Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1 (SI2OS1, IR2OS1)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Prvi, septembar 2015.
Datum: 4.9.2015.

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 jednog kontrolera ulaznog uređaja i jednog DMA kontrolera koji vrši prenos na jedan izlazni uređaj:

```
typedef unsigned int REG;
REG* ioCtrl =...;    // Input device control register
REG* ioStatus =...;  // Input device status register
REG* ioData =...;    // Input device data register
REG* dmaCtrl =...;   // DMA control register
REG* dmaStatus =...; // DMA status register
REG* dmaAddr =...;   // DMA buffer address register
REG* dmaCount =...;  // DMA buffer size register
const int BUFSIZE;
REG buffer[2][BUFSIZE];
```

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`).

Na jeziku C napisati kod operacije `transfer()` i prekidne rutine `dmaInterrupt()` za prekid sa DMA koje vrše prenos blokova podataka sa datog ulaznog uređaja tehnikom prozivanja (*polling*) na dati izlazni uređaj korišćenjem DMA na sledeći način.

Data su dva memorijska bafera za smeštanje blokova podataka, svaki veličine `BUFSIZE` (nizovi `buffer[2]`). Prvi blok podataka treba učitavati u prvi niz (`buffer[0]`), reč po reč, tehnikom prozivanja. Kada se ovaj bafer napuni, treba pokrenuti izlaznu operaciju za podatke iz ovog bafera preko DMA kontrolera, a ulaznu operaciju nastaviti uporednim učitavanjem u drugi niz (`buffer[1]`), čime ova dva bafera zamenjuju uloge. Kada se i ovo učitavanje završi, treba ponovo učitavati u prvi, a izlaz vršiti iz drugog bafera, odnosno zameniti uloge bafera itd. u nedogled. Pretpostaviti da izlazni prenos jednog celog bloka podataka preko DMA traje znatno kraće nego ulazni prenos jednog bloka, tako da će izlaz iz jednog bafera sigurno biti završen pre nego što se uporedni ulaz u drugi bafer završi. Sve eventualne greške ignorisati.

Rešenje:

2. (10 poena)

Školsko jezgro proširuje se konceptom tzv. *asinhronih signala*, koji podržavaju mnogi operativni sistemi, sa sledećim značenjem.

Kernel može „poslati“ *signal* nekoj korisničkoj niti. Signal je celobrojna konstanta u opsegu `1..SIGS-1`. Čim ta nit ponovo dobije procesor, umesto da odmah nastavi izvršavanje tamo gde je bila prekinuta, odnosno izgubila procesor, najpre će skočiti u proceduru za obradu tog signala, tzv. *signal handler*, pa kada se iz te procedure vrati, nastaviće dalje izvršavanje tamo gde je ono bilo prekinuto.

Na procedure za obradu signala pokazuju pokazivači u tabeli koja se nalazi u PCB svake niti, poput vektor tabele, u nizu `Thread::sigHandlers[SIGS]`. Za svaku nit, signalu *n* odgovara procedura za obradu na koju ukazuje pokazivač u ulazu `Thread::sigHandlers[`*n*`]` (ulaz 0 se ne koristi). Kada „šalje“ signal nekoj niti, kernel u PCB te niti postavi vrednost signala *n* u polje `Thread::signal`; vrednost različita od 0 u ovom polju ukazuje na postojanje poslatog signala, dok 0 znači da signala nema.

Dole je data implementacija sistemskog poziva `dispatch()` u školskom jezgru. Modifikovati ovu implementaciju tako da podrži obradu poslatog signala datoj niti. Smatrati da su na isti ovakav način realizovani i svi ostali sistemski pozivi i preuzimanja procesora.

```
void yield (jmp_buf old, jmp_buf new) {
  if (setjmp(old)==0)
    longjmp(new,1);
}

void dispatch () {
  lock();
  jmp_buf old = Thread::running->context;
  Scheduler::put(Thread::running);
  Thread::running = Scheduler::get();
  jmp_buf new = Thread::running->context;
  yield(old,new);
  unlock();
}
```

Rešenje:

3. (10 poena)

Neki operativni sistem podržava sledeće sistemske pozive za upravljanje nitima:

- `fork()`: kreira novu nit-dete kao klon roditelja, poput Unix sistemskog poziva *fork* (samo što kreira nit, a ne proces); u kontekstu roditelja vraća ID kreirane niti-deteta, a u kontekstu deteta vraća 0;
- `exit()`: gasi pozivajuću nit;
- `wait(int pid)`: suspenduje pozivajuću nit sve dok se nit dete te niti sa datim ID ne završi (ne ugasi se); argument 0 suspenduje pozivajuću nit sve dok se sva njena deca ne završe.

Korišćenjem ovih sistemskih poziva, realizovati operaciju `cobegin()` koja prima dva argumenta, pokazivače na dve funkcije, i koja pokreće uporedno izvršavanje dve niti nad te dve funkcije, a vraća kontrolu pozivaocu tek kada se obe te niti (odnosno funkcije) završe.

```
void cobegin (void (*f)(), void (*g)());
```

Rešenje: