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: Prvi, jun 2019.
Datum: 19. 6. 2019.

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 ulazno/izlazna uređaja:

```
typedef volatile unsigned int REG;
REG* io1Ctrl =...;   // Device 1 control register
REG* io1Status =...; // Device 1 status register
REG* io1Data =...;   // Device 1 data register
REG* io2Ctrl =...;   // Device 2 control register
REG* io2Status =...; // Device 2 status register
REG* io2Data =...;   // Device 2 data register
```

U upravljačkim registrima najniži bit je bit *Start* kojim se pokreće uređaj, a u statusnim registrima najniži bit je bit spremnosti (*Ready*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`).

Potrebno je napisati funkciju `transfer` koja najpre vrši ulaz bloka podataka zadate veličine sa prvog uređaja korišćenjem tehnike prozivanja (*polling*), a potom izlaz tog istog učitanog bloka podataka na drugi uređaj korišćenjem prekida, i vraća kontrolu pozivaocu tek kada se oba prenosa završe.

Rešenje:

2. (10 poena)

Raspoređivanje i promena konteksta u školskom jezgru implementirani su na sledeći način:

- Svi objekti klase `Thread` smešteni su u statički vektor `Thread::allThreads` tipa `Thread[NumOfThreads]`.
- Nestatički podatak član `Thread::isRunnable` govori o tome da li je odgovarajuća nit spremna za izvršavanje ili se baš ona izvršava, ili nije (jer je suspendovana).
- Statički podatak član `Thread::running` tipa `int` ukazuje na onaj element niza `allThreads` koji predstavlja nit koja se trenutno izvršava (tekuća nit).
- Za ivršavanje se bira prva sledeća spremna nit u nizu `allThreads` iza one tekuće, i tako u krug.
- Nestatički podatak član `Thread::context` tipa `jmp_buf` čuva procesorski kontekst niti.
- Implementirana je funkcija `yield(jmp_buf oldC, jmp_buf newC)` koja čuva kontekst procesora u prvi argument i restaurira kontekst iz drugog argumenta.

Korišćenjem date funkcije `yield`, implementirati funkciju `dispatch` koja treba da preda procesor niti koja je data kao argument, pod uslovom da je taj argument dat i da je ta nit spremna. Ako je data nit spremna, funkcija treba da vrati 0; u suprotnom, tekuća nit treba da nastavi izvršavanje, a ova funkcija da vrati -1. Ako nije zadata nit, ova funkcija treba da preda procesor sledećoj spremnoj niti i da vrati 0.

```
int dispatch (Thread* newT = 0);
```

Rešenje:

3. (10 poena)

U nekom operativnom sistemu postoje sledeći sistemski pozivi:

- `int thread_create(void(*)(void*),void*)`: kreira nit nad funkcijom na koju ukazuje prvi argument; ta funkcija prima jedan argument tipa `void*` i ne vraća rezultat; novokreirana nit poziva tu funkciju sa stvarnim argumentom jednakim drugom argumentu ovog sistemskog poziva; sistemski poziv vraća PID kreirane niti;
- `void wait(int pid)`: suspenduje pozivajuću roditeljsku nit dok se ne završi nit-dete sa zadatim PID; ako je vrednost argumenta 0, pozivajuća nit se suspenduje dok se ne završe sve niti-deca.

Korišćenjem bibliotečne funkcije `memcpy`, implementirati funkciju `par_memcpy` istog potpisa koja će iskoristiti paralelizam na multiprocesorskom sistemu i velike segmente memorije kopirati paralelno. Ova funkcija će to uraditi tako što će kreirati potreban broj uporednih niti, a svaka nit će kopirati po jedan blok (particiju traženog segmenta memorije) veličine `BLOCK_SIZE` (poslednja nit možda manje od toga). Ignorisati eventualne greške i prekoračenja.

```
void* memcpy (void* destination, const void* source, size_t num);
void* par_memcpy (void* destination, const void* source, size_t num);
size_t BLOCK_SIZE = ...;
```

Rešenje: