nije, objasniti zašto nije. Ako jeste, objasniti da li je promena koju predlaže potpuno ispravna ili nije i obrazložiti odgovor.

```
process X;
begin
  ...
  request(A);
  ...
  sync_send(Y,msg);
  ...
  release(A);
  ...
end;
```

```
process Y;
begin
  ...
  request(A);
  ...
  receive(X,msg);
  ...
  release(A);
  ...
end;
```

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem koristi algoritam davanja nove šanse (*second chance*) za zamenu stranica. Za alokaciju okvira i sprovođenje ovog algoritma zadužena je klasa `FrameAllocator` čija je delimična implementacija data dole. Jedan objekat ove klase vodi evidenciju o zadatom broju fizički susednih okvira memorije koji su inicijalno svi slobodni. Trenutno slobodni okviri ulančani su u jednostruko ulančanu listu tako da `headFree` ukazuje na indeks (redni broj) prvog slobodnog okvira u listi, dok element niza `next` predstavlja indeks sledećeg okvira u listi; krajnji element ima vrednost -1 u ovom polju. Alocirani okviri vezani su u kružnu, jednostruko ulančanu listu preko elemenata istog niza `next`, s tim da kazaljka `clockHead` ukazuje na sledeći kandidat za izbacivanje; inicijalno je ova lista prazna. Ukoliko je okvir sa indeksom *i* zauzet, element `pd[i]` ukazuje na objekat klase `PageDesc` koji predstavlja deskriptor stranice alocirane u taj okvir u PMT; operacije `getRefBit` i `clearRefBit` redom vraćaju, odnosno brišu vrednost bira referenciranja u deskriptoru stranice.

Implementirati operaciju `allocFrame` koja treba da vrati broj okvira koji se alocira. Ukoliko ima slobodnih blokova, treba vratiti jedan takav i ubaciti ga u kružnu listu zauzetih okvira. Ukoliko takvih nema, treba vratiti okvir stranice koja je izabrana za izbacivanje po ovom algoritmu.

```
class FrameAllocator {
public:
  FrameAllocator (int num) : numOfFrames(num), headFree(0), clockHead(-1) {
    for (int i=0; i<num-1; i++) next[i] = i+1, pd[i] = 0;
    next[num-1] = -1;
  }

  PageDesc* getPageDesc (int frame) const { return pd[frame]; }
  int allocFrame ();

private:
  int numOfFrames;
  int headFree, clockHead;
  int next[MAX_FRAMES];
  PageDesc* pd[MAX_FRAMES];
};
```

Rešenje:

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 veličine `PAGE_SIZE`, (u jedinicama (`sizeof(char)`) 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. Implementirati operaciju

```
void Buddy::free (void* addr, int size);
```

koja oslobađa blok veličine `2^size` na adresi `addr`. Ne treba proveravati ispravnost parametara.

```
class Buddy {
public:
  Buddy (void* memory);
  void* alloc (int size);
  void  free  (void* addr, 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:
  char* mem;
  Alloc bucket[BUCKET_SIZE][MAX_BLOCKS];
  int numOfBlocks[BUCKET_SIZE];
};

Buddy::Buddy (void* p) : mem((char*)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;
}

void* Buddy::alloc (int size) {
  int block = -1, current=size;
  for (; block<0 && current<BUCKET_SIZE; current++)
    block = getFreeBlock(current);

  if (block<0) return 0; // No available memory

  setBlock(--current,block,ALLOC);
  while (--current>=size) {
    block *= 2;
    setBlock(current,block+1,FREE);
  }

  return getBlockAddr(size,block);
}
```

Rešenje: