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, 4/2025.
Datum: 26. 10. 2025.

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)

Sistemskim pozivom *mmap* može se formirati logički segment virtuelne memorije koji proces deli sa svojom decom pokrenutom pozivom *fork* tako što se u flegovima uključi fleg `MAP_SHARED` (tada stranice ovog segmenta neće biti kopirane pri upisu). Ako je segment „anoniman“, tj. nije preslikan u sadržaj fajla, treba uključiti i fleg `MAP_ANONYMOUS`, a `fd` deskriptor postaviti na -1 (ofset je tada nebitan). Sistem će alocirati segment veličine zaokružene na ceo broj stranica, veći ili jednak traženoj veličini `size`. U slučaju neuspeha, ovaj sistemski poziv vraća `MAP_FAILED`.

```
Void* mmap(void* addr, size_t size, int prot, int flags,
           int fd, off_t offset);
```

U zaglavlju `bbuffer.h` definisana je šablonska klasa `BoundedBuffer<typename T, int size>` koja realizuje ograničeni bafer sa svom potrebnom sinhronizacijom i interfejsom kao što je dato na predavanjima. Elementi bafera su tipa `T`, a njegov kapacitet je `size`.

Korišćenjem sistemskih poziva *fork* i *wait* napisati program koji, kad se nad njim pokrene proces, pokrene jedan proces-dete sa kojim komunicira preko ograničenog bafera u deljenoj memoriji. Proces-dete je proizvođač koji u bafer stavlja redom znak po znak iz niza „Hello world“, uključujući i znak `'\0'`, a potom se završava. Proces-roditelj je potrošač koji iz bafera uzima znak po znak i ispisuje ih na svoj standardni izlaz, a potom čeka da se proces-dete završi i onda se i sam završava. Sve greške obraditi gašenjem procesa i ispisivanjem poruke na standardni izlaz za greške. Pomoć: u implementaciji se može koristiti koncept *placement new* jezika C++ kojim se objekat date klase može konstruisati (inicijalizovati pozivom konstruktora) u prostoru koji je već alociran i nalazi se na mestu na koje ukazuje `ptr`: `new (ptr) T(...)`.

Rešenje:

2. (10 poena)

U Školskom jezgru implementirana je statička operacija klase `Thread`

```
void Thread::yield (Thread* oldRunning, Thread* newRunning);
```

koja obavlja promenu konteksta oduzimajući procesor (tekućoj) niti na koju pokazuje parametar `oldRunning` i predajući procesor niti na koju pokazuje parametar `newRunning`. Korišćenjem ove operacije implementirati klasu `Event`, s tim da se uvek, pa i kod neblokirajućih poziva obavlja promena konteksta ukoliko raspoređivač odabere neku drugu nit za izvršavanje. U redu spremnih uvek se nalazi barem neka nit, makar nit *idle* koja ne radi ništa korisno (samo troši procesorsko vreme instrukcijama bez efekta). Klasa `Event` realizuje binarni semafor (događaj) bez ograničenja u pogledu toga koja nit sme da poziva operacije `signal` i `wait`.

Rešenje:

3. (10 poena)

U nekom sistemu implementira se keš blokova sa blokovskih uređaja („diskova“). Za svaki uređaj sa datim identifikatorom pravi se jedan objekat klase `BlockIOCache`, inicijalizovan tim identifikatorom, koji predstavlja keš blokova sa tog uređaja. Keš je interno organizovan kao heš mapa sa `MAPSIZE` ulaza. Za svaki ulaz `i` u mapi organizuje se niz `entries[i]` veličine `CACHESIZE` za keširane blokove veličine `BLKSIZE` koji se preslikavaju u taj ulaz `i` u heš mapi. Svaki element `map[i]` sadrži broj zauzetih prvih slotova, tj. keširanih blokova u ulazu `entries[i]`. Funkcija `hash` je heš funkcija koja preslikava broj bloka u ulaz u mapi. Svaki element niza `entries[i][j]` je struktura tipa `CacheEntry` u kojoj je polje `blkNo` broj bloka koji je keširan u tom elementu, a polje `buf` je sadržaj samog keširanog bloka.

Na početku složene operacije sa uređajem, kod koji koristi keš najpre traži da je potrebni blok učitan pozivom funkcije `getBlock` koja vraća pokazivač na niz bajtova u baferu – učitanom bloku. Pošto više ovakvih složenih operacija može biti ugnježđeno, 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`. Funkcija `evict(i)` izbacuje jedan keširani blok iz punog keša u nizu `entries[i]`, koji može da se izbaci jer nije u upotrebi, ukoliko takav može da nađe, oslobađa njegov ulaz i dekrementira `map[i]`, pa vraća njegov indeks u tom nizu `entries[i]`; ako takav ne može da nađe, ova funkcija vraća `CACHESIZE`.

Implementirati funkciju `getBlock` koja treba da obezbedi da je traženi blok u kešu, odnosno učita ga ako nije. Ako nema mesta u kešu jer nijedan blok ne može da se izbaci, treba vratiti *null*. Ostale članice date klase su implementirane, a na raspolaganju je i funkcija koja učitava blok na dati uređaj:

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

typedef unsigned char Byte; // Unit of memory
typedef unsigned long 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);
  void evict (int entry);
private:
  static const unsigned CACHESIZE = ...; // Cache entry size in blocks
  static const unsigned MAPSIZE = ...; // Hash map size in entries

  struct CacheEntry { BlkNo blkNo; int refCounter; Byte buf[BLKSIZE]; };

  int dev;
  int map[MAPSIZE];  // Hash map
  CacheEntry entries[MAPSIZE][CACHESIZE]; // Cache
};
```

Rešenje:

4. (10 poena)

U nekom binarnom fajlu zapisana je ogromna mapa koja preslikava ključeve u vrednosti (oba tipa `long`). Mapa je organizovana kao ulančana lista čvorova tipa `ListNode`. Ovi čvorovi složeni su u sadržaj fajla u proizvoljnom redosledu kao u niz, jedan iza drugog. Prvi element liste je uvek u ulazu 0 ovog niza, a na sledeći element u listi ukazuje polje `next` u čvoru (sadrži indeks sledećeg čvora u listi); vrednost 0 u polju `next` označava kraj liste. Svaki čvor sadrži `count` preslikavanja ključ u vrednost, a najviše njih `MAX_KEYS`. Ključevi su u nizu `keys`, a njihove odgovarajuće vrednosti u koje se preslikavaju u nizu `values`.

a)(5) Korišćenjem POSIX funkcije `read` za čitanje, realizovati funkciju `getNode` koja obezbeđuje da čvor smešten u dati indeks niza bude učitan u memoriju i vraća pokazivač na taj učitan čvor. Parametar `fd` ove funkcije je deskriptor već otvorenog fajla u kom je zapisana mapa. Ignorisati greške, tj. smatrati da su sadržaj fajla i struktura zapisana u njemu ispravni.

b)(5) Korišenjem realizovane funkcije `getNode`, implementirati funkciju `getValue` koja u opisanoj mapi koja je u već otvorenom fajlu sa deskriptorom `fd` pronalazi dati ključ i vraća vrednost u koju se taj ključ preslikava. Ukoliko ključ ne pronađe, funkcija treba da vrati -1.

```
const short MAX_KEYS = ...;
struct ListNode {
 long next;
 unsigned short count;
 long keys[MAX_KEYS];
 long values[MAX_KEYS];
};
ListNode* getNode (int fd, long index);
long getValue (int fd, long key);
```

R
e
š
e
n
j