Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

*Predmet:* Operativni sistemi 1
*Nastavnik:* prof. dr Dragan Milićev
*Odsek:* Softversko inženjerstvo, Računarska tehnika i informatika
*Kolokvijum:* Integralni, februar 2026.
*Datum:* 13. 3. 2026.

# *Kolokvijum iz Operativnih sistema 1*

*Kandidat:* _____

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

*Kolokvijum traje dva sata. Dozvoljeno je korišćenje literature.*

| | | | |
|---|---|---|---|
| *Zadatak 1* | _____/10 | *Zadatak 3* | _____/10 |
| *Zadatak 2* | _____/10 | *Zadatak 4* | _____/10 |

**Ukupno:** _____/40

**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)**

Korišćenjem POSIX sistemskog poziva *mmap*, kao i dole datih sistemskih poziva, napisati program koji otvara ili pravi nov segment logički deljene memorije u koji potom upisuje niz od 256 celih brojeva 0..255 i potom završava izvršavanje. Ignorisati greške. Dati su potpisi i kratka objašnjenja potrebnih sistemskih poziva:

- `shm_open`: ima isti potpis kao i sistemski poziv *open* ili *sem_open*, sa parametrima koji imaju isto značenje; otvara ili pravi nov sistemski objekat deljene memorije sa datim simboličkim imenom i pravima pristupa;
- `ftruncate(int fd, size_t size)`: odmah nakon otvaranja objekta deljene memorije sa `shm_open`, potrebno je ovim pozivom definisati veličinu regiona deljene memorije;
- `shm_unlink`: ima isti poptis kao i poziv *close*; zatvara objekat deljene memorije.

Rešenje:

## 2. (10 poena)

Data je implementacija procedure `yield` za pocesor picoRISC koja na steku čuva kontekst tekuće niti na čiji PCB ukazuje pokazivač smešten u `oldRunning`, a sa steka restaurira kontekst novoodabrane niti na čiji PCB ukazuje pokazivač `newRunning`.

Proširiti ovu proceduru tako da podržava uslovno čuvanje i restauraciju registara za brojeve u pokretnom zarezu (*floating point*). Procesor poseduje 8 ovakvih registara, označenih sa `fp0..fp7`. Korisnička nit mora zatražiti korišćenje ovih registara postavljanjem bita u razredu 2 procesorske statusne reči PSW. Procedura `yield` ove registre treba da čuva, odnosno restaurira, ako i samo ako ih odgovarajuća nit koristi.

```
yield:      push  r0
            push  r1
            ...
            push  r31
            push  psw
            load  r0,oldRunning
            store sp,[r0+offsSP]

            load  r0,newRunning
            load  sp,[r0+offsSP]
            pop   psw
            pop   r31
            ...
            pop   r1
            pop   r0
            ret
```

Rešenje:

## 3. (10 poena)

U nekom sistemu implementira se keš blokova sa blokovskih uređaja kodom koji je dat u nastavku. Za svaki uređaj sa datim identifikatorom pravi se jedan objekat klase `BlockIOCache`, inicijalizovan tim identifikatorom, koji predstavlja keš blokova sa tog uređaja. Keš je kapaciteta `CACHESIZE` blokova veličine `BLKSIZE`. Keš je interno organizovan kao heš mapa `map` sa `MAPSIZE` ulaza. Svaki ulaz niza `map` sadrži glavu liste keširanih blokova koji se preslikavaju u taj ulaz. Funkcija `hash` je heš funkcija koja preslikava broj bloka u ulaz u nizu `map`. Glava liste, kao i pokazivač na sledeći element u listi čuvaju se kao indeksi elementa niza `entries` koji sadrži keširane blokove; vrednost -1 označava kraj (*null*). Svaki element niza `entries` je struktura tipa `CacheEntry` u kojoj je polje `blkNo` broj bloka koji je keširan u tom elementu, polje `next` ukazuje na sledeći element liste u istom ulazu, a polje `buf` je sadržaj samog bloka.

Na početku složene operacije sa uređajem, kod koji koristi keš najpre zahteva da potrebni blok bude učitan pozivom funkcije `getBlock` koja vraća pokazivač na niz bajtova u baferu – učitanom bloku. Pošto više ovakvih složenih operacija može biti pokrenuto uporedo, blok iz keša može biti izbačen (zamenjen drugim) samo ako ga više niko ne koristi, što se realizuje brojanjem referenci u polju `refCounter` strukture `CacheEntry`. Prikazana je implementacija funkcije `getBlock` koja treba da obezbedi da je traženi blok u kešu, odnosno učita ga ako nije.

Potrebno je implementirati pomoćnu funkciju `getFreeEntry` koja treba da vrati indeks slobodnog ulaza u nizu `entries` u koji se može učitati traženi blok u keš. Inicijalno je keš prazan i svi ulazi u njemu su slobodni. Ova funkcija treba redom da zauzima elemente niza `entries`, sve dok ima slobodnih. Kada slobodnih ulaza više nema, ona treba da izbaci blok (snimi ga na disk) u prvom ulazu koji je na redu po LRU redosledu (*least recently used,* najdavnije korišćen), ali samo pod uslovom da se blok u tom ulazu ne koristi (tj. njegov `refCounter` je nula). Ako to nije zadovoljeno, treba da proba sa sledećim skorijim u listi svih korišćenih ulaza uređenoj hronološki. Ako nema mesta u kešu jer nijedan blok ne može da se izbaci, treba vratiti -1. Ukoliko je potrebno dodati ili izmeniti članove ove klase, precizno navesti kako to treba uraditi. Na raspolaganju je i funkcija koja učitava blok, odnosno upisuje blok na dati uređaj:

```
void ioRead (int device, BlkNo blk, Byte* buffer);
void ioWrite(int device, BlkNo blk, Byte* buffer);

typedef unsigned char Byte; // Unit of memory
typedef long BlkNo; // Device block number
const unsigned BLKSIZE = ...; // Block size in Bytes

class BlockIOCache {
public:
  BlockIOCache (int device);
  Byte* getBlock (BlkNo blk);
  ...
protected:
  static int hash (BlkNo);
  int getFreeBlock ();
  void pullEntry (int hand);

private:
  static const unsigned CACHESIZE = ...; // Cache size in blocks
  static const unsigned MAPSIZE = ...; // Hash map size in entries

  struct CacheEntry {
    BlkNo blkNo; int next, lruNext, lruPrev, refCounter; Byte buf[BLKSIZE];
  };
```

```
  int dev;
  int map[MAPSIZE];  // Hash map
  CacheEntry entries[CACHESIZE]; // Cache
  int numOfBlocks = 0; // Number of used entries
  int lruHead = -1, lruTail = -1; // Head, tail of the LRU list of all used entries
  ...
};

// Pull the given entry to the top of the LRU list (the most recently used)
void BlockIOCache::pullEntry (int hand) {
  if (lruHead==hand) return;
  CacheEntry& entry = entries[hand];
  if (entry.lruPrev!=-1)
    entries[entry.lruPrev].lruNext = entry.lruNext;
  else lruHead = entry.lruNext;
  if (entry.lruNext!=-1)
    entries[entry.lruNext].lruPrev = entry.lruPrev;
  else lruTail = entry.lruPrev;
  entry.lruPrev = -1;
  entry.lruNext = lruHead;
  if (lruHead!=-1) entries[lruHead].prev = hand;
  else lruTail = hand;
  lruHead = hand;
}

Byte* BlockIOCache::getBlock (BlkNo blk) {
  // Find the requested block in the cache and return it if present:
  int entry = hash(blk);
  for (int i=map[entry]; i!=-1; i=entries[i].next)
    if (entries[i].blkNo==blk) {
      pullEntry(i);
      entries[i].refCounter++;
      return entries[i].buf;
    }
  // The block is not in the cache, find a free slot to load it:
  int free = getFreeEntry(entry);
  if (free==-1) return 0;  // Error: cannot find space
  // Load the requested block:
  entries[free].blkNo = blk;
  entries[free].refCounter = 1;
  entries[free].next = map[entry];
  map[entry] = free;
  pullEntry(free);
  ioRead(dev,blk,entries[free].buf);
  return entries[free].buf;
}
```

Rešenje:

**4. (10 poena)**

U nekom fajl sistemu blok je veličine 512 bajtova, a broj bloka je 32-bitan. Primenjuje se kombinovani indeksirani pritup alokaciji blokova. FCB zauzima jedan blok i sadrži 120 direktnih ulaza sa brojevima blokova sa sadržajem, kao i jedan *single indirect* i jedan *double indirect* ulaz. Popuniti sledeću tabelu upisujući ukupan broj alociranih indeksnih blokova, kao i blokova sa podacima za fajl sa sadržajem maksimalne veličine u ovom sistemu. Brojeve pisati kao stepene dvojke ukoliko je tako preglednije.

| Nivo indeksa | Broj alociranih blokova za indekse | Broj alociranih blokova za sadržaj |
|---|---|---|
| *Direct* | 1 (za sam FCB) | |
| *Single indirect* | | |
| *Double indirect* | | |