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, avgust 2020.
Datum: 30. 8. 2020.

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)

Školsko jezgro proširuje se konceptom koji služi za alokaciju i dealokaciju nedeljivog resursa u sistemu, ponaša se kao binarni semafor, a implementiran je klasom `Resource` čiji je interfejs dat u nastavku. Kada želi da zauzme neki deljeni resurs, nit treba da pozove operaciju `acquire` objekta ove klase koji služi za međusobno isključenje pristupa tom resursu, a kada ga oslobodi, treba da pozove operaciju `release`.

Da bi se sprečila pojava mrtve blokade (*deadlock*), koristi se algoritam „čekaj ili umri“ (*wait-die*) koji funkcioniše na sledeći način. Svakoj niti pri samom kreiranju pridružuje se jedinstvena „vremenska marka“ (ceo broj dostupan u atributu `Thread::timestamp`), tako da starije niti imaju manju vrednost ove marke. Kada nit *Ta* zahteva resurs koji drži zauzeta nit *Tb*, onda:

- Ako je *Ta* starija nego *Tb* (*Ta*<*Tb*), nit *Ta* čeka suspendovana dok ne dobije resurs i tada operacija `acquire` vraća 1 (istu vrednost ova operacija vraća i kada je resurs slobodan, onda ga nit odmah dobija);
- Ako je *Ta* mlađa nego *Tb* (*Ta*>*Tb*), niti *Ta* se odbija zahtev tako što operacija `acquire` vraća 0, a nit onda tu situaciju obrađuje na odgovarajući način (pokušava ponovo ili odustaje).

Implementirati u potpunosti klasu `Resource`.

```
class Resource {
public:
  Resource ();
  int  acquire();
  void release();
};
```

Rešenje:

2. (10 poena)

Neki sistem koristi segmentno-straničnu organizaciju memorije. Logičke segmente koje je proces alocirao sistem opisuje strukturama tipa `SegDesc` za svaki proces. Kako bi se pretraga za logičkim segmentom kom odgovara adresirana stranica koja je generisala straničnu grešku što efikasnije izvršila, ovi desktiptori organizovani su u uređeno binarno stablo za svaki proces, tako da su u levom podstablu svakog čvora segmenti koji zauzimaju niže adrese, a desno oni koji zauzimaju više adrese od segmenta datog čvora. Logički segment je uvek poravnat i zaokružen na stranice. U strukturi `SegDesc` polja `left` i `right` ukazuju na koren levog odnosno desnog podstabla, polje `pg` sadrži broj prve stranice logičkog segmenta, a polje `sz` sadrži veličinu segmenta izraženu u broju stranica. Tip `size_t` je neoznačen celobrojni tip dovoljno velik da predstavi veličinu virtuelnog adresnog prostora.

```
struct SegDesc {
  SegDesc *left, *right;
  size_t pg, sz;
  ...
};
```

a)(5) Implementirati operaciju koja pretragom u stablu na čiji koren ukazuje prvi parametar pronalazi i vraća deskriptor segmenta kom pripada data stranica, a *null* ako takvog nema:

```
SegDesc* findSegDesc (SegDesc* root, size_t page);
```

U strukturi PCB svakog procesa postoji polje `segdesc` tipa `SegDesc*` koje ukazuje na koren strukture stabla deskriptora logičkih segmenata tog procesa. U strukturi `SegDesc` postoji polje `vtp` (*virtual table pointer*) kao pokazivač na strukturu (tabelu) pokazivača na funkcije koje za dati tip logičkog segmenta implementiraju operacije koje kernel poziva u različitim situacijama. Ova struktura ima polje `loadPage` koje ukazuje na funkciju koja datu stranicu učitava u dati okvir i ima sledeći potpis (vraća negativnu vrednost u slučaju greške):

```
int loadPage (size_t page, size_t frame);
```

Postoji i globalna funkcija bez parametara `allocateFrame` koja alocira slobodan okvir u operativnoj memoriji i vraća njegov broj tipa `size_t`, a 0 u slučaju neuspeha. Konačno, postoji funkcija kernela koja postavlja vrednost ulaza u PMT procesa na čiji PCB ukazuje prvi parametar, za stranicu datu drugim parametrom, i u taj ulaz upisuje broj okvira dat trećim parametrom, a prava pristupa podešava prema zapisu u deskriptoru segmenta datom poslednjim parametrom:

```
void setPMTEntry (PCB* pcb, size_t page, size_t frame, SegDesc* sd);
```

b)(5) Implementirati internu funkciju kernela `handlePageFault` koju kernel poziva kada treba da obradi straničnu grešku procesa na čiji PCB ukazuje prvi parametar, generisanu za stranicu datu drugim parametrom. Ova funkcija treba da vrati 0 u slučaju uspeha, a različite negativne kodove u slučaju različitih grešaka. Ova funkcija ne treba da rukuje stanjem procesa (to rade drugi delovi).

```
int handlePageFault (PCB* pcb, size_t page);
```

Rešenje:

3. (10 poena)

Date su deklaracije pokazivača preko kojih se može pristupiti registrima kontrolera jednog ulaznog uređaja:

```
typedef unsigned int REG;
REG* ioCtrl =...;   // control register
REG* ioStatus =...; // status register
REG* ioData =...;   // data register
const unsigned BlockSize = ...; // block size
```

Uređaj je blokovski, što znači da jedna ulazna operacija uvek prenosi jedan blok iste veličine `BlockSize` reči, ali jednu po jednu reč preko registra za podatke. U upravljačkom registru najniži bit je bit *Start* kojim se pokreće prenos. Za svaki nov prenos jednog bloka potrebno je upisati 1 u bit *Start*, a na kraju prenosa bloka u taj bit upisati 0. U statusnom registru najniži bit je bit *Ready* koji signalizira spremnost jedne reči u registru za podatke, a bit do njega bit greške (*Error*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). U slučaju greške u prenosu, uređaj generiše isti prekid kao i u slučaju spremnosti za prenos novog podatka.

Zahtevi za ulaznim operacijama prenosa blokova podataka na ovom uređaju vezani su u jednostruko ulančanu listu. Zahtev ima sledeću strukturu:

```
struct IORequest {
  REG* buffer; // Data buffer (block)
  int status; // Status of operation
  IORequest* next; // Next in the list
};
```

Na prvi zahtev u listi pokazuje globali pokazivač `ioHead`. Kada u praznu listu kernel stavi prvi zahtev, pozvaće operaciju `transfer()` koja treba da pokrene prenos za taj prvi zahtev. Kada se završi prenos bloka zadat jednim zahtevom, potrebno je u polje `status` date strukture preneti status završene operacije (0 – ispravno završeno do kraja, -1 – greška), izbaciti obrađeni zahtev iz liste i pokrenuti prenos za sledeći zapis u listi. Ako zahteva u listi više nema, ne treba uraditi više ništa (kada bude stavljao novi zahtev u listu, kernel će proveriti i videti da je ona bila prazna, pa pozvati ponovo operaciju `transfer()` itd.)

Potrebno je napisati kod operacije `transfer()`, zajedno sa odgovarajućom prekidnom rutinom `ioInterrupt()` za prekid od uređaja, pri čemu prenos treba vršiti tehnikom programiranog ulaza-izlaza uz korišćenje prekida.

```
void transfer ();
interrupt void ioInterrupt ();
```

Rešenje:

4. (10 poena)

Na raspolaganju su sledeći standardni POSIX sistemski pozivi:

- `int close (int fd)`: zatvara fajl sa zadatim deskriptorom;
- `int write (int fd, const void *buffer, size_t size)`: upisuje dati sadržaj u fajl sa zadatim deskriptorom; ako je fajl otvoren sa postavljenim *O_APPEND*, onda se pre upisa tekuća pozicija implicitno pomera na kraj fajla i upis uvek vrši iza kraja fajla, proširujući sadržaj;
- `int open(const char *pathname, int flags, mode_t mode)`: otvara fajl sa zadatom putanjom; argument *flags* mora da uključi jedno od sledećih prava pristupa: *O_RDONLY, O_WRONLY,* ili *O_RDWR.* Ako je uključen i fleg *O_CREAT*, fajl će biti kreiran ukoliko ne postoji; ako je pritom uključen i *O_EXCL*, a fajl već postoji, funkcija će vratiti grešku (-1), a kod greške postavljen u globalnoj sistemskoj promenljivoj *errno* biće jednak *EEXIST*. Ako je uključen *O_TRUNC*, i ako fajl već postoji, njegov sadržaj se pri otvaranju briše. Argument *mode* definiše prava pristupa za fajl koji se kreira samo u slučaju da je uključen *O_CREAT* (tada je i obavezan), i to na sledeći način:

S_IRWXU	*0x0700 user (file owner) has read, write and execute permission*
S_IRUSR	*0x0400 user has read permission*
S_IWUSR	*0x0200 user has write permission*
S_IXUSR	*0x0100 user has execute permission*
S_IRWXG	*0x0070 group has read, write and execute permission*
S_IRGRP	*0x0040 group has read permission*
S_IWGRP	*0x0020 group has write permission*
S_IXGRP	*0x0010 group has execute permission*
S_IRWXO	*0x0007 others have read, write and execute permission*
S_IROTH	*0x0004 others have read permission*
S_IWOTH	*0x0002 others have write permission*
S_IXOTH	*0x0001 others have execute permission*

U nekom programu koristi se u uređeno binarno stablo sa čvorom tipa strukture `Node` sa celim brojevima kao sadržajem čvora (polje `contents` tipa `int`), tako da su u levom podstablu (polje `left` tipa `Node*`) svakog čvora brojevi koji su manji, a u desnom (polje `right`) oni koji su veći ili jednaki od sadržaja datog čvora. Implementirati funkciju koja u fajl sa datim imenom upisuje niz celih brojeva iz ovakvog stabla sa datim korenom, uređen neopadajuće (funkcija vraća 0 u slučaju uspeha, vrednost manju od 0 neku grešku). Ako fajl ne postoji, treba ga kreirati sa pravima na čitanje i upis za vlasnika, a samo na čitanje za sve ostale.

```
int save (const char* fname, Node* root);
```

Rešenje: