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, jun 2020.
Datum: 22. 6. 2020.

Kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 2 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)

Neki procesor obrađuje prekide (hardverske i softverske) tako što tokom izvršavanja prekidne rutine koristi poseban stek koji se koristi u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina). Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar SSP procesora koji je dostupan samo u privilegovanom režimu.

Prilikom obrade prekida, procesor ništa ne stavlja na stek, pa tako ni ne menja sadržaj pokazivača vrha korisničkog steka (SP) koji je koristio prekinuti proces (SP zadržava staru vrednost), dok tekuću (staru) vrednost statusnog registra (PSW) i programskog brojača (PC) sačuva u posebne, za to namenjene registre, SPSW i SPC, respektivno. Prilikom povratka iz prekidne rutine instrukcijom `iret`, procesor restaurira registre PSW i PC prepisujući vrednosti iz registara SPSW i SPC i vraća se u korisnički režim, a time i na korisnički stek.

Ovaj isti sistemski stek se koristi tokom izvršavanja bilo kog koda kernela; kernel ima samo jedan takav stek (nema više niti). Prilikom promene konteksta, kontekst procesora treba sačuvati u odgovarajućim poljima strukture PCB, u kojoj postoji polje za čuvanje svakog od programski dostupnih registara; pomeraj ovog polja u odnosu na početak strukture PCB označen je simboličkim konstantama `offsPC`, `offsSP`, `offsPSW`, `offsR0`, `offsR1` itd. Procesor je RISC, sa *load/store* arhitekturom i ima 32 registra opšte namene (R0..R31).

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `s_call`, koji nema argumente, realizuje sistemski poziv zadat od strane korisničkog procesa vrednostima u registrima i po potrebi raspoređivanje, tako što smešta PCB na koji ukazuje pokazivač `running` u listu spremnih procesa ili na drugo mesto, a iz liste spremnih uzima jedan odabrani proces i postavlja pokazivač `running` na njega.

Na asembleru datog procesora napisati kod prekidne rutine `sys_call` koja vrši sistemski poziv korišćenjem potprograma `s_call`. Ova prekidna rutina poziva se softverskim prekidom iz korisničkog procesa, pri čemu se sistemski poziv realizuje softverskim prekidom.

Rešenje:

2. (10 poena)

Korišćenjem školskog jezgra i date klase `RowAdder`, implementirati funkciju `mat_add` koja sabira elemente svake vrste matrice `mat` i rezultat tog sabiranja smešta u odgovoarajući element niza `res`, i to radi uporedo i tako da kontrolu vraća pozivaocu tek kada su uporedna sabiranja svih vrsta matrice završena.

```
class RowAdder : public Thread {
public:
  RowAdder (int arr[], int cnt, int* res, Semaphore* sem)
    : a(arr), n(cnt), r(res), s(sem) {}
protected:
  virtual void run ();
private:
  int *a, n, *r;
  Semaphore* s;
};

void RowAdder::run () {
  *r = 0;
  for (int i=0; i<n; i++) *r += a[i];
  if (s) s->signal();
}

const int M = ..., N = ...;
int mat[M][N];
int res[M];

void mat_add ();
```

Rešenje:

3. (10 poena)

Virtuelni adresni prostor nekog sistema je 8TB (terabajta) i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 4KB. PMT (*page map table*) je organizovana u dva nivoa, s tim da je broj ulaza u PMT prvog nivoa dva puta manji od broja ulaza u PMT drugog nivoa.

Posmatra se jedan proces kreiran nad sledećim programom sa greškom:

```
#define M 4
#define N 0x1000
int src[M][N], dst[M][N];

int main (int argc, const char* argv[]) {
  for (int i=0; i<=M; i++)
    for (int j=0; j<=N; j++)
      dst[i][j] = src[i][j];
}
```

pod sledećim pretpostavkama:

- tip `int` je veličine 32 bita; promenljive `i` i `j` je prevodilac formirao kao automatske promenljive na steku;
- logički segment za kod ovog programa veličine je jedne stranice i nalazi se na dnu virtuelnog adresnog prostora procesa (na najnižim adresama), a segment za stek je veličine 32 stranice i nalazi se odmah iznad segmenta za kod;
- nizovi `src` i `dst` smešteni su jedan odmah iza drugog, tim redom, u isti logički segment memorije alociran za statičke podatke koji se nalazi iznad segmenta za stek; ovaj segment je onoliki koliko je najmanje stranica potrebno za smeštanje ovih nizova i dozvoljen je za upis;
- operativni sistem ne učitava nijednu stranicu pri kreiranju procesa, već sve stranice učitava tek na zahtev (*demand paging*).

a)(3) Prikazati logičku strukturu virtuelne adrese i označiti veličinu svakog polja.

Odgovor:

b)(3) Za koje vredosti promenljivih `i` i `j` i pristup do kog od dva niza `src` i `dst` će procesor prvi put generisati izuzetak koji će operativni sistem smatrati kao prestup (grešku) procesa? Obrazložiti.

Rešenje:

c)(4) Koliko straničnih grešaka (*page fault*) će operativni sistem obraditi uspešno za ovaj proces dok ga ne ugasi zbog prestupa, ako je operativni sistem za ovaj proces odvojio dovoljno okvira operativne memorije da smesti sve stranice koje taj proces regularno adresira? Obrazložiti.

Rešenje:

4. (10 poena)

U implementaciji nekog fajl sistema evidencija slobodnih blokova na particiji vodi se pomoću bit-vektora koji se kešira u memoriji u nizu `blocks` veličine `NumOfBytes` (konstanta `NumOfBlocks` predstavlja broj blokova na particiji). Svaki element ovog niza je veličine jednog bajta, a svaki bit odgovara jednom bloku na disku (1-zauzet, 0-slobodan). Pretpostaviti da je `NumOfBlocks` umnožak broja 8 (`BITS_IN_BYTE`). Date su pomoćne funkcije i funkcija `allocateBlock` koja vrši alokaciju slobodnog bloka tražeći prvi slobodan blok počev od onog datog argumentom. Blok 0 je uvek zauzet.

```
typedef unsigned char byte;
typedef unsigned long ulong;
const byte BITS_IN_BYTE = 8;
const ulong NumOfBlocks = ...;
const ulong NumOfBytes = NumOfBlocks/BITS_IN_BYTE;
byte blocks[NumOfBytes];

void blockToBit(ulong blkNo, ulong& bt, byte& mask) {
  bt = blkNo/BITS_IN_BYTE;
  mask = 1<<(blkNo%BITS_IN_BYTE);
}

void bitToBlk(ulong& blkNo, ulong bt, byte mask) {
  blkNo = bt*BITS_IN_BYTE;
  for (; !(mask&1); mask>>=1) blkNo++;
}

ulong allocateBlock (ulong startingFrom) {
  ulong bt = 0; byte msk = 0;
  for (ulong blk = startingFrom; blk<NumOfBlocks; blk++) {
    blockToBit(blk,bt,msk);
    if ((blocks[bt]&msk)==0) {
      blocks[bt] |= msk;
      return blk;
    }
  }
  for (ulong blk =1; blk<startingFrom; blk++) {
    blockToBit(blk,bt,msk);
    if ((blocks[bt]&msk)==0) {
      blocks[bt] |= msk;
      return blk;
    }
  }
  return 0;
}
```

Modifikovati funkciju `allocateBlock` tako da alocira blok koji je najbliži onom koji je tad argumentom, i to najbliži sa bilo koje strane.

Rešenje: