Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

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

Prvi 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)

Date su deklaracije pokazivača preko kojih se može pristupiti registrima dva izlazna uređaja:

```
typedef unsigned int REG;
REG* io1Ctrl =...;   // Device 1 control register
REG* io1Data =...;   // Device 1 data register
REG* io2Ctrl =...;   // Device 2 control register
REG* io2Data =...;   // Device 2 data register
REG* timer =...;     // Timer
```

U upravljačkim registrima bit 0 je bit *Start*. Prenos svakog pojedinačnog podatka na svaki od ovih uređaja zahteva se upisom podatka u registar podataka tog uređaja. Spremnost registra za podatke za prihvat novog izlaznog podatka prvi uređaj ne signalizira nikakvim signalom, ali je sigurno da je on spreman za novi najkasnije 50 ms nakon zadatog prethodnog zahteva (upisa podatka u registar). Spremnost registra za podatke za prihvat novog izlaznog podatka drugi uređaj signalizira prekidom.

Na magistralu računara vezan je i registar posebnog uređaja, vremenskog brojača. Upisom celobrojne vrednosti *n* u ovaj registar vremenski brojač počinje merenje vremena od *n* ms i, nakon isteka tog vremena, generiše prekid procesoru.

Potrebno je napisati kod, uključujući i obe prekidne rutine (od drugog uređaja i tajmera), koji vrši prenos po jednog bloka podataka na svaki od dva uređaja uporedo. Prenos se obavlja pozivom sledeće funkcije iz koje se vraća kada su oba prenosa završena:

```
void transfer (unsigned* blk1, int count1, unsigned* blk2, int count2);
```

Rešenje:

2. (10 poena)

U nekom asimetričnom multiprocesorskom operativnom sistemu jedan od procesora posvećen je samo obavljanju ulazno-izlaznih operacija koje su zahtevali korisnički procesi. Operacije sa svakim pojedinčanim ulazno-izlaznim uređajem obavlja po jedna interna kernel nit predstavljena objektom klase `IOThread`. Ove niti izvršavaju se samo na ovom posvećenom procesoru i nemaju nikakve veze sa ostalim nitima kernela niti korisničkim procesima (osim preuzimanja zahteva koje su oni postavili); na ovom procesoru izvršavaju se samo ove niti.

Svaka od tih niti ima sledeći generički oblik: ona uzme jedan zahtev za ulazno-izlaznom operacijom iz reda zahteva postavljenih za taj uređaj, pokrene operaciju sa uređajem na način specifičan za taj uređaj, a onda se suspenduje pozivom operacije `IOThread::suspend` dok uređaj ne signalizira spremnost za novu operaciju spoljašnjim prekidom. Prekidne rutine svih tih uređaja ne vrše promenu konteksta (preotimanje procesora), već samo postavljaju polje `IOThread::isReady` svoje niti na 1, čime ta nit ponovno postaje spremna i može da zada novu operaciju. Zbog svega ovoga nije potrebno raditi nikakvo maskiranje prekida niti međusobno isključenje sa ostalim procesorima.

Sve ove niti predstavljene su objektima klase `IOThread` u statičkom nizu `IOThread::allThreads`, a njihov broj je konstantan i iznosi `IOThread::NumberOfThreads`. Polje `IOThread::running` je pokazivač na nit koja se trenutno izvršava na ovom posvećenom procesoru. Na raspolaganju je funkcija:

```
void IOThread::yield (IOThread* oldThread, IOThread* newThread);
```

koja čuva kontekst tekuće niti u za to predviđeno polje objekta na koga ukazuje prvi argument i restaurira kontekst niti iz polja objekta na koga ukazuje drugi argument.

Implementirati operaciju `IOThread::suspend`. Za izvršavanje je dovoljno uzeti prvu (ili bilo koju drugu) spremnu nit iz niza `IOThread::allThreads`. Obratiti pažnju na to da je moguće da nijedna nit nije spremna za izvršavanje, jer su sve suspendovane i čekaju na završetak svojih operacija i prekide od svojih uređaja koji će ih ponovo učiniti spremnim; u tom slučaju procesor treba da uposleno čeka dok neka nit ne postane spremna.

Rešenje:

3. (10 poena)

U standardnom jeziku C++ postoje, između ostalog, sledeći elementi podrške za niti:

- `std::thread`: klasa kojom se predstavljaju niti; nova nit se pokreće odmah po kreiranju nekog objekta ove klase;
- `std::thread::thread`: konstruktor klase `std::thread` koji kreira nit nad funkcijom na koju ukazuje prvi argument; ta funkcija prima jedan argument i ne vraća rezultat; novokreirana nit poziva datu funkciju sa stvarnim argumentom jednakim drugom argumentu konstruktora;
- `std::thread::join`: suspenduje pozivajuću nit dok se ne završi nit čija se ova funkcija poziva;
- `thread_local`: specifikator životnog veka i načina alokacije koji se navodi u deklaraciji objekta; objekti ove kategorije životnog veka alociraju se kad god se kreira nova nit i nestaju sa završetkom te niti; svaka kreirana nit, uključujući i „glavnu“ nit kreiranu nad pozivom funkcije `main`, ima svoju sopstvenu kopiju (instancu) ovog objekta, različitu od instanci tog objekta u drugim nitima.

Precizno navesti šta sve može da ispiše sledeći C++ program (izlazne operacije ispisa na standardni izlaz su atomične):

```
thread_local int i=0;

void f (int id) {
  i=id;
  ++i;
  std::cout<<i;
}

void main() {
  i=9;
  std::thread t1(f,1);
  std::thread t2(f,2);
  std::thread t3(f,3);
  t1.join();
  t2.join();
  t3.join();
  std::cout<<i<<std::endl;
}
```

Rešenje: