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 2019.
Datum: 5. 5. 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)

Školsko jezgro proširuje se konceptom *mutex* koji predstavlja binarni semafor namenjen za međusobno isključenje pristupa kritičnim sekcijama ili deljenim resursima, ali uz mogućnost da nit može ugnežđeno „zauzimati“ (operacija *wait*) i „oslobađati“ (operacija *signal*) isti *mutex* na sledeći način:

- operaciju *signal* može da pozove samo nit koja je već zauzela *mutex* operacijom *wait*; u suprotnom, operacija *signal* vraća grešku;
- nit koja je već zauzela *mutex* operacijom *wait*, može ponovo izvršiti *wait* na njemu više puta, ali bez mrtve blokade sama sa sobom (samo prvi poziv *wait* slobodnog *mutex*-a ima efekta, ostali pozivi iz iste niti nemaju efekta); ova ista nit mora pozvati operaciju *signal* isti broj puta (upareno), tako da samo (poslednji) poziv operacije *signal* koji je uparen sa prvim efektivnim pozivom *wait* ima efekat oslobađanja *mutex*-a; u slučaju neuparenog poziva *signal* vratiti grešku.

Operacije *wait* i *signal* vraćaju celobrojnu vrednost, 0 u slučaju uspeha, negativnu vrednost u slučaju greške. Prikazati implementaciju klase `Mutex`, po uzoru na klasu `Semaphore` prikazanu na predavanjima (ne treba implementirati red čekanja niti, pretpostaviti da ta klasa postoji).

Rešenje:

2. (10 poena)

Neki program koristi dinamičko učitavanje. Proces uvek zauzima kontinualan deo svog virtuelnog adresnog prostora određene potrebne veličine, počev od virtuelne adrese 0. Za svaki modul predviđen za dinamičko učitavanje prevodilac u glavnom modulu programa organizuje sledeću strukturu – deskriptor tog modula:

```
struct ModDesc {
  char* base;  // Base address of the module
  size_t size; // Size of the module in sizeof(char)
  const char* name;  // Name of the module's file
};
```

U polju `base` je početna virtuelna adresa modula, ukoliko je taj modul učitan; ako modul još nije učitan, ovo polje ima vrednost *null*. Polje `size` definiše veličinu modula, a polje `name` naziv fajla sa sadržajem modula.

a)(5) Na jeziku C napisati pomoćnu funkciju `load_module` koja treba da proširi alocirani deo virtuelnog adresnog prostora procesa za veličinu datog modula i učita taj modul u to proširenje, pod sledećim pretpostavkama:

```
void load_module (ModuleDesc* mod);
```

- globalna promenljiva programa `mem_size` tipa `size_t` sadrži trenutnu veličinu zauzetog (alociranog) dela virtuelnog adresnog prostora (u odnosu na početnu virtuelnu adresu 0); ovaj prostor treba proširiti i modul učitati u to proširenje;
- `mem_extend(size_t)`: sistemski poziv koji proširuje alocirani deo virtuelnog adresnog prostora za zadatu veličinu;
- `load(const char* filename, char* addr)`: sistemski poziv koji na zadatu adresu `addr` u virtuelnom adresnom prostoru procesa učitava sadržaj fajla sa zadatim imenom;
- ignorisati greške u sistemskim pozivima (ukoliko ne može da izvrši uslugu traženu sistemskim pozivom, sistem gasi pozivajući proces).

b)(5) Na asembleru nekog zamišljenog jednostavnog RISC procesora napisati kod koji koristi opisanu funkciju `load_module`, a koji prevodilac generiše za svaki poziv nekog potprograma `fun` koji se nalazi u nekom modulu koji se dinamički učitava, pod sledećim pretpostavkama:

- svi registri i pokazivači su 32-bitni, kao i sva polja u strukturi `ModDesc`; adresibilna jedinica je bajt;
- pre izvršavanja traženog koda, stvarni argumenti potprograma `fun` već su stavljeni na stek, a u registru R0 je adresa deskriptora modula (`ModDesc*`) u kome se nalazi potprogram `fun` (ovo prevodilac zna u toku prevođenja);
- pomeraj (relativna adresa u odnosu na početak modula) potprograma `fun` je poznat prevodiocu u toku prevođenja; ovaj pomeraj označiti simboličkom konstantom `fun`.

Rešenje:

3. (10 poena)

Virtuelni adresni prostor nekog računara je 4GB i organizovan je stranično, adresibilna jedinica je bajt, a PMT je organizovana u dva nivoa. Broj ulaza u PMT oba nivoa je isti, kao i širina svakog ulaza koja je po 32 bita. PMT svakog nivoa zauzima tačno jednu celu stranicu.

Neki operativni sistem na ovom računaru zauzima najnižih 4MB raspoložive fizičke memorije za potrebe kernela. Kako ne bi menjao memorijski kontekst prilikom obrade sistemskih poziva, sistem preslikava (mapira) najviše stranice svakog kreiranog procesa u ovaj memorijski prostor kernela, s tim što postavlja indikatore u deskriptoru tih stranica u PMT tako da one nisu dostupne u neprivilegovanom (korisničkom) režimu rada procesora.

Kernel koristi tehniku *copy-on-write*. U implementaciji sistemskog poziva *fork*, sve PMT procesa roditelja i deteta, uključujući i PMT prvog nivoa, se ne kopiraju, nego se dele, sve dok je to moguće, odnosno dok kernel ne mora da ih razdvoji (kopira). Osim toga, pošto je deo memorije koji pripada kernelu deljen u virtuelnim adresnim prostorima svih procesa na istom mestu, one PMT koje se odnose na taj deo memorije svi procesi dele sve vreme.

a)(2) Kolika je veličina stranice i koliko ulaza ima PMT prvog nivoa? Prikazati postupak kojim se došlo do odgovora.

b)(3) Neki proces izvršio je sistemski poziv *exec* koji inicijalizuje memorijsku mapu tog procesa tako da, osim dela koji se preslikava u memoriju kernela, alocira jedan segment za kod veličine 1MB na početku svog virtuelnog adresnog prostora, jedan segment za podatke veličine 3MB odmah nakon segmenta koda, i segment za stek veličine 4MB odmah ispod memorijskog dela koji pripada kernelu. Koliko operativne memorije (osim one koja je prethodno već bila zauzeta zbog postojanja drugih procesa) je alocirano prilikom izvršavanja ovog sistemskog poziva za potrebe smeštanja PMT ovog procesa? Obrazložiti.

c)(2) Ovaj proces sada izvršava sistemski poziv *fork*. Koliko dodatne memorije (osim one koja je već bila zauzeta zbog postojanja ovog roditeljskog i drugih procesa) je dodatno alocirano za potrebe smeštanja PMT novog procesa deteta prilikom izvršavanja ovog sistemskog poziva? Obrazložiti.

d)(3) Posmatrani proces (roditelj) je potom odmah izvršio instrukciju poziva potprograma, što je uzrokovalo upisom u samo jednu stranicu na vrhu steka. Koliko dodatne memorije (osim one koja je već bila zauzeta pre toga) je dodatno alocirano za potrebe smeštanja PMT oba procesa, i roditelja i deteta zbog izvršavanja ove instrukcije? Obrazložiti.

Rešenje: