Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, decembar 2018.
Datum: 1. 12. 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) Mrtva blokada

Neki sistem sprečava mrtvu blokadu sledećom tehnikom. Svaki resurs ima svoj identifikator (tipa `unsigned`). Kada proces zatraži resurs, kernel interno poziva funkciju `resource_allocate`, koja treba da obavi alokaciju resursa i može imati sledeća tri ishoda:

- ako je resurs slobodan, on se svakako dodeljuje procesu i ova funkcija vraća 1;
- ako je resurs zauzet, a njegov identifikator je veći od svih drugih koje dati proces već drži zauzete, funkcija vraća 0, što znači da proces može i treba da čeka na taj resurs (suspenziju i promenu konteksta će obezbediti pozivajući kod kernela);
- ako je resurs zauzet, a njegov identifikator nije veći od svih drugih koje proces već drži zauzete, funkcija vraća -1, što znači da proces ne sme da čeka na ovaj resurs, radi sprečavanja mrtve blokade, pa će procesu iz sistemskog poziva biti vraćena greška (zahtev za alokaciju resursa biće odbijen).

Prikazati koje članove treba ugraditi u strukture PCB i RCB (*resource control block*) i implementirati sledeće funkcije koje kernel interno poziva za alokaciju, odnosno oslobađanje datog resursa od strane datog procesa; pritom, u druga dva slučaja, kao i u prvom slučaju ako proces alocira resurs čiji je identifikator veći od svih koje već drži zauzete, izvršavanje funkcije `resource_allocate`, kao i izvršavanje funkcije `resource_free` u svakom slučaju, treba da bude ograničenog vremena $O(1)$:

```
int  resource_allocate (PCB* p, RCB* r);
void resource_free     (PCB* p, RCB* r);
```

Rešenje:

2. (10 poena) Upravljanje memorijom

Za izbor stranice za zamenu neki sistem koristi algoritam *„nekorišćen nedavno“* (engl. *Not Recently Used*, NRU). Stranicama su pridruženi biti referenciranja i biti zaprljanosti (modifikacije) koje održava hardver. Sistem periodično briše bite referenciranja, tako da bit referenciranja ukazuje na to da li je stranica korišćena u poslednjem intervalu (nedavno). Za zamenu se bira *slučajna* stranica iz najniže grupe, ako takva postoji, pri čemu se stranice posmatraju klasifikovane po grupama na sledeći način:
0. nije korišćena, nije modifikovana
1. nije korišćena, jeste modifikovana (moguće je zato što je bit referenciranja obrisan)
2. korišćena, nije modifikovana
3. korišćena, modifikovana

Za evidenciju stranica sistem koristi jedan vektor `pages` čiji su elementi tipa `PageDescr` (deskriptori stranica). U svakom deskriptoru stranica postoji informacija kom procesu i kojoj njegovoj stranici pripada okvir na koji se odnosi ovaj deskriptor, kao i biti referenciranja i modifikacije prepisani iz PMT u bitima 1 i 0 polja `flags`, respektivno. U ovo polje sistem periodično prepisuje bite referenciranja i modifikacije iz PMT procesa, odnosno briše bite referenciranja kao što je navedeno.

```
struct PageDesc {
  PCB* pcb;
  unsigned long page;
  unsigned short flags;
};
const unsigned long NumOfPages = ...;
PageDesc pages [NumOfPages];
PageDesc* getVictim ();
```

Implementirati funkciju `getVictim` koja treba da vrati pokazivač na deskriptor stranice izabrane za zamenu.

Rešenje:

3. (10 poena) Upravljanje memorijom

Neki sistem koristi sistem ploča (engl. *slab*) za alokaciju memorije. Jedan keš predstavljen je objektom klase `Cache`. Podatak član `Cache::head` ukazuje na prvu ploču (`Slab`) u tom kešu. Svaka ploča (objekat klase `Slab`) na svom početku ima zaglavlje, a potom niz (`slots`) od `SlabSize` slotova (objekata klase `Slot`). Ploče u kešu ulančane su u jednostruku listu putem pokazivača `next`. U zaglavlju ploče je informacija o broju slobodnih slotova u ploči (`numOfFreeSlots`). Slobodni slotovi u slabu ulančani su u listu: indeks (u nizu `slots`) prvog slobodnog slota je dat u polju `Slab::head`, a na početku svakog slobodnog slota nalazi se indeks sledećeg slobodnog slota u listi (-1 za kraj). Konstruktor klase `Slab` propisno inicijalizuje celu ovu strukturu tako da sve slotove u ploči proglasi slobodnim i ulanča ih na odgovarajući način. Nova ploča alocira se dinamički, izrazom `new Slab(nextSlab)`.

Implementirati operaciju `Cache::alloc` koja treba da vrati slobodan slot, ali tako da se takav slot traži najpre u delimično popunjenoj ploči, ako takve nema onda u potpuno slobodnoj ploči, a ako ni takve nema, u novoj ploči koja se alocira i dodaje u keš.

```
class Slab {
public:
  Slab (Slab* next);
  static void* operator new (size_t); // May throw an exception (no memory)
private:
  unsigned numOfFreeSlots;
  long head;
  Slab* next;
  Slot slots[SlabSize];
};

class Cache {
public:
  Cache () : head(0) {}
  Slot* alloc ();
private:
  Slab* head;
};
```

Rešenje: