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, septembar 2015.
Datum: 25.8.2015.

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

U nekom sistemu primenjuje se izbegavanje mrtve blokade (engl. *deadlock avoidance*) korišćenjem bankarevog algoritma. Evidenciju zauzeća i alokaciju i dealokaciju resursa implementira klasa `ResourceAllocator`.

Potrebno je implementirati sledeću operaciju ove klase kojom se zahteva alokacija resursa:

```
int ResourceAllocator::request (int p, ResourceVector req);
```

Prvi argument ove operacije predstavlja broj procesa u opsegu između 0 i `np`, gde je `np` broj procesa, a drugi argument predstavlja vektor zahteva za alokacijom resursa. Ova operacija treba da, uz primenu bankarevog algoritma izbegavanja mrtve blokade, izvrši ili ne dozvoli alokaciju resursa datom procesu i vrati sledeću celobrojnu vrednost:

- 0 ukoliko je alokacija izvršena;
- -1 ako je vrednost argumenta `p` izvan dozvoljenog opsega;
- -2 ako je zahtev za alokacijom preko granice najave za taj proces;
- -3 ako nema dovoljno slobodnih resursa da bi se ispunio zahtev;
- -4 ako se zahtev ne može ispuniti jer bi sistem doveo u nebezbedno stanje.

Važe sledeće pretpostavke:

- definisani su i implementirani apstraktni tipovi podataka `AllocationMatrix` i `ResourceVector` koji realizuju matricu zauzeća, odnosno vektor resursa, prema potrebama bankarevog algoritma;
- definisane su i implementirane sve potrebne operacije i relacije preklapanjem operatora +, -, +=, -=, ==, !=, <, >, <= i >= za tip `ResourceVector` na adekvatan način, prema potrebama ovog algoritma;
- definisan je i implementiran operator [] za tip `AllocationMatrix` koji prima kao operand broj procesa, a vraća (referencu na) vektor resursa za taj proces u matrici, tipa `ResourceVector`;
- u klasi `ResourceAllocator` polje `alloc` tipa `AllocationMatrix` realizuje matricu alociranih resursa, polje `max` tipa `AllocationMatrix` realizuje matricu najavljenog maksimalnog korišćenja resursa, a polje `free` tipa `ResourceVector` realizuje vektor trenutno slobodnih resursa;
- operacija `isSafeState()` klase `ResourceAllocator` proverava da li je trenutno stanje predstavljeno objektom ove klase bezbedno (1) ili ne (0).

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem primenjuje aproksimaciju LRU algoritma zamene stranica u okviru istog procesa pomoću dodatnih bita referenciranja.

PMT procesa je organizovana u jednom nivou. Polje `pmt` u strukturi PCB ukazuje na PMT. PMT ima `PMTSIZE` ulaza tipa `unsigned int`. Jedan ulaz u PMT jednak je 0 ako se stranica ne može preslikati u okvir. U suprotnom, ulaz u PMT sadrži broj okvira (različit od 0), bite prava pristupa, kao i bit referenciranja.

Dodatne bite referenciranja za svaki proces sistem čuva u posebnoj strukturi koja je organizovana kao *hash* tabela. Ova struktura dostupna je kao polje `pageRefHash` u PCB svakog procesa. Ova tabela preslikava ključ (broj stranice) u vrednost (registar dodatnih bita referenciranja).

Data je implementacija operacije koja ažurira registre dodatnih bita referenciranja za proces sa datim PCB-om na osnovu bita referenciranja u PMT:

```
void updateReferenceRegs (PCB* pcb) {
  if (pcb==0) return; // Exception!
  for (unsigned page=0; page<PMTSIZE; page++) {
    unsigned frame = pcb->pmt[page];
    if (!frame) continue;
    unsigned ref = pcb->pageRefHash.getValue(page);
    ref >>= 1;
    if (frame&1) ref |= ~MAXINT;
    pcb->pageRefHash.setValue(page,ref);
    (pcb->pmt[page]) &= ~1U;  // Reset the reference bit
  }
}
```

Realizovati operaciju:

```
unsigned getLRUPage (PCB* pcb);
```

koja treba da vrati broj stranice koja je izabrana „žrtva“ za izbacivanje. Pretpostaviti da je prilikom poziva ove operacije bar jedna stranica datog procesa sigurno u memoriji.

Rešenje:

3. (10 poena) Upravljanje memorijom

Radi sprečavanja pojave zvane *trashing*, neki sistem prati učestanost strničnih grešaka za svaki proces na sledeći način. Za svaki proces broje se stranične greške u svakoj periodi (intervalu vremena) i pamte se ti brojevi za ukupno `PFLTCOUNTERS` poslednjih perioda.

U PCB svakog procesa postoji niz `pageFaultCounters` sa `PFLTCOUNTERS` elemenata tipa `unsigned` koji predstavlja kružni bafer brojača straničnih grešaka po periodama. Na brojač koji odgovara tekućoj periodi ukazuje polje `pageFaultCursor` tipa `int` u opsegu od `0` do `PFLTCOUNTERS-1`. Kada istekne data perioda, sistem pomera kurzor na sledeću poziciju u kružnom baferu, tako da elementi ovog bafera uvek čuvaju brojeve straničnih prekida poslednjih `PFLTCOUNTERS` perioda.

Implementirati sledeće operacije:

- `void incPageFaultCounter(PCB* pcb)`: za proces sa datim PCB-om inkrementira brojač straničnih grešaka za tekuću periodu; ova operacija poziva se prilikom svake stranične greške datog procesa;
- `void shiftPageFaultCounters(PCB* pcb)`: za proces sa datim PCB-om pomera kurzor u kružnom baferu; ova operacija poziva se periodično, na isteku svake periode, za svaki proces;
- `unsigned getNumberOfPageFaults(PCB* pcb)`: za proces sa datim PCB-om vraća ukupan broj straničnih grešaka u poslednjih `PFLTCOUNTERS` perioda; ova operacija poziva se prilikom provere pojave *trashing* za dati proces.

Rešenje: