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 2019.
Datum: 14. 1. 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 ima ugrađen mehanizam detekcije mrtve blokade zasnovan na bankarevom algoritmu (*Banker's algorithm*). Trenutno stanje alokacije resursa za četiri aktivna procesa je:

Allocation

	A	*B*	*C*
*P*1	1	0	1
*P*2	2	1	1
*P*3	0	1	1
*P*4	0	1	1

Request

	A	*B*	*C*
*P*1	0	1	3
*P*2	0	0	0
*P*3	2	0	1
*P*4	3	1	0

Available

A	*B*	*C*
0	0	0

Matrica *Request* predstavlja zahteve za resursima koje aktivni procesi trenutno imaju postavljene (sve nule znače da taj proces trenutno ne traži resurse).

a)(7) Da li mrtva blokada *sigurno* postoji i ako je tako, koji od ovih procesa *sigurno* jesu u mrtvoj blokadi? Prikazati i obrazložiti postupak.

b)(3) Ako je odgovor pod a) odrečan, navesti koliko najmanje resursa bi trebalo da traži proces *P2* da bi sistem sigurno ušao u mrtvu blokadu.

Ako je odgovor pod a) potvrdan, navesti koji od procesa koji su sigurno u mrtvoj blokadi treba ugasiti da bi se mrtva blokada rešila. Navesti sva rešenja.

Rešenje:

2. (10 poena) Upravljanje memorijom

Za izbor stranice za zamenu neki sistem koristi algoritam „*nekorišćen nedavno*“ (engl. *Not Recently Used*, NRU). Stranicama su pridruženi biti referenciranja i biti zaprljanosti (modifikacije) koje održava hardver. Sistem periodično briše bite referenciranja, tako da bit referenciranja ukazuje na to da li je stranica korišćena u poslednjem intervalu (nedavno). Za zamenu se bira *prva* (najniža) stranica iz najniže grupe, ako takva postoji, pri čemu se stranice posmatraju klasifikovane po grupama na sledeći način:
0. nije korišćena, nije modifikovana
1. nije korišćena, jeste modifikovana (moguće je zato što je bit referenciranja obrisan)
2. korišćena, nije modifikovana
3. korišćena, modifikovana.

Sistem primenjuje straničenje u dva nivoa, virtuelna adresa je 32 bita, stranica je veličine 4 KB, adresibilna jedinica je bajt, a smeštanje višebajtnih reči u memoriju je po šemi niži bajt-niža adresa (*little endian*). PMT prvog i drugog nivoa imaju isti broj ulaza. Svaki ulaz u PMT prvog nivoa je veličine 32 bita i sadrži adresu PMT drugog nivoa (ili 0 ako je PMT drugog nivoa nealocirana). Svaki ulaz u PMT drugog nivoa, odnosno deskriptor stranice, sadrži dve 32-bitne reči. Nižu reč koristi hardver prilikom preslikavanja, dok je viša reč potpuno slobodna za korišćenje od strane operativnog sistema. U nižoj reči, najviši bit je bit zaprljanosti, a bit do njega bit referenciranja. Kernel preslikava svoj deo fizičke memorije u najviših 16 MB virtuelnog adresnog prostora svakog procesa, a prilikom izvršavanja kernel koda preslikavanje odgovara tekućem procesu (aktivan je memorijski kontekst tog procesa).

```
typedef unsigned int   uint32; // 32 bits
typedef unsigned short uint16; // 16 bits
typedef unsigned char  uint8;  // 8 bits
typedef uint32 PgDesc[2];
PgDesc* getVictimPageDesc (uint32* pmt);
```

Implementirati funkciju `getVictimPageDesc` koja na opisani način treba da odredi stranicu žrtvu i vrati pokazivač na njen deksriptor u PMT drugog nivoa, za proces sa datom PMT prvog nivoa.

Rešenje:

3. (10 poena) Upravljanje memorijom

Dat je izvod iz dokumentacije za POSIX/Linux sistemske pozive `mmap` i `munmap`:

mmap, munmap - map or unmap files or devices into memory

```
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 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; on Linux, the mapping will be created at a nearby page boundary. 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)`.*
>
> *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:...*
>
> *The `flags` argument determines whether updates to the mapping are visible to other processes mapping the same region, and whether updates are carried through to the underlying file. This behavior is determined by including exactly one of the following values in `flags`:...*
>
> *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.*

Precizno objasniti šta radi sledeći program i kratko prokomentarisati svaki od segmenata koda funkcije *main* koji su odvojeni proredima (šta taj deo koda radi):

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define handle_error(msg) \
  do { perror(msg); exit(EXIT_FAILURE); } while (0)

int main(int argc, char *argv[]) {
  char *addr;
  int fd;
  struct stat sb;
  off_t offset, pa_offset;
  size_t length;
  ssize_t s;
```

```
  if (argc < 3 || argc > 4) {
    fprintf(stderr, "%s file offset [length]\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  fd = open(argv[1], O_RDONLY);
  if (fd == -1)
    handle_error("open");

  if (fstat(fd, &sb) == -1)           /* To obtain file size */
    handle_error("fstat");

  offset = atoi(argv[2]);
  pa_offset = offset & ~(sysconf(_SC_PAGE_SIZE) - 1);
    /* offset for mmap() must be page-aligned */

  if (offset >= sb.st_size) {
    fprintf(stderr, "offset is past end of file\n");
    exit(EXIT_FAILURE);
  }

  if (argc == 4) {
    length = atoi(argv[3]);
    if (offset + length > sb.st_size)
      length = sb.st_size - offset;
        /* Can't access bytes past end of file */
  } else {    /* No length arg ==> to end of file */
    length = sb.st_size - offset;
  }

  addr = mmap(NULL, length + offset - pa_offset, PROT_READ,
              MAP_PRIVATE, fd, pa_offset);
  if (addr == MAP_FAILED)
    handle_error("mmap");

  s = write(STDOUT_FILENO, addr + offset - pa_offset, length);
  if (s != length) {
    if (s == -1)
      handle_error("write");

    fprintf(stderr, "partial write");
    exit(EXIT_FAILURE);
  }

  munmap(addr, length + offset - pa_offset);
  close(fd);

  exit(EXIT_SUCCESS);
}
```

Odgovor: