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, decembar 2019.
Datum: 1. 12. 2019.

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**.

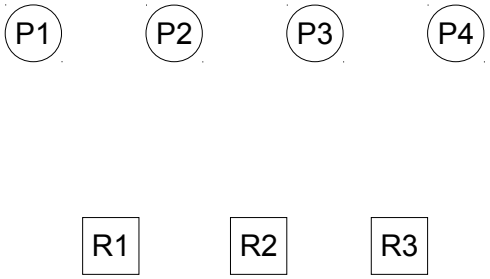
1. (10 poena) Mrtva blokada

Neki sistem izbegava mrtvu blokadu algoritmom zasnovanim na grafu alokacije. U sistemu je aktivno četiri procesa ($P1..P4$). Procesi su inicijalno najavili korišćenje resursa $R1..R3$ prema sledećoj tabeli.

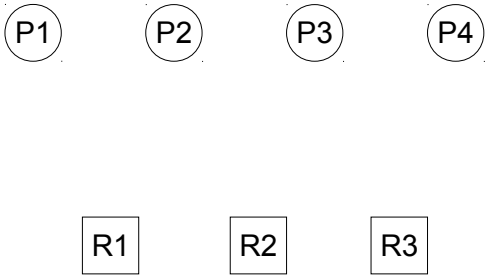
Proces	$P1$	$P2$	$P3$	$P4$
Najavio korišćenje resursa	$R1$, $R2$	$R1$, $R3$	$R1$, $R3$	$R2$, $R3$

Procesi su već izdali sledeće zahteve za alokaciju resursa: $P1$-$R1$, $P4$-$R2$.

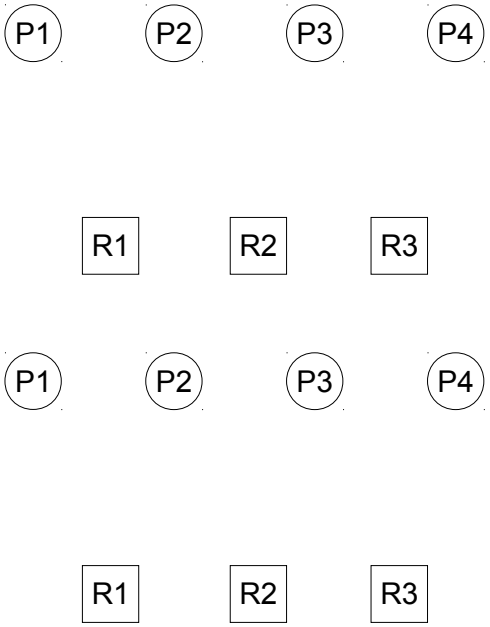
a)(2) Nacrtati graf zauzeća resursa u ovom stanju, nakon izvršenih navedenih zahteva.



b)(3) Nakon zahteva pod a) proces $P2$ traži resurs $R3$. Nacrtati graf alokacije nakon ovog zahteva.



c)(2) Nakon zahteva b) proces $P3$ traži resurs $R1$. Nacrtati graf alokacije nakon ovog zahteva.



d)(3) Nakon zahteva pod c) proces $P4$ oslobađa $R2$. Nacrtati graf alokacije nakon ovog zahteva.

2. (10 poena) Upravljanje memorijom

U nekom računaru virtuelne i fizičke adrese su 32-bitne, a PMT je organizovana u dva nivoa, pri čemu tabele oba nivoa imaju isti broj ulaza, po 1024. U PMT prvog nivoa (tip `PMT0`) svaki ulaz je 32-bitna reč koja sadrži samo pokazivač na početak PMT drugog nivoa; vrednost 0 označava da taj deo virtuelne memorije nije u upotrebi. U PMT drugog nivoa (tip `PMT1`) svaki ulaz sadrži dve 32-bitne reči. Niža reč sadrži broj okvira u koji se stranica preslikava; vrednost 0 u ovoj reči označava da stranica ne može da se preslika (nije u upotrebi ili nije u memoriji). U višoj reči najviši bit je bit referenciranja (*reference bit*), bit do njega bit zaprljanosti (*dirty bit*), dok ostale bite koristi ili hardver ili operativni sistem. Pri tome najnižih 8 bita u ovoj reči operativni sistem koristi kao dodatne bite referenciranja, jer primenjuje algoritam zamene stranica koji aproksimira LRU pomoću ovih dodatnih bita referenciranja.

Implementirati funkciju `scanPages` koja ima dvostruku ulogu. Ona treba da ažurira dodatne bite referenciranja za sve stranice procesa čija je PMT prvog nivoa preneta kao prvi argument (aktivnost koju sistem svakako radi periodično, kao podršku algoritmu zamene stranica), ali da pri istom obilasku stranica pronađe stranicu-žrtvu za izbacivanje i broj te stranice upiše u lokaciju na koju ukazuje argument `victim`; ako je ovaj pokazivač *null*, ne treba da pronalazi žrtvu, samo da ažurira dodatne bite referenciranja.

```
typedef unsigned char uint8;  // 8 bits
typedef unsigned int uint32;  // 32 bits
const uint32 PMT0size = 1024, PMT1size = 1024;
typedef uint32 PMT0[PMT0size];
typedef uint32 PMT1[PMT1size][2];

void scanPages (PMT0 pmt0, uint32* victim);
```

Rešenje:

3. (10 poena) Upravljanje memorijom

U nekom korisničkom programu čvorovi nekog stabla su strukture tipa `Node` i složeni su u niz. Stablo uvek ima barem koren koji se uvek nalazi u elementu niza sa indeksom 0. Svaki čvor, pored ostalih podataka, sadrži indeks (u istom nizu) čvora koji je njegovo prvo dete (član `child`) i indeks čvora koji je njegov prvi susedni blizanac (dete istog roditelja, član `sibling`); vrednost 0 označava nepostojanje deteta, odnosno blizanca. Data je procecedura `traverse` koja stablo dato prvim argumentom obilazi po dubini, rekurzivno, počev od čvora datog drugim argumentom.

Niz je, kako je to definisano standardom jezika C i C++, složen kontinualno u virtuelnoj memoriji, a čvorovi stabla su (propisno uvezani poljima `child` i `sibling`) proizvoljno raspoređeni po nizu. Veličina strukture `Node` je tako podešena da jedna stranica virtuelne memorije sadrži ceo broj elemenata opisanog niza (tj. jedan element niza se nikada neće prelomiti na dve susedne stranice). Deo procedure `traverse` označen sa * pristupa (čitanje ili upis) samo poljima strukture `Node` samo tekućeg elementa niza `tree[node]`. Obratiti pažnju i na to da je kod petlje napisan namerno na način kako je napisan (mesto označeno sa **), umesto ovako, kako bi se očekivalo:

```
for (unsigned cur=tree[node].child; cur!=0; cur=tree[cur].sibling)
  traverse(tree,cur);
```

Ovo je urađeno zbog toga da bi instrukcije ove procedure pristupale samo tekućem čvoru, i kada (u rekurzivnom pozivu) pređu na drugi čvor, više ne pristupaju ponovo poljima istog starog čvora, kako bi bilo u gore navedenom kodu (zbog izvršavanja `cur=tree[cur].sibling` na kraju iteracije, a nakon rekurzivnog poziva koji obilazi druge čvorove).

```
struct Node {
  ...
  unsigned sibling, child;
};

void traverse (Node tree[], unsigned node) {
  ... // (*) Read/write access to the fields of tree[node] only
  for (unsigned cur=tree[node].child, next; cur!=0; cur=next) {
    next = tree[cur].sibling;  // (**)
    traverse(tree,cur);
  }
}
```

Dato stablo obrađuje se onda počev od korena ovako:

```
extern Node tree[];
...traverse(tree,0)...
```

a)(5) Kako treba urediti (poređati) čvorove stabla u nizu da bi data procedura `traverse`, bez ikakve izmene u toj proceduri, uvek generisala teorijski najmanji broj straničnih grešaka (*page fault*) prilikom pristupa datom nizu sa stablom, i to za svaki algoritam zamene stranica i za svaki broj okvira koji je na raspolaganju za smeštanje niza, pa čak i za samo jedan takav okvir? Koliki je tada broj tih straničnih grešaka?

b)(5) Implementirati proceduru `repack` koja postojeće stablo u nizu `oldTree` „prepakuje“, odnosno njegove čvorove preraspoređuje u novi niz `newTree` (niz je već alociran u potrebnoj veličini) na način pod a). Definisana je globalna promenljiva `size`, koja se pre prepakivanja inicijalizuje na 0, i koja prati tekuću veličinu novog niza, odnosno broj do tada prepakovanih čvorova.

```
extern unsigned size;
void repack (Node oldTree[], Node newTree[], unsigned oldNode);
```

Dato stablo `oldTree` prepakuje se onda u stablo `newTree` počev od korena ovako:

```
extern Node oldree[], newTree[];
size = 0;
repack (oldTree, newTree, 0);
```