Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Integralni, oktobar 3/2025.
Datum: 30. 10. 2025.

Kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje dva sata. Dozvoljeno je korišćenje literature.

Zadatak 1	_____/10	*Zadatak 3*	_____/10
Zadatak 2	_____/10	*Zadatak 4*	_____/10

Ukupno: _____/40

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 nekom računaru sa straničnom organizacijom virtuelna adresa je širine 48 bita, adresibilna jedinica je bajt, a stranica je veličine 4 KB. Kernel operativnog sistema preslikava najniži 1 GB virtuelnog adresnog prostora svakog procesa u isti deo fizičkog adresnog prostora za svoje potrebe (ROM, ulaz-izlaz, RAM koji koristi sam kernel itd.), dok je ostatak virtuelnog adresnog prostora svojstven svakom procesu. Posmatra se *n* pokrenutih procesa od kojih svaki, osim tih prvih 1 GB virtuelnog adresnog prostora koji kernel preslikava u zajednički deo fizičke memorije, koristi samo po još jednu stranicu sa sopstvenim sadržajem koja je smeštena odmah iznad tog deljenog dela (prvih narednih 4 KB). Kernel deli sve stranice u kojima su PMT-i koje može da deli između različitih procesa za potrebe preslikavanja svog prostora (najniži 1 GB).

Izračunati koliko ukupno stranica memorije zauzimaju svi PMT-ovi svih tih *n* procesa za sledeće slučajeve organizacije PMT-a. Odgovor i postupak precizno obrazložiti.

a)(5) PMT je organizovan u najmanjem potrebnom broju nivoa, s tim da svaki PMT svakog nivoa zauzima tačno po jednu stranicu i ima ulaze veličine 64 bita.[1]

Odgovor: _____ stranica

b)(5) Procesor sada drugačije organizuje preslikavanje adresa i PMT. Za najniži 1 GB virtuelnog adresnog prostora koristi se poseban PMT na koju ukazuje poseban PMTP (označimo ga kao PMTPX), što znači da ako je viših 18 bita virtuelne adrese jednako 0, preslikavanje koristi ovaj PMTPX. Za preostali deo adresnog prostora koristi se drugi PMTP (PMTPY). U oba slučaja koristi se PMT tako da ima najmanji potreban broj nivoa, pod istim uslovom da svaki ulaz zauzima 64 bita i da svaki PMT zauzima tačno jednu stranicu.

Odgovor: _____ stranica

[1]Ovo je opis Intel 64-bitne arhitekture koja koristi samo nižih 48 bita za virtuelnu adresu.

2. (10 poena)

U Školskom jezgru implementirana je statička operacija klase `Thread`

```
void Thread::yield (Thread* oldRunning, Thread* newRunning);
```

koja obavlja promenu konteksta oduzimajući procesor (tekućoj) niti na koju pokazuje parametar `oldRunning` i predajući procesor niti na koju pokazuje parametar `newRunning`. Korišćenjem ove operacije implementirati klasu `Event` koja realizuje binarni semafor (događaj) uz ograničenje da samo nit koja je „vlasnik“ događaja, a to je nit u čijem kontekstu je događaj kreiran, može da poziva operaciju `wait`, dok operaciju `signal` može da poziva bilo koja nit. U slučaju da je ovo ograničenje prekršeno, operacija treba da vrati -1, u suprotnom treba da vrati 0.

Rešenje:

3. (10 poena)

Računar poseduje DMA kontroler koji ima `NDMAs` nezavisnih kanala koji mogu da obavljaju isto toliko uporednih izlaznih operacija. Date su deklaracije:

```
const int NDMAs = ...;
typedef volatile unsigned int REG;
extern REG dmaCtrl[NDMAs];     // DMA control registers
extern REG dmaStatus[NDMAs];   // DMA status registers
extern REG dmaAddress[NDMAs];  // DMA block address registers
extern REG dmaCount[NDMAs];    // DMA block size registers
```

DMA kontroler poseduje po jedan upravljački, statusni, registar za zadavanje adrese bloka i registar za zadavanje veličine bloka za svaki svoj kanal; registri svake od ovih grupa su redom vezani na susedne adrese, tako da se mogu posmatrati kao niz registara prema gornjim deklaracijama. U upravljačkom registru najniži bit je bit *Start* kojim se pokreće prenos jednog bloka preko odgovarajućeg DMA kanala, a u statusnom registru najniži bit je bit završetka prenosa (*TransferComplete*), a bit do njega bit greške (*Error*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). Kada DMA kanal završi zadati prenos, on se automatski zaustavlja (nije ga potrebno zaustavljati upisom u upravljački registar). Završetak prenosa sa bilo kog DMA kanala generiše isti zahtev za prekid procesoru (signali završetka operacije sa svih DMA kanala vezani su na ulazni zahtev za prekid preko OR logičkog kola).

Zahtevi za izlaznim operacijama na nekom uređaju na koji se prenos blokova vrši preko bilo kog od ovih DMA kanala vezani su u jednostruko ulančanu listu. Zahtev ima sledeću strukturu:

```
struct IORequest {
  REG* buffer; // Data buffer (block)
  unsigned int size; // Buffer (blok) size
  int status; // Status of operation
  IORequest* next; // Next in the list
};
```

Na prvi zahtev u listi pokazuje globali pokazivač `ioHead`. Kada u praznu listu kernel stavi prvi zahtev, pozvaće operaciju `transfer()` koja treba da pokrene prenos za taj prvi zahtev na bilo kom trenutno slobodnom DMA kanalu. Kada se završi prenos zadat jednim zahtevom na jednom DMA kanalu, potrebno je u polje `status` date strukture preneti status završene operacije (0 – ispravno završeno do kraja, -1 – greška) i pokrenuti prenos za sledeći zapis u listi na tom DMA kanalu, i potom izbaciti zahtev iz liste. Obratiti pažnju na to da više DMA kanala mogu završiti prenos i generisati prekid istovremeno. Ako zahteva u listi više nema, ne treba uraditi više ništa (kada bude stavljao novi zahtev u listu, kernel će proveriti i videti da je ona bila prazna, pa pozvati ponovo operaciju `transfer()` itd.).

Potrebno je napisati kod operacije `transfer()` i prekidne rutine `dmaInterrupt()` za prekid od DMA kontrolera. Smatrati da se navedene funkcije izvršavaju izolovano, ne treba brinuti o međusobnom isključenju jer je ono rešeno na višem nivou.

```
void transfer ();
interrupt void dmaInterrupt ();
```

Rešenje:

4. (10 poena)

Fajl sistem FAT16 koji je koristio MS DOS svaki ulaz u direktorijumu predstavlja strukturom `FATDirEntry` koja zauzima tačno 32 bajta i u kojoj su upisani atributi fajla ili poddirektorijuma. (Ova struktura zapravo predstavlja FCB koji je ugrađen u sam zapis sadržaja direktorijuma.) Spisak ulaza u direktorijumu zapisuje se kao sadržaj bilo kog fajla. Ovaj spisak je prost niz ovakvih struktura koje se redom dodaju u niz kako se elementi dodaju u direktorijum, pri čemu se za obrisane tj. slobodne ulaze prvi bajt ove strukture postavlja na posebnu vrednost `0xE5`. Spisak se završava ili kada se u FAT lancu naiđe na kraj (vrednost 0 za klaster) ili prvi bajt ulaza ima vrednost 0. Naziv ulaza ima uvek 8 znakova, a ekstenzija 3 znaka (dopunjeni su blanko znakovima ako su kraći) i ovi stringovi se ne završavaju znakom `'\0'`. Jedan bit u polju `attr` ukazuje na to da li ulaz predstavlja fajl ili poddirektorijum. Polje `firstCluster` u ovoj strukturi ukazuje na prvi klaster sa zapisom sadržaja fajla odnosno poddirektorijuma. Klaster na disku je uvek veličine 512 bajtova (jedan blok).

```
typedef struct {
    char     name[8];         // File name (padded with spaces)
    char     ext[3];          // File extension (padded with spaces)
    uint8_t  attr;            // File attributes
    uint8_t  reserved;        // Reserved (used for Windows NT or VFAT)
    uint8_t  creationTimeTenth; // Creation time in tenths of a second
    uint16_t creationTime;    // Creation time
    uint16_t creationDate;    // Creation date
    uint16_t lastAccessDate;  // Last access date
    uint16_t highCluster;     // Unused; always 0 for FAT16
    uint16_t writeTime;       // Last write time
    uint16_t writeDate;       // Last write date
    uint16_t firstCluster;    // First cluster
    uint32_t fileSize;        // File size in bytes
} FATDirEntry;

void* FSCache::getCluster(uint16_t cluster);
uint16_t FAT::getNextCluster(uint16_t cluster);

FATDirEntry* getEntry(FATDirEntry* dir,const char name[],const char ext[]);
```

Statička funkcija `FAT::getNextCluster` vraća broj sledećeg klastera u listi unutar FAT za zadati klaster, a funkcija `FSCache::getCluster` obezbeđuje da je zadati klaster učitan u operativnu memoriju i vraća pokazivač na njega. Realizovati funkciju `getEntry` koja za zadati FCB koji sigurno predstavlja direktorijum (ne treba proveravati, već je provereno) pronalazi ulaz sa datim imenom i ekstenzijom, ukoliko postoji, i vraća pokazivač na taj ulaz (FCB); ako ne postoji, vraća *null*.

Rešenje: