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, januar 2020.
Datum: 8. 1. 2020.

Drugi kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 1,5 sat. Dozvoljeno je korišćenje literature.

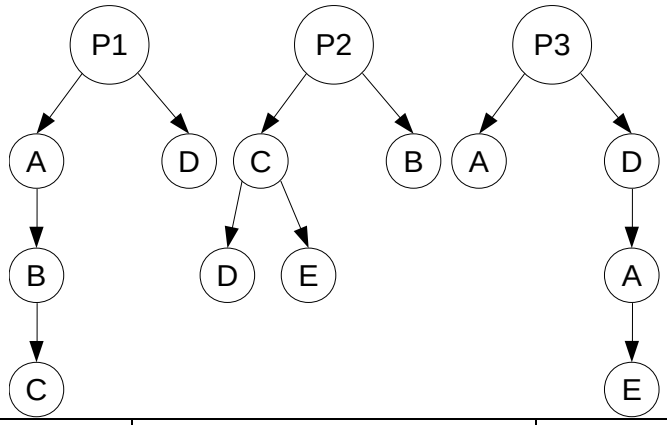
Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

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 konkurentan program sastoji se od uporednih procesa koji poseduju ugnežđene kritične sekcije zaštićene blokirajućim međusobnim isključenjem. Jedan primer takvog programa dat je (konceptualno) dole. Za ovakav program moguće je izvršiti statičku analizu korišćenja kritičnih sekcija. Rezultat takve analize je struktura podataka oblika šume stabala, kakva je data dole za ovaj primer. Potrebno je ispitati da li ovakav program može ući u mrtvu blokadu.




```
process P1 {
  mutex A {
    mutex B {
      mutex C {}
    }
  }
  mutex D {}
}
```

```
process P2 {
  mutex C {
    mutex D {}
    mutex E {}
  }
  mutex B {}
}
```

```
process P3 {
  mutex A {}
  mutex D {
    mutex A {
      mutex E {}
    }
  }
}
```

a)(7) Koncizno i precizno opisati algoritam (i prateću strukturu podataka koju taj algoritam koristi) koji ispituje postojanje mogućnosti mrtve blokade.

b)(3) Prikazati kako taj algoritam deluje na datom primeru, tj. situaciju mrtve blokade koju taj algoritam otkriva za dati primer, ako ona postoji, odnosno dokaz da ona ne postoji.

Rešenje:

2. (10 poena) Upravljanje memorijom

Dat je fajl čiji je sadržaj binaran i sastoji se od celih binarnih brojeva tipa `unsigned`, uzastopno zapisanih u istom formatu u kom se predstavljaju i u memoriji. Na samom početku sadržaja fajla je rezervisano mesto za jedan takav broj u koji treba upisati izračunatu vrednost. Iza njega se nalazi neoznačen ceo broj koji sadrži veličinu niza brojeva; ova veličina sigurno nije veća od `MAX_ARRAY_SIZE`. Iza toga se nalazi niz brojeva tipa `unsigned` ove veličine.
Korišćenjem tehnike memorijski preslikanih fajlova, napisati program koji se poziva sa jednim argumentom, nazivom fajla čiji je sadržaj u opisanom formatu, i koji treba da pronađe maksimum elemenata niza zapisanog u tom fajlu i upiše ga na predviđeno mesto na početku tog fajla. Na raspolaganju su sledeći POSIX sistemski pozivi sa izvodom iz dokumentacije:

```
#include <fcntl.h>
int open (const char *pathname, int flags);
int close(int fd);
```

The open() *system call opens the file specified by* pathname. *If the specified file does not exist, it may optionally (if* O_CREAT *is specified in* flags*) be created by* open().
The return value of open() *is a file descriptor, a small, nonnegative integer that is used in subsequent system calls to refer to the open file. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.*
The argument flags *must include one of the following access modes:* O_RDONLY, O_WRONLY, *or* O_RDWR. *These request opening the file read-only, write-only, or read/write, respectively.*
open() *returns the new file descriptor, or -1 if an error occurred.*

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t
offset);
int munmap(void *addr, size_t length);
```

mmap() *creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in* addr. *The* length *argument specifies the length of the mapping (which must be greater than 0).*
If addr *is* NULL, *then the kernel chooses the (page-aligned) address at which to create the mapping; this is the most portable method of creating a new mapping. If* addr *is not* NULL, *then the kernel takes it as a hint about where to place the mapping*[...] *The address of the new mapping is returned as the result of the call.*
The contents of a file mapping are initialized using length *bytes starting at offset* offset *in the file (or other object) referred to by the file descriptor* fd. offset *must be a multiple of the page size as returned by* sysconf(_SC_PAGE_SIZE).
After the mmap() *call has returned, the file descriptor,* fd, *can be closed immediately without invalidating the mapping.*
The prot *argument describes the desired memory protection of the mapping (and must not conflict with the open mode of the file). It is either* PROT_NONE *or the bitwise OR of one or more of the following flags:*

PROT_EXEC *Pages may be executed.*
PROT_READ *Pages may be read.*
PROT_WRITE *Pages may be written.*
PROT_NONE *Pages may not be accessed.*

On success, mmap() *returns a pointer to the mapped area. On error, the value* MAP_FAILED *(that is,* (void *)-1*) is returned.*
The munmap() *system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references. The region is also automatically unmapped when the process is terminated. On the other hand, closing the file descriptor does not unmap the region.*

Izostvaljajući detalje, argument *flags* u pozivu *mmap()* treba postaviti na *MAP_SHARED*.

Rešenje:

3. (10 poena) Upravljanje memorijom

Kernel nekog operativnog sistema koristi *buddy* alokator memorije za potrebe alokacije svojih struktura podataka. Najmanja jedinica alokacije je jedna stranica veličine 4KB, a adresibilna jedinica je bajt. Na početku je ceo segment memorije kojom rukuje ovaj alokator slobodan, a stanje strukture podataka koju koristi ovaj alokator prikazana je tabelarno. U prvoj koloni je stepen *n*, a u drugoj koloni su početne adrese slobodnih blokova veličine 2^n stranica.

n	*Početne adrese (hex) slobodnih blokova veličine* 2^n
4	A 00 00
3	-
2	-
1	-
0	-

a)(5) Prikazati na isti način stanje ove strukture podataka nakon sledeće sekvence alokacija blokova date veličine: 8KB, 4KB, 16KB, 8KB.

b)(5) Nakon sekvence pod a), oslobođeni su blokovi na adresama A2000h i A8000h. Prikazati na isti način stanje ove strukture podataka nakon završetka svih ovih operacija dealokacije.

Rešenje:
a)

n	*Početne adrese (hex) slobodnih blokova veličine* 2^n
4	
3	
2	
1	
0	

b)

n	*Početne adrese (hex) slobodnih blokova veličine* 2^n
4	
3	
2	
1	
0	