Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1 (SI2OS1, IR2OS1)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, april 2017.
Datum: 29.4.2017.

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)

Neki multiprocesorski operativni sistem sa preotimanjem (*preemptive*) dozvoljava preuzimanje tokom izvršavanja koda kernela, osim u kritičnim sekcijama, kada se pristupa deljenim strukturama podataka jezgra kojima mogu da pristupaju različiti procesori. Svaku takvu deljenu strukturu „štiti“ jedna celobrojna promenljiva koja služi za međusobno isključenje pristupa sa različitih procesora pomoću tehnike *spin lock*, dok se zabrana preuzimanja na datom procesoru obezbeđuje maskiranjem prekida. Međusobno isključenje pristupa kritičnoj sekciji, odnosno deljenoj strukturi, obavlja se pozivima operacija čiji je argument pokazivač na celobrojnu promenljivu koja „štiti“ datu deljenu strukturu:

```
void lock (short* lck);
void unlock (short* lck);
```

na ulazu, odnosno izlazu iz kritične sekcije. Pošto postoji potreba da kernel kod istovremeno pristupa različitim deljenim strukturama, kritične sekcije se mogu ugnježdavati. Zato demaskiranje prekida treba uraditi tek kada se izađe iz krajnje spoljašnje kritične sekcije (a ne pri svakom pozivu `unlock`).

Podaci koji su svojstveni svakom procesoru (svaki procesor ima svoju instancu ovih podataka) grupisani su u strukturu tipa `ProcessorPrivate`. (Ovu strukturu možete proširivati po potrebi.) Na raspolaganju su i sledeće funkcije ili makroi:

- `disable_interrupts()`: onemogućava spoljašnje prekide na ovom procesoru;
- `enable_interrupts()`: dozvoljava spoljašnje prekide na ovom procesoru;
- `test_and_set(short* lck)`: „obavija“ instrukciju procesora tipa *test-and-set*;
- `this_processor()`: vraća celobrojni identifikator tekućeg procesora (na kome se kod izvršava);
- `ProcessorPrivate processor_private[NumOfProcessors]`: niz struktura sa „privatnim“ podacima svakog procesora.

Implementirati operacije `lock` i `unlock`.

Rešenje:

2. (10 poena)

Neki operativni sistem primenjuje kontinualnu alokaciju memorije i nudi sistemski poziv kojim proces može da zatraži proširenje prostora koji zauzima. Implementirati operaciju

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

koja se koristi u implementaciji ovog sistemskog poziva i koja treba da proširi memorijski prostor procesa na čiji PCB ukazuje prvi argument za veličinu datu drugim argumentom. Ova funkcija treba da pokuša proširenje prostora procesa najpre bez njegove relokacije, ako je moguće (ukoliko je prostor iza njega slobodan i dovoljne je veličine), a potom i uz relokaciju, ako je to neophodno i moguće. U slučaju uspeha, ova funkcija treba da vrati 0, a u slučaju neuspeha -1.

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

- `void* 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_allocate_at(size_t sz, void* at)`: alocira prostor veličine date prvim argumentom na (fizičkoj) adresi datoj drugim argumentom; u slučaju uspeha vraća 0, a u slučaju neuspeha (na datoj adresi nije slobodan prostor potrebne veličine) vraća -1;
- `mem_allocate(size_t sz, void** location)`: alocira prostor veličine date prvim argumentom gde god je to moguće; u slučaju uspeha vraća 0, a adresu alociranog prostora smešta u pokazivač na koji ukazuje drugi argument; u slučaju neuspeha (nema slobodnog prostora potrebne veličine) vraća -1;
- `mem_free(void* at, size_t sz)`: oslobađa prostor na adresi datoj prvim argumentom i veličine date drugim argumentom;
- `mem_copy(void* from, void* to, size_t sz)`: kopira memorijski sadržaj veličine date trećim argumentom sa prve na drugu adresu.

Rešenje:

3. (10 poena)

Virtuelni adresni prostor nekog sistema je 16EB (eksabajt, $1E=2^{60}$) i organizovan je straničnо, adresibilna jedinica je bajt, a stranica je veličine 16KB. Fizički adresni prostor je veličine 1TB (terabajt). PMT (*page map table*) je organizovana u dva nivoa, s tim da su i broj ulaza, kao i širina ulaza u PMT prvog i drugog nivoa isti (PMT oba nivoa su iste veličine). PMT oba nivoa smeštaju se u memoriju uvek poravnate na fizički okvir, odnosno uvek počinju na početku okvira. Zbog toga se u ulazu prvog nivoa čuva samo broj okvira u kom počinje PMT drugog nivoa, dok se preostali biti do celog broja bajtova u ulazu ne koriste; vrednost 0 u svim bitima označava da preslikavanje nije dozvoljeno. U jednom ulazu PMT drugog nivoa čuva se broj okvira u koji se stranica preslikava i još 2 najniža bita koja koduju prava pristupa (00 – nedozvoljen pristup, stranica nije alocirana, 01 – dozvoljeno samo izvršavanje instrukcije, 10 – dozvoljeno samo čitanje podataka, 11 – dozvoljeno i čitanje i upis podataka), dok se ostali biti ne koriste. Jedan ulaz u PMT prvog i drugog nivoa zauzima minimalan, ali ceo broj bajtova. Sistem ne vrši učitavanje stranica na zahtev niti zamenu stranica, već su sve stranice procesa (odnosno sve one i samo one koje pripadaju logičkim segmentima virtuelne memorije koje je proces alocirao) učitane prilikom njegovog kreiranja i stalno su u memoriji do njegovog gašenja.

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

Odgovor:

b)(7) Implementirati sledeću funkciju:

```
int sharePage (unsigned* pmtFrom, unsigned long pageFrom,
      unsigned* pmtTo, unsigned long pageTo);
```

Ovu funkciju poziva kod kernela kada obrađuje sistemski poziv kojim jedan proces, na čiju PMT prvog nivoa ukazuje `pmtTo`, zatraži deljenje svoje stranice `pageTo` sa stranicom `pageFrom` drugog procesa, na čiju PMT prvog nivoa ukazuje `pmtFrom`, odnosno želi da svoju stranicu `pageTo` preusmeri na isti okvir koji već koristi stranica `pageFrom`. U slučaju da data stranica nije alocirana u `pmtFrom`, treba vratiti -1 kao signal greške. Pretpostavlja se da proces koji traži ovu uslugu (`pmtTo`) nije već imao alociranu datu stranicu `pageTo` (ona se alocira tek pri ovom pozivu).

Na raspolaganju je interna funkcija kernela `alloc_pmt` koja alocira jednu PMT u memoriji, popunjava sve njene ulaze nulama i vraća broj okvira u kom počinje ta alocirana PMT.
Veličine tipova su sledeće: `int` – 32 bita, `long` – 64 bita, `short` – 16 bita.

Rešenje: