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, septembar 2020.
Datum: 20. 9. 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)

Dostupni su sledeći delovi školskog jezgra, kao i funkcije iz standardne biblioteke:

- `Thread::running`: statički član – pokazivač na objekat klase `Thread` koji predstavlja tekuću nit;
- `Thread::context`: nestatički član tipa `jmp_buf` u kom se čuva procesorski kontekst niti;
- `const size_t STACK_SIZE`: veličina prostora za stek niti (u jednicama `sizeof(char)`);
- `Thread::stack`: pokazivač tipa `void*` koji ukazuje na adresu početka dela memorije u kom je alociran stek niti;
- `jmp_buf::sp`: polje za sačuvanu vrednost registra SP; stek raste ka nižim adresama, a SP ukazuje na prvu slobodnu lokaciju veličine `sizeof(int)`;
- `Scheduler::put(Thread*)`: statička članica kojom se u red spremnih stavlja data nit;
- `void* malloc(size_t sz)`: standardna funkcija koja alocira prostor veličine `sz` (u jedinicama `sizeof(char)`); funkcija `free(void*)` oslobađa ovako alociran prostor;
- `memcpy(vodi* dst, const void* src, size_t size)`: kopira memorijski sadržaj veličine `size` sa mesta na koje ukazuje `src` na mesto na koje ukazuje `dst`;
- `setjmp()`, `longjmp()`: standardne bibliotečne funkcije.

Pomoću ovih elemenata implementirati sistemski poziv školskog jezgra – funkciju:

```
Thread* t_fork();
```

kojom se kreira nova nit kao „klon“ pozivajuće, roditeljske niti, sa istim početnim kontekstom, ali sa sopstvenom kontrolom toka, po uzoru na standardni sistemski poziv `fork()` za procese. U slučaju uspeha, u kontekstu roditeljske niti ova funkcija treba da vrati pokazivač na objekat niti deteta, a u kontekstu niti deteta treba da vrati 0; u slučaju greške, treba da podigne izuzetak tipa `ThreadCreationException`.

Rešenje:

2. (10 poena)

Neki operativni sistem podržava sledeće sistemske pozive za upravljanje nitima:

- `fork()`: kreira novu nit-dete kao klon roditelja, poput Unix sistemskog poziva *fork* (samo što kreira nit, a ne proces); u kontekstu roditelja vraća ID kreirane niti-deteta, a u kontekstu deteta vraća 0;
- `exit()`: gasi pozivajuću nit;
- `wait(int pid)`: suspenduje pozivajuću nit sve dok se nit dete te niti sa datim ID ne završi (ne ugasi se); argument 0 suspenduje pozivajuću nit sve dok se svi njeni potomci ne završe.

Korišćenjem ovih sistemskih poziva, realizovati operaciju `cobegin()` koja prima tri argumenta: niz pokazivača na funkcije, niz argumenata tipa `void*` i ceo broj *n*, a koja pokreće uporedno izvršavanje *n* niti nad funkcijama iz datog niza, pri čemu funkciju iz *i*-tog elementa niza pokazivača na funkcije poziva sa *i*-tim elementom u nizu argumenata, a vraća kontrolu pozivaocu tek kada se sve te niti (odnosno funkcije) završe. Obratiti pažnju na to da je nit koja poziva ovu funkciju `cobegin` možda ranije pokrenula neke druge niti koje ne treba čekati pre izlaska iz `cobegin`. Ignorisati sve moguće greške.

```
typedef void (*PF)(void*);

void cobegin (PF f[], void* af[], int n);
```

Rešenje:

3. (10 poena)

Neki sistem koristi segmentnu organizaciju memorije. Slobodne fragmente operativne memorije sistem opisuje strukturama tipa `FreeSegDesc` za svaki proces. Kako bi pretraga pri alokaciji memorije bila efikasnija, ovi desktiptori organizovani su u uređeno binarno stablo, tako da su u levom podstablu svakog čvora deskriptori slobodnih fragmenata koji su manji ili jednaki, a desno onih koji su veći od fragmenta datog čvora. U strukturi `FreeSegDesc` polja `left` i `right` ukazuju na koren levog odnosno desnog podstabla, polje `addr` sadrži početnu adresu fragmenta, a polje `sz` sadrži veličinu fragmenta. Tip `size_t` je neoznačen celobrojni tip dovoljno velik da predstavi veličinu adresnog prostora.

```
struct FreeSegDesc {
  FreeSegDesc *left, *right;
  size_t addr, sz;
  ...
};
```

Implementirati operaciju koja pretragom u stablu na čiji koren ukazuje prvi parametar pronalazi i vraća deskriptor slobodnog fragmenta u koji se može alocirati segment zadate veličine, a *null* ako takvog nema, i to radi primenom *first fit* algoritma alokacije. Ova operacija ne treba da ažurira ni stablo ni pronađeni deskriptor, već samo da pronađe i vrati pokazivač na odgovarajući deskriptor.

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

Rešenje:

4. (10 poena)

Neki fajl sistem primenjuje kombinovanu tehniku indeksirane alokacije sadržaja fajla. U FCB fajla polje `singleIndex` predstavlja direktni indeks kao niz od `SingleIndexSize` elemenata, pri čemu svaki element sadrži broj fizičkog bloka sa sadržajem fajla (ili 0, ako je neiskorišćen). Ako veličina sadržaja fajla preraste veličinu podržanu ovim indeksom, naredni blokovi sadržaja fajla (preko ove veličine) indeksiraju se indeksom u dva nivoa. Za te potrebe, polje `dblIndex` u FCB sadrži broj bloka na disku u kom je indeks drugog nivoa. Taj blok sa indeksom drugog nivoa sadrži `DblIndexSize` ulaza sa brojevima blokova sa sadržajem fajla, odnosno 0 ako ulaz nije iskorišćen. Polje `size` u FCB sadrži veličinu fajla u bajtovima. Veličina bloka je `BlockSize`. Svi ulazi u indeksu prvog nivoa i indeksu drugog nivoa (ako postoji) su sigurno postavljeni na validne vrednosti (broj bloka sa sadržajem ili 0).

Realizovati funkciju `truncateFile` koja treba da obriše sadržaj fajla čiji je FCB dat:

```
void truncateFile (FCB* fcb);
```

Na raspolaganju je funkcija koja vraća pokazivač na sadržaj bloka na disku sa datim brojem, učitanog u keš:

```
void* getBlock (PBlock blkNo);
```

kao i funkcija koja blok sa datim brojem označava slobodnim:

```
void freeBlock (PBlock blkNo);
```

Rešenje: