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, april 2018.
Datum: 29. 4. 2018.

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.

```
#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).

```
#include <fcntl.h>
pid_t getpid(void);
```

getpid() *shall return the process ID of the calling process.*

Korišćenjem ovih sistemskih poziva, implementirati apstrakciju `Mutex` koja predstavlja sinhronizacionu primitivu za međusobno isključenje, koja se može koristiti samo u te svrhe, tj. uz ograničenje da samo proces koji je zaključao kritičnu sekciju može da je otključa: ukoliko to nije zadovoljeno, operacije `exit` treba da vrati grešku (negativnu vrednost). Ova apstrakcija treba da ima sledeći objektno orijentisani interfejs:

```
class Mutex {
public:
  Mutex (const char *name);
  int entry ();
  int exit();
};
```

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 u jedinicama `sizeof(char)`.

Fragmenti slobodne memorije ulančani su u jednostruko ulančanu listu, uređenu po rastućem poretku adresa slobodnih fragmenata, 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 in sizeof(char)
};
```

Napisati funkciju `proc_relocate()` koja vrši relokaciju datog procesa sa ciljem obavljanja lokalizovane, ograničene kompakcije na sledeći način. Ako i samo ako i ispred i iza datog procesa u memoriji postoji slobodan fragment, onda se taj proces pomera na početak slobodnog fragmenta ispred njega, kako bi se slobodan prostor ta dva fragmenta spojio u jedan, iza tog procesa. Ukoliko je ova kompakcija urađena, funkcija vraća 1, inače vraća 0.

```
int proc_relocate (PCB* pcb);
```

Rešenje:

3. (10 poena)

Neki sistem zauzima najnižih *n* okvira raspoložive fizičke memorije za potrebe kernela. Kako ne bi menjao memorijski kontekst prilikom obrade sistemskih poziva, sistem preslikava (mapira) najviših *n* stranica svakog kreiranog procesa u ovaj memorijski prostor kernela, s tim da postavlja indikatore u deskriptoru tih stranica u PMT (*page map table*) tako da one nisu dostupne u neprivilegovanom (korisničkom) režimu rada procesora. Ostatak virtuelnog adresnog prostora dostupan je samom procesu za njegove potrebe i u korisničkom režimu.

Virtuelni adresni prostor je 16EB (eksabajt, $1E=2^{60}$) i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 16KB. Fizički adresni prostor je veličine 1TB (terabajt). PMT je organizovana u dva nivoa, s tim da su i broj ulaza, kao i širina ulaza u PMT prvog i drugog nivoa isti (PMT oba nivoa su iste veličine). PMT oba nivoa smeštaju se u memoriju uvek poravnate na fizički okvir, odnosno uvek počinju na početku okvira. Zbog toga se u ulazu prvog nivoa čuva samo broj okvira u kom počinje PMT drugog nivoa, dok se preostali biti do celog broja bajtova u ulazu ne koriste; vrednost 0 u svim bitima označava da preslikavanje nije dozvoljeno. U jednom ulazu PMT drugog nivoa čuva se broj okvira u koji se stranica preslikava u najnižim bitima, dok 5 najviših bita ima sledeće značenje (tim redom, od najvišeg ka najnižem): bit koji govori da li je data stranica dostupna u neprivilegovanom (korisničkom) režimu rada procesora, bit koji govori da li je preslikavanje stranice moguće ili ne, i još 3 bita koja koduju prava pristupa (*rwx*), dok se ostali biti ne koriste. Jedan ulaz u PMT prvog i drugog nivoa zauzima minimalan, ali ceo broj bajtova.

Implementirati sledeću funkciju:

```
void initPMT (unsigned* pmt, unsigned long pageFrom, unsigned long nPages,
              unsigned long frameFrom, unsigned short rwx);
```

Ovu funkciju poziva kod kernela kada inicijalizuje PMT novokreiranog procesa, na čiju (već alociranu) PMT prvog nivoa ukazuje argument `pmt`, kako bi obavio opisano preslikavanje dela virtuelnog prostora u kernel prostor. Ona treba da preslika `nPages` susednih stranica počev od stranice `pageFrom` u isto toliko susednih okvira počev od okvira `frameFrom`. Argument `rwx` sadrži bite prava pristupa koje treba postaviti u deskriptore svih tih stranica.

Na raspolaganju je interna funkcija kernela `alloc_pmt2` koja alocira jednu PMT drugog nivoa u memoriji, popunjava sve njene ulaze nulama i vraća broj okvira u kom počinje ta alocirana PMT. Pretpostaviti da će ova funkcija uvek uspeti da alocira PMT u memoriji (ignorisati greške). Veličine tipova su sledeće: `int` – 32 bita, `long` – 64 bita, `short` – 16 bita.

Obratiti pažnju na to da se ceo kernel kod izvršava u memorijskom kontekstu nekog procesa, što znači da su sve adrese (vrednosti pokazivača) virtuelne. Za konverziju neke fizičke adrese (vrednosti pokazivača) u kernel prostoru u virtuelnu adresu (u najvišem delu virtuelnog prostora svakog procesa), na raspolaganju je sledeća funkcija:

```
void* kaddrPtoV (void* physicalAddr);
```

Rešenje: