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, jun 2021.
Datum: 13. 6. 2021.

Integralni kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 2 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)

Neki sistem ima segmentnu organizaciju memorije. Virtuelna adresa je 32-bitna, adresibilna jedinica je bajt, a maksimalna veličina fizičkog segmenta je 1 MB. Fizički adresni prostor je veličine 1 TB. Jedan ulaz u SMT-u sadrži prava pristupa *rwx* u najniža tri bita, granicu fizičkog segmenta (*limit* u opsegu od 0 do maksimalne veličine segmenta minus 1) u bitima do njih, a zatim fizičku adresu u bitima do njih; ovaj deskriptor (ulaz u SMT-u) zauzima najmanji potreban ceo broj bajtova.

Operativni sistem alocira segmente na zahtev, tako da pri kreiranju procesa ne alocira i ne učitava nijedan fizički segment. Vrednost 0 u polju za fizičku adresu u deskriptoru segmenta u SMT-u označava da preslikavanje nije moguće (segment nije alociran ili nije učitan).

Jedan alociran logički segment procesa opisan je deskriptorom tipa `SegDesc` u kom su, između ostalog, sledeća polja:

- `unsigned startAddr`: početna virtuelna adresa logičkog segmenta, svakako poravnata na početak fizičkog segmenta;
- `unsigned size`: veličina logičkog segmenta u bajtovima (može biti i veća od maksimalne veličine fizičkog segmenta);
- `unsigned short rwx`: prava pristupa za ceo logički segment u tri najniža bita.

Implementirati sledeću funkciju:

```
void initSegment (SegDesc* sd, unsigned long* smt);
```

Ovu funkciju poziva kod kernela kada inicijalizuje SMT novokreiranog procesa za svaki logički segment sa datim deskriptorom `sd`. Na već alociran SMT ukazuje `smt`. Veličine tipova su sledeće: `int` – 32 bita, `long` – 64 bita, `short` – 16 bita.

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 operacije `wait` i `signal` klase `Semaphore`, 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).

Rešenje:

3. (10 poena)

U nekom sistemu implementira se keš blokova sa blokovskih uređaja („diskova“) 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, a polje `buf` je sadržaj samog 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`.

Član `freeHead` je glava liste slobodnih elemenata u nizu `entries` (-1 ako slobodnih nema). Funkcija `evict` izbacuje jedan keširani blok iz punog keša, ukoliko takav može da nađe, oslobađa njegov ulaz i stavlja ga u listu slobodnih.

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

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

  int dev;
  int map[MAPSIZE];  // Hash map
  CacheEntry entries[CACHESIZE]; // Cache
  int freeHead; // Free entries list head
};
```

Rešenje:

4. (10 poena)

Na raspolaganju je POSIX API za fajlove:

```
int open (const char* path, int flags);
int close (int fd);
ssize_t read (int fd, void *buf, size_t count);
```

Tip `ssize_t` je označen celobrojni tip, isti kao `size_t`, samo što može da prihvati i vrednost -1. Funkcija `read` vraća stvarno učitan broj bajtova (0 ili više); ako je taj broj manji od `count`, stiglo se do kraja fajla. Funkcije `open`, `close` i `read` vraćaju -1 u slučaju greške.

Korišćenjem ovog interfejsa realizovati apstrakciju `IFStream` kao klasu čiji je interfejs dat dole. Ova klasa apstrahuje ulazni znakovni tok vezan za fajl. Staza do fajla zadaje se parametrom konstruktora. Zatvaranje fajla treba obezbediti destruktorom. Operacija `getc` vraća jedan znak učitan sa toka. Kada se došlo do kraja fajla, operacija `eof` vraća `true`. Ukoliko je u nekoj ranijoj operaciji sa tokom, uključujući i otvaranje, došlo do greške, funkcije `err` i `eof` vraćaju `true`. Ukoliko je tok u stanju greške ili se stiglo do njegovog kraja, operacija `getc` treba da vraća `'\-1'`. Učitavanje treba da bude što je moguće efikasnije u smislu da se sa uređaja učitava najmanji mogući broj blokova za sekvencijalni pristup. Veličina bloka u bajtovima je `BLOCK_SIZE`, a veličina znakova u bajtovima je `CHAR_SIZE`.

```
class IFStream {
public:
  IFStream (const char* path);
 ~IFStream ();

  bool eof () const;
  bool err () const;
  char getc ();

private:
  static const size_t BLOCK_SIZE = ..., CHAR_SIZE = 2;
};
```

Ova apstrakcija može se koristiti ovako:

```
IFStream str("myfile.txt");
while (!str.err() && !str.eof()) {
  char c = str.getc();
  ...
}
```

R
e
š
e
n
j