Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, avgust 2024.
Datum: 1. 9. 2024.

Drugi kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 90 minuta. 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)

Dat je deo POSIX API-a za niti (*pthread*). Pomoću ovih funkcija napraviti dole dati objektno orijentisani API koji niti predstavlja kao objekte klase `Thread` sa značenjem funkcija kao u školskom jezgru i sa sinhronizacijom u destruktoru koja obezbeđuje da se objekat te klase neće uništiti sve dok se pripadajuća nit ne završi. Ignorisati greške.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void*(*thread_body)(void*), void *arg);
int pthread_join(pthread_t *thread, void*);

class Thread {
public:
   void start ();
protected:
   Thread();
   virtual ~Thread();
   virtual void run ();
};
```

Rešenje:

2. (10 poena)

Pravi se veoma jednostavno jezgro operativnog sistema za neki monoprogramski, jednoprocesorski ugrađeni (*embedded*) računar. Pošto taj računar treba da izvršava samo jedan, za njega namenjen i ugrađen program, ni procesor ni jezgro ne podržavaju virtuelne adresne prostore, zaštitu memorije, niti privilegovani režim rada procesora. Jezgro zato i ne podržava procese, ali podržava niti i njihovo uporedno izvršavanje. Svi sistemski pozivi su implementirani kao jednostavni pozivi jedinstvene C funkcije koja prima broj sistemskog poziva i pokazivač na strukturu sa argumentima tog poziva, koja potom poziva odgovarajuću funkciju za obradu konkretnog sistemskog poziva dinamičkim vezivanjem preko tabele vektora sistemskih poziva. Procesor i jezgro podržavaju i asinhrone spoljašnje hardverske prekide koje obrađuje jedinstvena prekidna rutina `int_handler`, a koja kao parametar dobija broj ulaza u IVT. Radi sprečavanja konflikata zbog asinhronih spoljašnjih prekida, na ulazu u obradu sistemskog poziva prekidi se maskiraju pozivom makroa `mask_interrupts`, a na izlazu demaskiraju pozivom makroa `unmask_interrupts`. Prekidne rutine i jedinstvene funkcije za obradu sistemskog poziva date su dole sa pripadajućim deklaracijama.

Preurediti dati deo jezgra tako da se za sprečavanje konflikata u jezgru ne koristi maskiranje prekida, već sledeća strategija: u prekidnoj rutini se samo beleži da se prekid dogodio, a nakon obrade sistemskog poziva, kada obrada prekida ne može da izazove konflikt, svi do tada novopristigli prekidi se obrađuju sinhronim pozivima odgovarajućih C funkcija preko tabele vektora prekida. Ne treba obezbediti i obradu prekida tokom izvršavanja koda korisničkih niti.

```
extern const unsigned short ivt_size; // IVT size
extern void(*ivt[ivt_size])(); // Interrupt vector table (IVT)
extern int(*scvt[])(void*); // System call vector table (SCVT)

interrupt void int_handler (unsigned short entry) {
  ivt[entry]();
}

int sys_call (int num, void* args) {
  mask_interrupts();
  int ret = scvt[num](args);
  Thread* oldT = Thread::running;
  Thread* newT = Scheduler::get();
  Thread::yield(oldT,newT); // Context switch from oltT to newT
  unmask_interrupts();
  return ret;
}
```

Rešenje:

3. (10 poena)

Pravi se softver za neki mali specijalizovani ugrađeni (*embedded*) računar sa malo RAM memorije i jednostavnim jezgrom operativnog sistema koje podržava POSIX niti i semafore. Ovaj računar treba da obrađuje podatke tipa `char` koje učitava sa nekog ulaznog uređaja kao tok znakova, a koji neprekidno stižu na ulazu, da ih obrađuje tako što transformiše svaki učitani znak u po jedan transformisan znak i da tako dobijene znakove isto tako, kao tok šalje na izlazni uređaj. Taj posao treba da obavlja bez prestanka.

Učitavanje podataka sa ulaznog uređaja radi funkcija `read_data` koja može da učita niz znakova zadate dužine na zadato mesto (u zadati bafer). Ova funkcija je napravljena i na raspolaganju je. Ona ulaznu operaciju obavlja prozivanjem uređaja (*polling*), uz mnogo uposlenog čekanja, a po potrebi uspavljuje pozivajuću nit na neko vreme ako ulazni podaci nisu duže raspoloživi jer je ulazni uređaj spor. Isto tako radi i data funkcija `write_data` koja prenosi niz znakova iz datog izlaznog bafera zadate veličine na izlazni uređaj. Obradu jednog niza znakova radi napravljena funkcija `process_data` koja te znakove čita iz datog ulaznog bafera i transformisani niz znakova iste dužine upisuje u dati izlazni bafer.

Zbog sporosti ulaznog i izlaznog uređaja i mnogo čekanja na njih, opisanu obradu treba organizovati tako da se opisane tri funkcije izvršavaju u tri uporedne niti koje sarađuju poput cevovoda (*pipeline*) razmenjujući podatke preko dva bafera, ulaznog i izlaznog, svaki veličine 256 znakova, a sinhronizuju se pomoću semafora, s tim da se ulazne i izlazne niti izvršavaju uporedo: dok ulazna nit učitava podatke u ulazni bafer, izlazna nit može da šalje prethodno obrađeni niz podataka iz izlaznog bafera na izlazni uređaj. Na raspolaganju su te funkcije:

```
extern void read_data (char* buffer, size_t size);
extern void write_data (const char* buffer, size_t size);
extern void process_data (const char* in_buf, char* out_buf, size_t size);
```

Napisati kompletan C/C++ program, sa svim potrebnim deklaracijama i funkcijom `main` koji obavlja opisani prenos i obradu. Ignorisati greške. Podsetnik na potpise POSIX funkcija vezanih za niti i semafore:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void*(*thread_body)(void*), void *arg);
sem_t *sem_open(const char *name,int oflag,mode_t mode,unsigned int value);
int sem_wait(sem_t *sem); int sem_post(sem_t *sem);
```

Rešenje: