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: Drugi, jun 2019.
Datum: 19. 6. 2019.

Drugi kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 1,5 sat. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____% = _____/15

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)

U nekom sistemu postoje sledeći sistemski pozivi:

- `sem_t* sem_create(unsigned init_value)`: kreira semafor sa zadatom inicijalnom vrednošću;
- `int sem_wait(sem_t* semaphore)`
  `int sem_signal(sem_t* semaphore)`: standardne operacije *wait* i *signal* na zadatom semaforu;
- `int thread_create(void(*)(void*),void*)`: kreira nit nad funkcijom na koju ukazuje prvi argument; ta funkcija prima jedan argument tipa `void*` i ne vraća rezultat; novokreirana nit poziva tu funkciju sa stvarnim argumentom jednakim drugom argumentu ovog sistemskog poziva; sistemski poziv vraća PID kreirane niti.

Sve funkcije vraćaju negativnu vrednost ili *null* pokazivač u slučaju greške. Korišćenjem ovih sistemskih poziva, uz pomoću semafora za sinhronizaciju, napisati program koji kreira *N* niti koje će pozivati funkciju `process` bez argumenata jedna nakon druge, strogo u poretku numeracije tih niti (nit *i*+1 treba da pozove funkciju `process` kada nit *i* završi sa tim pozivom). Ignorisati sve greške.

Rešenje:

2. (10 poena)

Neki operativni sistem primenjuje kontinualnu alokaciju memorije i nudi sistemski poziv kojim proces može da zatraži „skupljanje“ prostora koji zauzima, tj. oslobađanje vršnog dela svog prostora koji je do sada zauzimao. Implementirati operaciju

```
int shrink (PCB* pcb, size_t by);
```

koja se koristi u implementaciji ovog sistemskog poziva i koja treba da smanji alociran memorijski prostor procesa na čiji PCB ukazuje prvi argument za veličinu datu drugim argumentom. Pri tom, ovu priliku sistem po potrebi koristi i da izvrši lokalnu defragmentaciju na sledeći način: ako je neposredno ispred memorije koju proces zauzima već postojao slobodan fragment, a kako je njegovim „skupljanjem“ iza njega sigurno nastao slobodan fragment, proces se relocira na početak slobodnog fragmenta ispred njega, kako bi se ova dva slobodna fragmenta spojila u jedan. U slučaju uspeha, ova funkcija treba da vrati 0, a u slučaju neke greške -1.

U strukturi PCB postoje, između ostalog, i sledeća polja:

- `char* base`: početna (bazna) fizička adresa procesa u memoriji;
- `size_t size`: veličina memorijskog prostora koji proces trenutno zauzima.

Na raspolaganju su i sledeće funkcije:

- `mem_free(void* at, size_t sz)`: oslobađa prostor na adresi datoj prvim argumentom i veličine date drugim argumentom, uz spajanje sa slobodnim fragmenatima ispred i iza novonastalog po potrebi;
- `relocate(PCB* pcb, void* to)`: relocira dati proces na novo zadato mesto, uz svo neophodno ažuriranje strukture za evidenciju slobodne memorije.

Slobodni fragmenti dvostruko su ulančani u listu na čiji prvi element ukazuje `fmem_head`. Fragmenti su ulančani u listu po rastućem redosledu svojih početnih adresa. Svaki slobodni fragment predstavljen je strukturom `FreeMem` koja je smeštena na sam početak tog slobodnog fragmenta:

```
struct FreeMem {
  FreeMem* next; // Next in the list
  FreeMem* prev; // Previous in the list
  size_t size;   // Size of the free fragment
};
```

Rešenje:

3. (10 poena)

Virtuelni adresni prostor nekog sistema je 4GB i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 8KB. PMT (*page map table*) je organizovana u dva nivoa, s tim da je broj ulaza u PMT prvog nivoa dva puta manji od broja ulaza u PMT drugog nivoa.

Posmatra se jedan proces kreiran nad sledećim programom:

```
#define N 0x1000
int src[N], dst[N];

int main () {
  for (int i=0; i<N; i++) dst[i] = src[i];
}
```

pod sledećim pretpostavkama:

- tip `int` je veličine 32 bita; promenljivu `i` je prevodilac fomirao kao registarsku promenljivu (njena vrednost se ne čuva u operativnoj memoriji);
- segment za kod ovog programa veličine je jedne stranice, a segment za stek je veličine 32 stranice;
- nizovi `src` i `dst` smešteni su jedan odmah iza drugog u segment memorije alociran za statičke podatke; ovaj segment je onoliki koliko je najmanje stranica potrebno za smeštanje ovih nizova;
- operativni sistem ne učitava nijednu stranicu pri kreiranju procesa, već sve stranice učitava tek na zahtev (*demand paging*).

a)(3) Prikazati logičku strukturu virtuelne i označiti veličinu svakog polja.

Odgovor:

b)(4) Koliko straničnih grešaka (*page fault*) će izazvati izvršavanje ovog procesa ako je operativni sistem za ovaj proces odvojio dovoljno okvira operativne memorije da smesti sve stranice koje taj proces adresira? Obrazložiti.

Rešenje:

c)(3) Ako je operativni sistem za ovaj proces odvojio samo 4 okvira operativne memorije, a za zamenu (izbacivanje) bira onu stranicu istog tog procesa koja je najdavnije učitana u memoriju, koja stranica će biti prva izbačena iz memorije i kojom stranicom će biti zamenjena (stranice imenovati kao *n*-te stranice segmenta za kod, stek ili podatke)? Obrazložiti.

Rešenje: