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: Integralni, avgust 2021.
Datum: 28. 8. 2021.

Integralni kolokvijum iz Operativnih sistema 2

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)

U nekom veoma jednostavnom kernelu broj procesa je ograničen na `MAX_PROCESSES`, a strukture PCB čuvaju se u nizu `procs` te dimenzije. Svaki element ovog niza u polju `isUsed` tipa `bool` ukazuje na to da li je taj PCB „u upotrebi“ (da li predstavlja kreiran proces) ili nije (slot u nizu je slobodan). Skup spremnih procesa nije realizovan posebnom strukturom, već se stanje procesa vodi kao njegov atribut: u strukturi PCB polje `isRunnable` tipa `bool` ukazuje na to da li je dati proces izvršiv (spreman ili se izvršava). Raspoređivač je realizovan klasom `Scheduler` čija funkcija `get` treba da vrati PCB procesa izabranog za izvršavanje, a *null* ako izvršivog procesa nema. Implementirati ovu klasu ako je algoritam raspoređivanja:

a)(5) *round-robin*;

b)(5) po prioritetu; polje `pri` tipa `int` u PCB-u sadrži prioritet (niža vrednost - viši prioritet).

Rešenje:

2. (10 poena)

U nekom operativnom sistemu primenjuje se sledeći protokol sprečavanja mrtve blokade. Svaki proces dobija jedinstvenu vremensku marku (polje `timestamp` tipa `long` u strukturi PCB) u trenutku svog nastanka. Proces koji zahteva zauzeće nekog zauzetog resursa može da čeka na taj resurs samo ako je stariji (ima manju vremensku marku) od procesa koji taj resurs drži; u suprotnom, proces će dobiti grešku kao rezultat zahteva za zauzeće resursa.

Resurs je u kernelu implementiran klasom `Resource`. Implementirati njene operacije `allocate` i `release`. Operaciju `allocate` poziva kernel kada opslužuje zahtev procesa za alokaciju datog resursa. Ona treba da vrati: 1 ako je resurs bio slobodan, proces koji ga je tražio ga dobija i treba da nastavi izvršavanje; 0 ako je resurs zauzet, a proces treba da ga čeka i kernel treba da ga blokira (operacija `allocate` ne treba da radi blokiranje i promenu konteksta procesa, to radi ostatak kernela); -1 ako je resurs zauzet, a proces ne sme da ga čeka, već mu treba vratiti grešku. Operacija `release` treba da vrati pokazivač na PCB onog procesa koji eventualno treba deblokirati jer je čekao pa dobio resurs. Pokazivač `running` ukazuje na PCB tekućeg procesa čiji se sistemski poziv opslužuje. Na raspolaganju je klasa `Queue` koja implementira red pokazivača na PCB sa svojim operacijama:

- `isEmpty()`: vraća `true` ako je red prazan;
- `first()`: vraća prvi element u redu, bez njegovog izbacivanja iz reda;
- `removeFirst()`: vraća prvi element u redu i izbacuje ga iz reda;
- `insert(PCB*, long=0)`: ubacuje dati pokazivač na PCB u red, poštujući uređenje po celom broju datom kao drugi argument (ili na kraj ako je taj argument 0).

Red može biti neuređen ili uređen u oba poretka (rastući ili opadajući) u zavisnosti od parametra svoje konstrukcije.

Rešenje:

3. (10 poena)

Neki kernel štiti se od zaglavljivanja („batrganja“, *trashing*) praćenjem učestanosti straničnih grešaka (*page fault*) svakog procesa. Osim toga, kernel vodi rezervoar (*pool*) slobodnih okvira. Svakom procesu kernel određuje broj okvira koje bi smeo da koristi; ovaj broj vodi se u polju `allowedFrames` strukture PCB. Broj okvira koje proces stvarno koristi (koje zauzimaju njegove učitane stranice) zapisan je u polju `usedFrames` strukture PCB. Poseban deo kernela prati učestanost pojave straničnih grešaka i procesu koji ih pravi prečesto povećava broj dozvoljenih okvira (ali mu ih ne dodeljuje) prostim uvećanjem broja `allowedFrames`, a procesu koji ih pravi retko oduzima okvire, umanjuje njegov `allowedFrames` i `usedFrames` i tako oslobođene okvire stavlja u rezervoar slobodnih.

Implementirati funkciju `pageFault` koju kernel izvršava prilikom obrade stranične greške tekućeg procesa. Ova funkcija treba da obezbedi učitavanje tražene stranice (stranica je sigurno validna, tj. pripada alociranom segmentu, što je provereno pre poziva ove funkcije). Okvir za traženu stranicu treba obezbediti iz rezervoara slobodnih, ukoliko je to procesu dozvoljeno, odnosno preotimanjem neke druge stranice istog procesa, ako to nije slučaj.

```
void pageFault (PCB* pcb, size_t page);
```

Ova funkcija izvršava se u kontekstu pozivajućeg procesa (u istom toku kontrole i na istom steku), tako da kad ona pozove neki blokirajući potprogram, taj proces neće nastaviti izvršavanje dok se tok kontrole ne deblokira i blokirajući potprogram ne vrati kontrolu pozivaocu. Osim toga, izvršavanje ove funkcije je izolovano, pa tokom njenog izvršavanja nije potrebno brinuti o eventualnom utrkivanju: međusobno isključenje je obezbeđeno na višem nivou granularnosti, pre poziva ove funkcije. Na raspolaganju su sledeće funkcije; sve su sinhrone i potencijalno blokirajuće (gde je to potrebno):

- `size_t getFrameFromPool()`: vraća broj jednog slobodnog okvira iz rezervoara i izbacuje ga odatle; ukoliko je rezervoar prazan, blokira poziv dok se ne pojavi slobodan okvir preotimanjem od nekog procesa;
- `void getVictimPage(PCB*, size_t* page, size_t* frame)`: za proces sa zadatim PCB-om bira stranicu za izbacivanje po algoritmu zamene (*page replacement algorithm*) i njen broj upisuje u `*page`, a broj okvira koji ona zauzima upisuje u `*frame`;
- `void setPageUnavailable(PCB*, size_t page)`: u PMT-u datog procesa označava datu stranicu kao nedostupnu (nije dozvoljeno njeno adresiranje);
- `void setPageAvailable(PCB*, size_t page, size_t frame)`: u PMT-u datog procesa označava datu stranicu kao dostupnu (dozvoljeno je njeno adresiranje), tako da se preslikava u dati okvir;
- `void savePage(PCB*, size_t page, size_t frame)`: snima datu stranicu datog procesa iz datog okvira u prostor za zamenu (*swap space*);
- `void loadPage(PCB*, size_t page, size_t frame)`: učitava datu stranicu datog procesa iz prostora za zamenu u dati okvir.

Rešenje:

4. (10 poena)

Napisati kod servera koji održava stanje jednog računa. Stanje na računu se zadaje kao parametar konstruktora servera. Server treba da osluškuje klijente sve dok je stanje na računu veće od nule. Stanje na računu ne sme da bude negativno. Sredstva sa računa se povlače u dva koraka. U prvom koraku samo se rezervišu sredstva. U drugom koraku se sa računa povlače ona sredstva koja su rezervisana. Vrednost stvarno povučenih sredstava je manja ili jednaka rezervisanim u prvom koraku. Sredstva koja se ne povuku, oslobađaju se za dalje korišćenje. Prvi i drugi korak se realizuju pomoću različitih konekcija, tj. klijentskih priključnica. Server treba da ima mogućnost da obrađuje proizvoljan broj konekcija paralelno. U slučaju bilo koje greške server treba da odbije zahtev (npr. nema dovoljno sredstava na računu, pokušava se drugi korak, a nije urađen prvi, …). Server prihvata zahteve klijenata na portu 5555. Klijentski procesi se ne implementiraju, ali studenti mogu da uvedu pretpostavku kako rade.

Rešenje: