Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, januar 2022.
Datum: 20. 1. 2022.

Drugi kolokvijum iz Operativnih sistema 2

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 = _____%

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 5 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), *P1*: *release*(2, 3, 0).

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

Allocation

	A	*B*	*C*
P1	2	3	0
P2	1	2	2
P3	0	2	1
P4	1	1	2

Request

	A	*B*	*C*
P1	0	0	0
P2	0	0	0
P3	5	3	4
P4	0	0	0

Available

A	*B*	*C*
6	2	5

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem primenjuje lokalnu zamenu stranica algoritmom davanja nove šanse (*second chance*). Evidencija okvira vodi se u statičkom nizu `ProcessFrames::frames`, čiji je svaki element tipa strukture `FrameDesc`. Svakom okviru broj *i* fizičke memorije (u opsegu `0..NUM_FRAMES-1`) odgovara *i*-ti element ovog niza. Polje `page` u strukturi `FrameDesc` sadrži broj stranice koja je smeštena u dati okvir. Svakom procesu pridružen je jedan objekat klase `ProcessFrames` koji implementira operacije potrebne za zamenu stranica datog procesa. Elementi niza `frames` koji predstavljaju okvire datog procesa ulančani su dvostruko u kružnu listu tako što polja `prev` i `next` strukture `FrameDesc` sadrže indekse u nizu `frames` prethodnog, odnosno sledećeg elementa u listi. (Svaki proces ima svoju kružnu listu za zamenu sopstvenih stranica.) Atribut `cursor` sadrži „kazaljku“ – indeks okvira. Procesu je uvek dodeljen najmanje jedan okvir (dodeljuje se pri inicijalizaciji objekta `ProcessFrames`). Potpuna definicija klase `ProcessFrames` data je dole.

Klasa `PMT` implementira PMT jednog procesa i poseduje operaciju

```
uint32* PMT::getPageDesc (size_t page) const;
```

koja za dati broj stranice datog procesa vraća pokazivač na 32-bitni deskriptor stranice u kom je najniži bit referenciranja.

Implementirati sve operacije klase `ProcessFrames` koje poziva ostatak kernela u odgovarajućim koracima postupka zamene stranica, tako da rade sledeće:

- konstruktor prima pokazivač na PMT datog procesa i jedan okvir koji mu se inicijalno dodeljuje;
- `getPage` vraća broj stranice koja je smeštena u dati okvir;
- `setPage` postavlja broj stranice koja je smeštena u dati okvir;
- `getVictim`vraća broj okvira u kome je stranica-žrtva koja je na redu za izbacivanje (kazaljka ostaje na toj stranici);
- `replaceVictim` obaveštava ovaj objekat da je stranica-žrtva zamenjena i da treba pomeriti kazaljku;
- `addFrame` poziva kernel kada datom procesu dodeljuje nov okvir za njegove stranice.

```
class ProcessFrames {
public:
  ProcessFrames (PMT* pmt, size_t frame);

  static size_t getPage (size_t frame) const;
  static void   setPage (size_t frame, size_t page);

  size_t getVictim () const;
  void   replaceVictim ();
  void   addFrame (size_t frame);

private:
  struct FrameDesc {
    size_t page;
    size_t next, prev;
  };
  static FrameDesc frames[NUM_FRAMES];
  PMT* pmt;
  size_t cursor;
};
```

Rešenje:

3. (10 poena) Upravljanje memorijom

Radi sprečavanja pojave zvane *trashing*, neki sistem prati učestanost straničnih grešaka procesa na sledeći način. Za svaki proces broje se stranične greške u polju `pageFaultCounter` strukture PCB. Sistem periodično resetuje ove brojače svih procesa. Za navedene potrebe postoje sledeće operacije čija je implementacija data dole:

- `void incPageFaultCounter(PCB* pcb)`: poziva se prilikom svake stranične greške datog procesa da bi ažurirala brojač;
- `void pfcPeriodicUpdate(PCB* pcb)`: poziva se periodično za svaki proces;
- `unsigned getNumberOfPageFaults(PCB* pcb)`: poziva se prilikom provere pojave zvane *trashing* za dati proces da bi vratila prebrojane stranične greške datog procesa.

```
void incPageFaultCounter (PCB* pcb) {
  pcb->pageFaultCounter++;
}

void pfcPeriodicUpdate (PCB* pcb) {
  pcb->pageFaultCounter = 0;
}

unsigned getNumberOfPageFaults (PCB* pcb) {
  return pcb->pageFaultCounter;
}
```

Ovakvo rešenje ima sledeći problem. Pošto su trenuci kada se ispituje postojanje pojave zvane *thrashing* (i poziva `getNumberOfPageFaults`) nezavisni od trenutaka periodičnih ažuriranja brojača (kada se poziva `pfcPeriodicUpdate`), a ovi prvi se dešavaju znatno ređe od ovih drugih, može se dogoditi da se čitanje vrednosti brojača dogodi brzo nakon njihovog resetovanja, pa se dobija potpuno pogrešna slika: neki proces koji je u prethodnoj periodi generisao mnogo straničnih grešaka neće biti primećen kao problematičan. Predložiti bolje rešenje i implementirati date tri operacije.

Rešenje: