Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1 (SI2OS1, IR2OS1)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, maj 2015.
Datum: 9.5.2015.

Drugi kolokvijum iz Operativnih sistema 1

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 = _____% = _____/15

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)

U nastavku je dat parcijalni opis neznatno izmenjenog (pojednostavljenog) POSIX API za semafore. Napisati delove koda dva procesa koji međusobno isključuju izvršavanja svojih kritičnih sekcija korišćenjem ovih semafora.

```
#include <fcntl.h>
/* For O_* constants */
#include <sys/stat.h>
sem_t *sem_open(const char *name, int oflag, unsigned int value);
#include <semaphore.h>
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_close(sem_t *sem);
```

sem_open() creates a new POSIX semaphore or opens an existing semaphore. The semaphore is identified by name of the form /somename; that is, a null-terminated string of up to NAME_MAX-4 (i.e., 251) characters consisting of an initial slash, followed by one or more characters, none of which are slashes. Two processes can operate on the same named semaphore by passing the same name to sem_open().

The oflag argument specifies flags that control the operation of the call. (Definitions of the flags values can be obtained by including <fcntl.h>.) If O_CREAT is specified in oflag, then the semaphore is created if it does not already exist. If both O_CREAT and O_EXCL are specified in oflag, then an error is returned if a semaphore with the given name already exists.

If O_CREAT is specified in oflag, then the value argument must be specified. The value argument specifies the initial value for the new semaphore. If O_CREAT is specified, and a semaphore with the given name already exists, then value is ignored.

After the semaphore has been opened, it can be operated on using sem_post() and sem_wait(). When a process has finished using the semaphore, it can use sem_close() to close the semaphore.

On success, sem_open() returns the address of the new semaphore; this address is used when calling other semaphore-related functions.

sem_post() increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait() call will be woken up and proceed to lock the semaphore.

sem_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement (i.e., the semaphore value rises above zero).

Rešenje:

2. (10 poena)

U nekom sistemu koristi se kontinualna alokacija memorije. U PCB postoje sledeća polja:

- `char* mem_base_addr`: početna adresa u memoriji na koju je smešten proces;
- `size_t mem_size`: veličina dela memorije koju zauzima proces;
- `PCB* mem_next`: pokazivač na sledeći u listi procesa; PCB procesa su ulančani u ovu listu po redosledu njihovog smeštanja u memoriji (po rastućem redosledu `mem_base_addr`); glava ove liste je globalna promenljiva `proc_mem_head`.

Na početak dela operativne memorije koja se koristi za smeštanje procesa ukazuje pokazivač `user_proc_mem_start`, a na njegov kraj (na poslednji znak) pokazivač `user_proc_mem_end`; oba su tipa `char*`.

Fragmenti slobodne memorije ulančani su u jednostruko ulančanu listu, pri čemu se svaki element te liste, tipa `FreeSegment`, smešta na sam početak slobodnog fragmenta. Glava ove liste je u globalnoj promenljivoj `mem_free_head`. Struktura `FreeSegment` izgleda ovako:

```
struct FreeSegment {
  FreeSegment* next;  // Next free segment in the list
  size_t size; // Total size of the free segment
};
```

Napisati proceduru `mem_compact()` koja vrši kompakciju slobodnog prostora relokacijom procesa.

Rešenje:

3. (10 poena)

U nekom sistemu koristi se straničenje sa PMT u jednom nivou i tehnika *copy-on-write*. Tabela deskriptora alociranih memorijskih segmenata jednog procesa ima sledeći sadržaj:

Segment#	*Starting page*	*Size in pages*	*RWX*
1	00h	4h	001
2	12h	3h	100
3	1Ah	2h	110

PMT ovog procesa u posmatranom stanju ima sledeći sadržaj (svi ulazi koji nisu navedeni imaju sadržaj 000 u polju *RWX*, što hardveru znači da preslikavanje nije dozvoljeno iz bilo kog razloga i da treba generisati *page fault*):

Page#	*Frame#*	*RWX*
00h	10h	
01h	11h	
02h	12h	
03h	13h	
12h	14h	
13h	15h	
14h	16h	
1Ah	17h	
1Bh	18h	

a)(2) U prethodnu tabelu upisati vrednosti u kolonu *RWX*, ako su sve stranice učitane.

b)(4) U opisanom stanju ovaj proces kreira proces-dete pozivom `fork()`. Prikazati PMT oba procesa (i roditelja i deteta) nakon uspešnog završetka ovog poziva.

c)(4) Nakon toga, proces-dete izvršava instrukciju koja upisuje vrednost u lokaciju na stranici 1Ah. Prikazati PMT procesa-deteta nakon što je operativni sistem obradio izuzetak koji je nastao tokom izvršavanja ove instrukcije. Pretpostaviti da je prvi slobodan okvir 19h.

Rešenje:

b) PMT procesa-roditelja:

Page#	*Frame#*	*RWX*
00h		
01h		
02h		
03h		
12h		
13h		
14h		
1Ah		
1Bh		

PMT procesa-deteta:

Page#	*Frame#*	*RWX*
00h		
01h		
02h		
03h		
12h		
13h		
14h		
1Ah		
1Bh		

c) PMT procesa-deteta:

Page#	*Frame#*	*RWX*
00h		
01h		
02h		
03h		
12h		
13h		
14h		
1Ah		
1Bh		