Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Računarska tehnika i informatika, Softversko inženjerstvo
Kolokvijum: Drugi, avgust 2023.
Datum: 27. 8. 2023.

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)

a)(7) Korišćenjem Unix sistemskih poziva *fork, exit* i *wait/waitpid* implementirati funkciju `max` koja pronalazi maksimum celobrojnih vrednosti `val` sadržanih u čvorovima binarnog stabla predstavljenih strukturom `Node`, tako što proces koji poziva ovu funkciju sa argumentom koji ukazuje na koren datog stabla obrađuje taj koren i njegovo levo podstablo, a za obradu desnog podstabla kreira poseban proces koja teče uporedo sa obradom levog podstabla, i tako dalje rekurzivno za svako podstablo na isti opisani način. Ignorisati greške.

b)(3) Ako se ova funkcija pozove za koren potpunog balansiranog binarnog stabla sa *n* nivoa i 2^n - 1čvorova, koliko ukupno procesa obrađuje ovo stablo, uključujući i početni koji poziva ovu funkciju za koren ovog stabla? Precizno obrazložiti odgovor.

```
struct Node {
  Node *left, *right;
  int val;
};

int max (Node* nd);
```

Rešenje:

2. (10 poena)

a)(7) Korišćenjem standardnih funkcija `setjmp` i `longjmp` u školskom jezgru implementirati sledeće dve sistemske usluge (precizno navesti sva eventualna potrebna proširenja i napisati implementacije ovih funkcija):

- `void Thread::suspend()`: statička funkcija članica klase `Thread` koja bezuslovno suspenduje pozivajuću (tekuću) nit.
- `void Thread::resume()`: nestatička funkcija članica klase `Thread` koja reaktivira iz suspenzije nit za koju je pozvana, ukoliko je ta nit suspendovana sa `suspend()`; u suprotnom nema nikakvog efekta.

b)(3) Opisane usluge iskorišćene su za uslovnu sinhronizaciju dve niti na sledeći način:

- nit na čiji objekat klase `Thread` ukazuje pokazivač `t` i koja čeka na uslov izvršava:

  ```
  if (!condition) Thread::suspend();
  ```

- nit koja signalizira ispunjenje uslova izvršava:

  ```
  t->resume();
  ```

Da li je navedena upotreba ovih usluga ispravna? Precizno obrazložiti odgovor.

Rešenje:

3. (10 poena)

Klasa `Data` predstavlja neku strukturu podataka i poseduje konstruktor kopije. Objekti ove klase mogu se praviti operatorom `new` i brisati operatorom `delete`. Pokazivač `sharedData` je deljen između uporednih niti i ukazuje na deljeni objekat ove klase. Klasa `OptimisticCCtrl` implementira optimistički pristup kontroli konkurentnosti nad objektom na koga ukazuje deljeni pokazivač i namenjena je da se koristi na dole dat način. Svaki pokušaj transakcije izmene deljenog objekta mora da se započne pozivom operacije `startTrans` kojoj se dostavlja adresa pokazivača na deljeni objekat klase `Data`. Ova operacija vraća pokazivač na kopiju objekta nad kojim transakcija može da radi izmene (upis). Na kraju treba pozvati operaciju `commit` koja vraća `true` ako je uspela, `false` ako je detektovan konflikt; u ovom drugom slučaju transakcija mora da se pokuša iznova sve dok konačno ne uspe.

```
Data* sharedData = ...
OptimisticCCtrl* ctrl = new OptimisticCCtrl();

bool committed = false;
do {
  Data* myCopy = ctrl->startTrans(&sharedData);
  myCopy->write(...); // Write to *myCopy
  committed = ctrl->commit();
while (!committed);
```

Implementirati klasu `OptimisticCCtrl` čiji je interfejs dat dole. Na raspolaganju je sistemska funkcija `cmp_and_swap` koja radi atomičnu proveru jednakosti vrednosti pokazivača `*shared` i `read` i ako su oni isti, u `*shared` upisuje vrednost `copy`.

```
class OptimisticCCtrl {
public:
  OptimisticCCtrl ();

  Data* startTrans (Data** shared);
  bool  commit ();

};

bool cmp_and_swap(void** shared, void* read, void* copy);
```

R
e
š
e
n
j