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: Drugi, septembar 2013.
Datum: 30.8.2013.

Drugi 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	*Zadatak 4*	_____/10

Ukupno: _____/40 = _____% = _____/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)

U nekom operativnom sistemu postoje sledeći sistemski pozivi koji se odnose na klasične brojačke semafore za sinhronizaciju između uporednih procesa[1]:

- `sem_t* sem_open(const char* name, unsigned int value)` Otvara semafor sa datim simboličkim imenom `name` za korišćenje u pozivajućem procesu, ukoliko je semafor sa tim imenom već kreiran (od strane istog ili nekog drugog procesa). Ukoliko nije, kreira takav semafor sa datom početnom vrednošću `value`. Različiti procesi tako mogu da dele isti semafor. Struktura `sem_t` predstavlja ručku do semafora (deskriptor). U slučaju greške, vraća 0.
- `int sem_wait(sem_t* sem)` Operacija *wait* na datom semaforu. Vraća 0 u slučaju uspeha, -1 u slučaju greške.
- `int sem_post(sem_t* sem)` Operacija *signal* na datom semaforu. Vraća 0 u slučaju uspeha, -1 u slučaju greške.
- `int sem_unlink(sem_t* sem)` Oslobađa semafor od strane pozivajućeg procesa. Semafor se dealocira kada ga oslobode svi procesi koji ga koriste. Vraća 0 u slučaju uspeha, -1 u slučaju greške.

Sve navedene deklaracije nalaze se u `semaphore.h`.

Napisati program koji (u funkciji `main`) obezbeđuje međusobno isključenje kritične sekcije u odnosu na druge procese koji su na isti način konstruisani.

Rešenje:

[1] Ovo je nešto pojednostavljen standardni POSIX API.

2. (10 poena)

U nekom operativnom sistemu svi sistemski pozivi izvršavaju se kao softverski prekid koji skače na prekidnu rutinu označenu kao `sys_call`, dok se sama identifikacija sistemskog poziva i njegovi parametri prenose kroz registre procesora. Jezgro tog operativnog sistema je višenitno – poseduje više internih kernel niti koje obavljaju različite poslove: izvršavaju uporedne I/O operacije, vrše druge interne poslove jezgra itd. Prilikom obrade sistemskog poziva `sys_call`, prema tome, prekidna rutina samo oduzima procesor tekućem korisničkom procesu uz čuvanje njegovog konteksta i dodeljuje procesor tekućoj kernel niti.

Jedna od tih niti zadužena je za samu obradu zahteva koju je korisnički proces postavio u sistemskom pozivu. Ona preuzima parametre poziva iz sačuvanih registara i, na osnovu vrste sistemskog poziva, poziva odgovarajuću proceduru kernela za taj sistemski poziv.

Ovde se posmatraju sistemski pozivi za operacije na binarnim semaforima (događajima). Unutar kernela, binarni semafori su realizovani klasom `Event`, poput klase `Semaphore` date u školskom jezgru. Realizovati operacije `wait` i `signal` ove klase, koje poziva gore pomenuta kernel nit kada obrađuje te sistemske pozive. Na raspolaganju su operacije `lock()` i `unlock()` koje obezbeđuju međusobno isključenje unutar jezgra, kao i klasa `Scheduler` koja implementira raspoređivanje, kao u školskom jezgru. Na korisnički proces koji je izvršio sistemski poziv ukazuje pokazivač `runningUserThread` unutar kernela.

Rešenje:

3. (10 poena)

Data je jedna realizacija klase `DLArray` koja apstrahuje dinamički niz elemenata tipa `double` velike dimenzije zadate prilikom inicijalizacije parametrom `size`. U toj implementaciji, u svakom trenutku se u memoriji drži samo jedan keširani blok ovog niza veličine zadate parametrom `blockSize`. Implementacija koristi dinamičko učitavanje bloka u kome se nalazi element kome se pristupa uz zamenu (*swapping*), zapravo neku vrstu preklopa (*overlay*), tako da se u memoriji uvek nalazi samo jedan keširani blok kome se trenutno pristupa, dok se ceo niz nalazi u fajlu. Interfejs te klase je u njenom javnom delu.

U funkcijama `fread` i `fwrite` za pristup fajlu i učitavanje, odnosno snimanje datog niza elemenata tipa `double` na zadatu poziciju u fajlu, pozicija se izražava u jedinicama veličine binarnog zapisa tipa `double`, počev od 0, što znači da se fajl kroz ove funkcije može posmatrati kao veliki niz elemenata tipa `double`:

```
void fread (FHANDLE, int position, double* buffer, int bufferSize);
void fwrite(FHANDLE, int position, double* buffer, int bufferSize);
```

Modifikovati datu implementaciju ove klase (bez modifikacije njenog interfejsa) tako da se u memoriji nalaze uvek dva bloka iz niza u dva slota (umesto u jednom, kako je sada), pri čemu se u jedan slot preslikavaju parni, a u drugi neparni blokovi niza.

```
class DLArray {
public:
  inline DLArray (int size, int blockSize, FHANDLE fromFile);

  inline double get (int i);  // Get element [i]
  inline void   set (int i, double x); // Set element [i]

protected:
  inline void save();
  inline void load(int blockNo);
  inline void fetch(int blockNo);

private:
  FHANDLE file;
  int size, blockSize;
  int curBlock;
  int dirty;
  double* block;
};


DLArray::DLArray (int s, int bs, FHANDLE f) :
  file(f), size(s), blockSize(bs), curBlock(0), dirty(0) {
  block = new double[bs];
  if (block) load(curBlock);
}

void DLArray::save () {
  fwrite(file,curBlock*blockSize,block,blockSize);
  dirty=0;
}


void DLArray::load (int b) {
  curBlock = b;
  fread(file,curBlock*blockSize,block,blockSize);
  dirty = 0;
}
```

```
void DLArray::fetch(int b) {
  if (curBlock!=b) {
    if (dirty) save();
    load(b);
  }
}

double DLArray::get (int i) {
  if (block==0 || i<0 || i>=size) return 0; // Exception
  fetch(i/blockSize);
  return block[i%blockSize];
}

void DLArray::set (int i, double x) {
  if (block==0 || i<0 || i>=size) return; // Exception
  fetch(i/blockSize);
  if (block[i%blockSize]!=x) {
    block[i%blockSize]=x;
    dirty=1;
  }
}
```

Rešenje:

4. (10 poena)

Virtuelni adresni prostor nekog sistema je 16EB (eksabajt, $1E=2^{60}$) i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 16KB. Fizički adresni prostor je veličine 1TB (terabajt). PMT (*page map table*) je organizovana u dva nivoa, s tim da su i broj ulaza, kao i širina ulaza u PMT prvog i drugog nivoa isti (PMT oba nivoa su iste veličine). PMT oba nivoa smeštaju se u memoriju uvek poravnate na fizički okvir, odnosno uvek počinju na početku okvira. Zbog toga se u ulazu prvog nivoa čuva samo broj okvira u kom počinje PMT drugog nivoa, dok se preostali biti do celog broja bajtova u ulazu ne koriste; vrednost 0 u svim bitima označava da preslikavanje nije dozvoljeno. U jednom ulazu PMT drugog nivoa čuva se broj okvira u koji se stranica preslikava i još 2 bita koja koduju prava pristupa (00 – nedozvoljen pristup, stranica nije u memoriji, 01 – dozvoljeno samo izvršavanje instrukcije, 10 – dozvoljeno samo čitanje podataka, 11 – dozvoljeno i čitanje i upis podataka), dok se ostali biti ne koriste. Jedan ulaz u PMT prvog i drugog nivoa zauzima minimalan, ali ceo broj bajtova.

Kada sistem kreira nov proces, ne učitava inicijalno ni jednu njegovu stranicu, niti alocira ijednu PMT drugog nivoa, već samo alocira PMT prvog nivoa, čije sve ulaze inicijalizuje nulama. Stranice se potom dohvataju na zahtev, tokom izvršavanja procesa, kada se po potrebi alociraju i PMT drugog nivoa.

Odgovoriti na sledeća pitanja uz detaljna obrazloženja postupka.

a)(3) Prikazati logičku strukturu virtuelne i fizičke adrese i označiti veličinu svakog polja.

Odgovor:

b)(3) Koliko memorije minimalno zauzima PMT alocirana za proces prilikom njegovog kreiranja?

Odgovor:

c)(4) Koliko ukupno memorije zauzimaju sve PMT alocirane za proces koji je u dosadašnjem toku izvršavanja koristio najnižih 256GB svog virtuelnog adresnog prostora?

Odgovor:

Izrada: