Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, januar 2018.
Datum: 15. 1. 2018.

Drugi kolokvijum iz Operativnih sistema 2

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 = _____%

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) Upravljanje deljenim resursima

Neki sistem koristi jednostavnu implementaciju grafa alokacije resursa (engl. *resource allocation graph*) na sledeći način:

- U svakom od `NProc` ulaza „tabele procesa“, tj. niza `processes`, nalazi se jedan PCB. Identifikator procesa (`pid`) je ulaz njegovog PCB u ovom nizu (`0..NProc-1`).
- U svakom od `NRes` ulaza „tabele resursa“, tj. niza `resources`, nalazi se jedan RCB (*resource control block*). Identifikator resursa (`rid`) je ulaz njegovog RCB u ovom nizu (`0..NRes-1`).
- Polje `rid` u PCB sadrži vrednost `rid` onog resursa na koji taj proces čeka, a -1 ako proces ne čeka ni na jedan resurs.
- Polje `pid` u RCB sadrži vrednost `pid` onog procesa koji je zauzeo taj resurs, a -1 ako je resurs slobodan.
- Operaciju `resource_allocate` poziva jezgro tokom obrade sistemskog poziva kojim proces `pid` traži resurs `rid`. Ako je resurs slobodan, operacija treba da ga zauzme i vrati 1. Ako je resurs zauzet, proces treba da čeka, a operacija treba da vrati 0. (Pozivalac ove operacije će potom suspendovati dati proces ili omogućiti njegovo dalje izvršavanje, u zavisnosti od ove vraćene vrednosti.)
- Operaciju `resource_free` poziva jezgro prilikom obrade sistemskog poziva kojim proces koji zauzima dati resurs `rid` oslobađa taj resurs. Ova operacija treba da vrati `pid` onog procesa koji je čekao na dati resurs, a koji treba deblokirati jer je dobio taj resurs, a -1 ako takvog procesa nema.

```
int resource_allocate (int pid, int rid);
void resource_free (int rid);
```

a)(7) Implementirati ove dve operacije korišćenjem samo datih struktura podataka.

b)(3) Koji problem ima dato rešenje? Predložiti način rešavanja tog problema.

Rešenje:

2. (10 poena) Mrtva blokada

Neki sistem primenjuje protokol pod nazivom *wait-die* („čekaj-umri“) za sprečavanje mrtve blokade. Svakom procesu dodeljena je „vremenska marka“ (u oznaci TS) koja predstavlja apsolutno vreme (trenutak) njegovog kreiranja (ne menja se do kraja života procesa). Kada proces *p* traži neki resurs koga već zauzima proces *q*, postupa se na sledeći način: ako je TS(*p*)<TS(*q*), tj. proces *p* je „stariji“ od procesa *q*, proces *p* će čekati dok ne dobije resurs; u suprotnom, procesu *p* se vraća greška, odnosno odbija se njegov zahtev za alokaciju resursa i taj proces mora ili da odustane od tog zahteva, ili da ponovo pokuša da zatraži resurs kasnije.

Klasa `Process` predstavlja jedan proces. U toj klasi polje `next` tipa `Process*` služi za ulančavanje u jednostruke liste, a polje `timestamp` sadrži vremensku marku procesa. Klasa `Resource` predstavlja jedan resurs. U njoj postoje sledeći članovi:

- `head` i `tail` tipa `Process*` predstavljaju glavu i rep jednostruko ulančane liste procesa koji čekaju na taj resurs;
- `usedBy` tipa `Process*` ukazuje na proces koji je zauzeo resurs (*null* ako je resurs slobodan);
- operacija `int Resource::allocate(Process* p)` traži zauzimanje resursa od strane datog procesa; ukoliko je resurs slobodan, operacija treba da zauzme resurs i da vrati 1; ukoliko pozivajućem procesu treba odbiti zahtev za alokaciju (sistemski poziv mu vraća grešku), operacija treba da vrati -1; ukoliko proces treba da čeka na resurs, operacija treba da smesti proces u red čekanja i da vrati 0; pozivalac ove operacije će suspendovati taj proces i izvršiti promenu konteksta.

a)(6) Implementirati operaciju `allocate`.

b)(4) Dokazati da opisani protokol sprečava mrtvu blokadu.

Rešenje:

3. (10 poena) Upravljanje memorijom

Neki sistem koristi model (aproksimacije) radnog skupa (engl. *working set*) da bi sprečio pojavu zvanu *thrashing.* Svakoj stranici koja je alocirana u virtuelnom prostoru procesa, bez obzira na to da li se trenutno nalazi u fizičkoj memoriji ili ne, pridružuje se registar dodatnih bita referenciranja (istorije bita referenciranja), koji se periodično ažurira pomeranjem udesno i upisom trenutne vrednosti bita referenciranja s leve strane. Da bi procenio veličinu radnog skupa procesa, sistem povremeno prebrojava sve stranice (bilo one trenutno u memoriji ili ne) koje u bitu referenciranja ili u nekom od tri najviša bita (najskorije) istorije imaju jedinicu. Skup tih stranica smatra (aproksimacijom) radnog skupa.

Sistem primenjuje straničenje u dva nivoa, virtuelna adresa je 32 bita, stranica je veličine 4 KB, adresibilna jedinica je bajt, a smeštanje višebajtnih reči u memoriju je po šemi niži bajt-niža adresa (*little endian*). PMT prvog i drugog nivoa imaju isti broj ulaza. Svaki ulaz u PMT prvog nivoa je veličine 32 bita i sadrži adresu PMT drugog nivoa (ili 0 ako je PMT drugog nivoa nealocirana). Svaki ulaz u PMT drugog nivoa, odnosno deskriptor stranice, sadrži dve 32-bitne reči. Nižu reč koristi hardver prilikom preslikavanja, dok je najviši bit više reči bit referenciranja, a ostali biti ove više reči su potpuno slobodni za korišćenje od strane operativnog sistema. Najviših 7 od tih bita kernel koristi za dodatne bite referenciranja (istoriju), dok ostale koristi za druge potrebe. Kernel preslikava svoj deo fizičke memorije u najviših 16 MB virtuelnog adresnog prostora svakog procesa, a prilikom izvršavanja kernel koda preslikavanje odgovara tekućem procesu (aktivan je memorijski kontekst tog procesa).

```
typedef unsigned int   uint32; // 32 bits
typedef unsigned short uint16; // 16 bits
typedef unsigned char  uint8;  // 8 bits
uint32 getWorkingSetSize (uint32* pmt);
```

Implementirati funkciju `getWorkingSetSize` koja na opisani način treba da izračuna veličinu (aproksimacije) radnog skupa procesa sa datom PMT prvog nivoa.

Rešenje: