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, april 2024.
Datum: 28. 4. 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)

Koristeći Unix sistemske pozive *fork, exit, wait* i *getpid* koji vraća *pid* pozivajućeg procesa, kao i funkciju *printf*, napisati C program koji, kada se nad njim pokrene (roditeljski) proces, pokreće *N* procesa dece, gde je *N* predefinisana konstanta. Svaki *i*-ti proces dete treba da se vrti u praznoj petlji *i**1.000.000 puta i potom završi. Nakon što pokrene sve procese – svoju decu, roditeljski proces treba da ih sačekuje da završe, jednog po jednog, u bilo kom redosledu, i da za svaki završeni proces dete ispiše njegov redni broj *i* i njegov *pid*. Kada sačeka da sva njegova deca završe, završava se i roditeljski proces.

Rešenje:

2. (10 poena)

U školskom jezgru promena konteksta implementirana je korišćenjem date funkcije `yield()`, u sistemskom pozivu `dispatch()` i na svim ostalim mestima na sličan način kao što je dato.

Potrebno je implementirati sistemski poziv (nestatičku funkciju članicu):

```
int Thread::join (int* status=nullptr);
```

kojim pozivajuća nit čeka (suspenduje se ako je potrebno) dok se ne završi nit dete na koje ukazuje `this` i potom vraća 0; ako `this` ukazuje na nit koja nije dete pozivajuće niti, pozivalac ne čeka ništa i ova funkcija vraća -1. Ukoliko je parametar `status` različit od *null*, u dati izlazni parametar treba upisati povratni status završene niti deteta.

Za te potrebe treba implementirati i sledeće nestatičke funkcije članice:

- `void Thread::created()`: poziva je jezgro interno za datu novokreiranu nit (`this`), kada je ta nit kreirana;
- `void Thread::completed(int status)`: poziva je jezgro za datu nit (`this`), kada se ta nit završila; celobrojni argument je dostavila završena nit kao svoj povratni status.

Pretpostaviti da se objekti klase `Thread` nikada ne uništavaju (ili barem ne uništavaju dok se ne završe i unište roditelji). Ukoliko proširujete klasu `Thread` novim članovima, precizno navedite kako.

```
Inline void yield (Thread* oldThr, Thread* newThr) {
  if (setjmp(oldThr->context)==0) longjmp(newThr->context,1);
}

void Thread::dispatch () {
  lock();
  Thread* oldThr = Thread::running;
  Scheduler::put(oldThr);
  Thread* newThr = Thread::running = Scheduler::get();
  yield(oldThr,newThr);
  unlock();
}
```

Rešenje:

3. (10 poena)

Korišćenjem školskog jezgra pravi se softver za kontrolu ulaza u neku javnu garažu. Kontrolu obavlja klasa `GarageController` čiji je interfejs dat dole. Objekat ove klase inicijalizuje se parametrom koji zadaje broj slobodnih mesta za parkiranje unutar garaže. Kada neki automobil dođe ispred rampe na ulazu garaže, odgovarajuća nit koja upravlja tom rampom poziva operaciju `entry` objekta ove klase. Ukoliko u garaži trenutno nema slobodnih mesta, ova operacija blokira pozivajuću nit dok se u garaži ne pojavi slobodno mesto. Ukoliko slobodnih mesta ima, ova operacija ažurira broj slobodnih mesta u garaži smanjujući ga za jedan i vraća kontrolu pozivaocu. Kada automobil prođe kroz izlaznu rampu, nit koja kontroliše tu rampu poziva operaciju `exit` objekta ove klase, čime se broj slobodnih mesta uvećava za jedan. Operacija `getFreeSlots` vraća broj trenutno slobodnih mesta u garaži.

Implementirati klasu `GarageController` u potpunosti korišćenjem semafora školskog jezgra.

```
class GarageController {
public:
  GarageController (unsigned slots);
 ~GarageController ();

  void entry ();
  void exit ();
  unsigned getFreeSlots () const;
};
```

Rešenje: