Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, novembar 2024.
Datum: 30. 11. 2024.

Drugi kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 90 minuta. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____%

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) Mrtva blokada

Neki sistem izbegava mrtvu blokadu primenom bankarevog algoritma. Dole je dato trenutno stanje sistema. Svi procesi su najavili maksimalno korišćenje po 6 instanci svakog od tri tipa resursa. Matrica *Request* evidentira zahteve na čekanju koji nisu mogli da budu zadovoljeni, bilo zbog toga što nije bilo dovoljno slobodnih resursa ili zbog izbegavanja mrtve blokade, pa su procesi koji su ih postavili suspendovani. Počev od datog stanja izdati su sledeći zahtevi datim redom:

P4: *request*(4, 2, 3), *P3*: *release*(2, 3, 0).

Prikazati stanje sistema (bez matrice *Max*) nakon obrade svakog od ovih zahteva. Dati sva rešenja.

Allocation

	A	*B*	*C*
P1	1	3	2
P2	2	3	3
P3	3	4	1
P4	2	2	3

Request

	A	*B*	*C*
P1	4	3	4
P2	0	0	0
P3	0	0	0
P4	0	0	0

Available

A	*B*	*C*
4	3	5

Rešenje:

2. (10 poena) Virtuelna memorija

Dole su date dve varijante istog dela programa koje na različite načine rade istu obradu četiri statički alocirana trodimenzionalna celobrojna niza `a`, `b`, `c` i `d`. Pretpostavke su sledeće:

- Dati trodimenzionalni nizovi su svi istih dimenzija i svaki od njih zauzima u memoriji tačno *n* celih susednih stranica i počinje od početka stranice.
- Dimenzije nizova `MAX_2`, `MAX_1` i `MAX_0` su konstantni izrazi vrednosti poznatih prevodiocu za vreme prevođenja.
- Brojače petlji `i2`, `i1` i `i0` prevodilac će alocirati u registre, ne u memoriju.
- Sve instrukcije datog dela programa smeštene su u istu stranicu koja je ranije već učitana u memoriju i zaključana, tako da neće biti zamenjivana tokom izvršavanja.
- Procesor je RISC sa *load/store* arhitekturom i dovoljnim brojem registara.

a)(5) Koliko stranica zauzima lokalitet procesa koji izvršava ovaj deo programa u svakom od dva navedena slučaja? Precizno obrazložiti odgovor.

b)(5) Pod uslovom da je ovom procesu dodeljeno tačno onoliko okvira koliko obuhvata njegov lokalitet (uključujući i jednu zaključanu stranicu sa instrukcijama), a da se stranice sa podacima u kojima su smešteni delovi ovih nizova zamenjuju po FIFO algoritmu i nijedna inicijalno nije učitana u memoriju, koliko straničnih grešaka izaziva proces koji izvršava ovaj deo programa u svakom od dva navedena slučaja? Precizno obrazložiti odgovor.

1)

```
#define max(x,y) ((x)>=(y)?(x):(y))

for (int i2=0; i2<MAX_2; i2++)
  for (int i1=0; i1<MAX_1; i1++)
    for (int i0=0; i0<MAX_0; i0++)
      a[i2][i1][i0] += b[i2][i1][i0];

for (int i2=0; i2<MAX_2; i2++)
  for (int i1=0; i1<MAX_1; i1++)
    for (int i0=0; i0<MAX_0; i0++)
      a[i2][i1][i0] *= c[i2][i1][i0];

for (int i2=0; i2<MAX_2; i2++)
  for (int i1=0; i1<MAX_1; i1++)
    for (int i0=0; i0<MAX_0; i0++)
      a[i2][i1][i0] = max(a[i2][i1][i0],d[i2][i1][i0]);
```

2)

```
#define max(x,y) ((x)>=(y)?(x):(y))

for (int i2=0; i2<MAX_2; i2++)
  for (int i1=0; i1<MAX_1; i1++)
    for (int i0=0; i0<MAX_0; i0++) {
      a[i2][i1][i0] += b[i2][i1][i0];
      a[i2][i1][i0] *= c[i2][i1][i0];
      a[i2][i1][i0] = max(a[i2][i1][i0],d[i2][i1][i0]);
    }
```

Odgovori:

3. (10 poena) Alokacija memorije

U jezgru nekog operativnog sistema primenjuje se sistem ploča (*slab allocator*) za alokaciju struktura za potrebe jezgra. Za alokaciju nove ploče kada u kešu više nema slobodnih slotova i samog keša koristi se niži sloj koji implementira neki drugi alokator. U nastavku su date delimične deklaracija i definicije klasa `Cache` i `Slab` i implementacije nekih njihovih operacija. Slotovi za alokaciju su tipa `X`. Metode `lock()` i `unlock()` služe za međusobno isključenje u više procesorskom sistemu prilikom pristupa objektima klasa `Cache` i `Slab`.

U cilju povećanja performansi alokacije u više procesorskom sistemu pored datog keša koristi se i po jedan lokalni pomoćni keš za svaki procesor. Pomoćni keš sadrži uvek samo jednu ploču. Alokacija se uvek vrši iz pomoćnog keša, dok globalni keš (objekat klase `Cache`) sadrži sve ploče koje nisu lokalne. Slot koji se briše može da se nalazi u bilo kojoj ploči (i lokalnoj i globalnoj).

a)(7) Implementirati klasu `CacheLocal` čija je delimična deklaracija:

```
class CacheLocal {
public:
    X* alloc();
};
```

b)(3) Broj procesora u sistemu je predefinisan konstantom `CPU_NUM`, dok se redni broj procesora na kome se izvršava nit može dobiti pomoću funkcije `getCpuId()`. Implementirati funkciju koja je zadužena da alocira jedan objekat tipa `X` u celom sistemu:

```
  X* alloc();

class Slab {
public:
  Slab ();
  static void* operator new (size_t s) { return kernel_alloc(s); }
  void lock();
  void unlock();
  X* alloc();
  Slab* getNext() { return nextSlab; }
  void setNext(Slab *slab) { nextSlab = slab; }
private:
  Slab* nextSlab;
};

class Cache {
public:
  X* alloc();
  void addSlab(Slab* newSlab);
  static void* operator new (size_t s) { return kernel_alloc(s); }
  void lock();
  void unlock();

private:
   Slab* headSlab;
};

X* Cache::alloc() {
    X* ret = 0;
    lock();
    for (Slab* cur = headSlab; cur != 0; cur = cur->getNext()) {
        cur->lock();
        ret = cur->alloc();
```

```
        cur->unlock();
        if (ret != 0) break;
    }
    if (!ret) {
        Slab* newSlab = new Slab();
        if (newSlab != 0) {
            ret = newSlab->alloc();
            addSlab(newSlab);
        }
    }
    unlock();
    return ret;
}

void Cache::addSlab(Slab* newSlab) {
    newSlab->setNext(headSlab);
    headSlab = newSlab;
}
```

Rešenje: