Operacijski sustavi za ugrađena računala

Ispit: 24. 6. 2026.

_____
ime i prezime

Pisati čitko – nečitka rješenja su kriva. U mnogim zadatcima nisu zadane adrese memorija/registara – koristiti zadana imena kao da su adrese (kao da su prethodno definirane, npr. kroz #define ADDR 0x123123). Zadatke 1-2 rješavati na ovom papiru.

1. (1) Napisati makro `ALIGN(X)` koji će vratiti 32-bitovnu vrijednost iz X ali s obrisana četiri najmanje značajna bita.

   najjednostavnije rješenje: **`#define ALIGN(X)  ((X) & 0xFFFFFFF0)`**

   konstantu s zadnja četiri bita obrisana se može dobiti i na druge načine: **`~15`**, **`~(1<<4 - 1)`**

   može se i pomaknuti lijevo, pa onda desno: **`(((X)>>4)<<4)`**

   ovaj makro vraća vrijednost, ne mijenja X! stoga ni do-while nisu ispravni

2. (1) Popraviti makro `#define INCMOD(X,N) X=(X+1)%N` da se može primjenjivati najopćenitije moguće (gdje to ima smisla).

   ovaj makro treba promijeniti X;

   kad bi X bio obična varijabla/ne mijenjajući izraz, dovoljno bi bilo X, N i sve samo staviti u zagrade: **`#define INCMOD(X,N) ((X)=((X)+1)%(N))`**

   do-while nije potreban u ovom slučaju; nije greška ali nije neophodan

   što ako se makro pozove s: **`INCMOD(a[i++], b[j++])`**? onda gornji makro nije dobar jer se X pojavljuje dva puta! N se javlja samo jednom tako da bi za njega bilo OK.

   taj se problem da riješiti složenijim makroom. prvi pokušaj:

   ```
   #define INCMOD(X,N)  do { typeof(X) _x = (X);_x = (_x + 1) % (N); (X)=_x;} while(0)
   ```

   nije baš uspješan jer se opet **X** javlja dva puta (neki su ovako rješavali; neki su zaboravili zadnju naredbu **`(X)=_x;`**)

   jedno rješenje dohvaća adresu od X i s njom radi (tada nam zadnja naredba ne treba):

   ```
   #define INCMOD(X,N)  do { typeof(X) *_x = &(X);*_x = (*_x + 1) % (N);} while(0)
   ```

   nedostatak ovog rješenja je da se ne može koristiti s desne strane =, dok onaj prvi može! npr. **`a = INCMOD(c[i++], d[j++]) * 50;`**

   općenitije rješenje bi bilo uz korištenje proširenja gcc-a:

   ```
   #define INCMOD(X,N)         \
   ({                          \
       typeof(X) *_x = &(X);   \
       *_x = (*_x + 1) % (N);  \
       *_x;                    \
   })
   ```

   netko je predložio rješenje **`((++(X)) %= (N))`** koje je možda također ispravno...

   sva prikazana rješenja su bodovana kao ispravna (s 1 bodom)

3. (2) Navesti nekoliko načina dinamičkog upravljanja memorijom (algoritmi) te njihova dobra i loša svojstva gledano iz perspektive korištenja u ugradbenim sustavima i sustavima za rad u stvarnom vremenu.

   prvi odgovarajući – jednostavna implementacija (brzi); složenost O(N); velika fragmenatcija

   najbolji odgovarajući – jednostavna implementacija (brzi); složenost O(N); minimalna fragmentacija

   TLSF ili slični – složenija implementacija; složenost O(1); manja fragmentacija

4. (2) Izgradnja ugradbenog sustava koji koristi procese zahtjeva nekoliko bitnih izmjena. Navesti koje su sve potrebne izmjene (gdje) i zašto (svi koraci u `Chapter_08_Processes`).

   poziv jezgrinih funkcija programskim prekidom

izvođenje jezgrinih funkcija u privilegiranom načinu rada procesora, dretvi u korisničkom

zasebno prevođenje jezgre i programa

odvajanje adresnog prostora nekim mehanizmom (segmentacija, straničenje)

5. (4) Neki sustav ima nekoliko memorija: `BOOT`, `PROG`, `CONST` i `RAM`. Prve tri su samo za čitanje tj. u njih se upisuju podatci „programiranjem“ posebnim sklopom iz slike sustava koju treba pripremiti. U `BOOT` treba staviti sve nastalo prevođenjem datoteke `boot.c`. U `PROG` sve instrukcije nastale iz svih ostalih datoteka s kodom. U `CONST` sve konstante iz svih ostalih datoteka. Sve varijable koje se mijenjaju pripremiti za učitavanje u `CONST`, ali da rade tek kada se premjeste u `RAM` što treba napraviti funkcija `premjesti()` iz datoteke `boot.c`. Napisati skriptu za prevođenje `ldscript.ld` te funkciju `premjesti`.

ldscript.ld

```
SECTIONS {
      .boot BOOT : AT(BOOT) {
            boot.o (*)
      }
      .prog PROG: AT(PROG) {
            * (.text)
      }
      .const CONST: AT(CONST) {
            * (.rodata)
      }
      vars = CONST + SIZEOF(const);
      .vars RAM : AT(vars) {
            * (.data .bss)
      }
      vars_size = SIZEOF(.vars);
}
```

premjesti()

```
{
      extern char vars, vars_size, RAM;

      size_t size = (size_t) &vars_size;
      char *a = &vars;
      char *b = &RAM;
      int i;
      for (i = i; i < size; i++)
            b[i] = a[i];
}
```

6. (5) Izvorni kôd za neki ugradbeni sustav nalazi se u direktorijima `kernel`, `include` (samo zaglavlja), `io` te `prog`. Pretpostaviti da u svakom od navedenih direktorija već postoji datoteka `files.ini` koja sadrži popis objekata koje treba napraviti pri prevođenju. Npr. u `kernel/files.ini` može pisati: `kernel = core.o mm.o threads.o` itd. U datoteci `config.ini` već postoje definirane varijable `CFLAGS`, `LDDFLAGS` i `LDLIBS` s nekim potrebnim vrijednostima. Napisati `Makefile` za prevođenje sustava i izgradnju slike `firmware.elf` uz korištenje postojeće linker skripte `ldscript.ld` (dodati `-T ldscript.ld` pri povezivanju). Stvaranje popisa svih objekata može se napraviti kroz naredbe oblika: `OBJK = $(addprefix kernel/,$(kernel))`. Nadopuniti varijablu `CFLAGS` „informacijom“ da su zaglavlja u direktoriju `include`. Ukoliko su za nešto dovoljna implicitna pravila nije ih potrebno eksplicitno navoditi u `Makefile`-u.

Makefile

```
include config.ini
include kernel/files.ini #tu je definirana varijabla „kernel“
include io/files.ini     #tu je definirana varijabla „io“
include prog/files.ini   #tu je definirana varijabla „prog“
CFLAGS += -I include
OBJS = $(addprefix kernel/,$(kernel))
OBJS += $(addprefix io/,$(io))
OBJS += $(addprefix prog/,$(prog))

firmware.elf: $(OBJS)
    ld $(LDFLAGS) $(OBJS) $(LDLIBS) -T ldscript.ld -o $(@)
```

7. Neki sustav ima sklop za prihvat prekida s jednim 16-bitovnim statusnim registrom `SR`. Sklop će proslijediti sve zahtjeve za prekid procesoru dok će naprave koje su tražile prekid biti vidljive kroz bitove registra `SR`. Procesor treba obrisati odgovarajući bit `SR`-a (npr. po obradi prekida) jer dok to ne napravi naprava drži svoj zahtjev za prekid (i sklop ga prosljeđuje procesoru).

   a. (3) Ostvariti prekidni podsustav kroz sučelja:

```
void prekidi_inicijaliziraj();
void *prekid_registriraj(int i, void (*obrada)(int));
void prekidi_prihvat();
```

gdje se obrada obavlja s zabranjenim prekidanjem (nema prioriteta prekida).

```
void (*obrade[16])(int);

void prekidi_inicijaliziraj() {
	int i;
	for (i = 0; i < 16; i++)
		obrade[i] = NULL;
}
void *prekid_registriraj(int i, void (*obrada)(int)) {
	void *prije = obrade[i];
	obrade[i] = obrada;
	return prije;
}
void prekidi_prihvat() {
	int i;
	for (i = 0; i < 16; i++) {
		if ((*SR & (1<<i))) {
			if (obrade[i])
				obrade[i](i);
			*SR = (*SR) & ~(1<<i); //uvijek obriši, čak i ako nema reg.f.
		}
	}
}
```

neki su unutar prekidi_prihvat() zabranjivali i dozvoljavali prekidanje – to nije potrebno! pri prihvatu prekida oni su automatski zabranjeni!

b. (4) Ostvariti prekidni podsustav kroz ista sučelja kao u a. ali s obradom prema prioritetu gdje je prioritet jednak indeksu bita u registru SR (korisni dio obrade se treba moći prekidati; dodatnom strukturom podataka pamtiti što se zbiva u sustavu).

bez pamćenja tko je tražio prekid, koji je u obradi i sl. ne može se ispravno riješiti zadatak

```
void (*obrade[16])(int);
int prio[16]; //spremljeni prioriteti – spremi prioritet prekinute obrade
int tp;       //tekući prioritet
int oc[16];   //oznake čekanja – koji prekidi čekaju na početak obrade

void prekidi_inicijaliziraj() {
	int i;
	for (i = 0; i < 16; i++) {
		obrade[i] = NULL;
		prio[i] = 0;
		oc[i] = 0;
	}
	tp = -1;
}
void *prekid_registriraj(int i, void (*obrada)(int)) {
	void *prije = obrade[i];
	obrade[i] = obrada;
	return prije;
}
void prekidi_prihvat() {
	int i, obradjeno;
	do {
		for (i = 0; i < 16; i++) {
			if ((*SR & (1<<i) != 0) {
				if (obrade[i])
```

```
                        oc[i] = 1;
                    *SR = (*SR) & ~(1<<i);
                }
            }
            obradjeno = 0;
            for (i = 15; i > tp; i--) {
                if (oc[i]) {
                    oc[i] = 0;
                    prio[i] = tp;
                    tp = i;
                    dozvoli_prekidanje(); //obavezno!
                    obrade[i](i);
                    zabrani_prekidanje();
                    tp = prio[i];
                    obradjeno = 1;
                    break;
                }
            }
        }
        while (obradjeno);
    }
```

8. (5) Neki sustav ima pet 20-bitovnih brojila: prvo na adresi `BR1` odbrojava s 1 MHz, iduća četiri na adresama `BR2-5` odbrojavaju s frekvencijom od 50 KHz. Brojila odbrojavaju od zadnje učitane vrijednosti do nule; izazivaju prekid; u brojilo učitavaju zadnju postavljenu vrijednost; učitavanjem nule se brojilo zaustavlja (što je početno stanje). Prvo brojilo koristiti samo za ostvarenje precizna sata u jedinici mikrosekunde (`void postavi_sat(uint32 broj);` `uint32 dohvati_sat()`). Ostale iskoristiti za ostvarenje četiri nezavisna alarma (`void postavi_alarm(int id, uint32 za_koliko_usec, void (*funkcija)())`). Postavljanjem alarma `id` poništava se prethodno postavljeni alarm za isti `id`. Brojila izazivaju prekide s brojevima `INTBR1-5`. Registraciju prekida brojila obaviti kroz sučelje `registriraj_prekid(int irq, void (*obrada)(int irq))`.

```
//2^20 = 1048576 => malo preko milion; radi jednostavnosti proračuna može se koristiti
//1000000 kao najveća vrijednost, a i smanjuje se gubitak zbog preciznosti
#define MAXBR 1000000
//50 KHz => 1 otkucaj = 1/50000 = 0,00002 = 20 usec => pretvorba usec=>otk: usec/20

uint32 *br[5] = {BR1, BR2, BR3, BR4, BR5};
uint32 irq[5] = {INTBR1, INTBR2, INTBR3, INTBR4, INTBR5};
uint32 ucitano[5] = {MAXBR, 0, 0, 0, 0};
uint32 odgoda[5] = {MAXBR, 0, 0, 0, 0};
void (*alarm[5])();
uint32 sat = 0;

void init() {
    sat = 0;
    for (int i = 0; i < 3; i++) {
        *br[i] = odgoda[i];
        alarm[i] = NULL;
        registriraj_prekid(irq[i], obrada);
    }
}
void postavi_sat (uint32 broj) {
    sat = broj;
    *br[0] = 0;
}
uint32 dohvati_sat() {
    return sat + odgoda[0] - *br[0];
}
void postavi_alarm(int id, uint32 za_koliko_usec, void (*funkcija)()) {
```

```
    odgoda[id] = za_koliko_usec / 20;
    alarm[id] = funkcija;
    ucitano[id] = min(odgoda[id], MAXBR); //ili s if
    *br[id] = ucitano[id];
}
void obrada(int irq) {
    switch(irq) {
          case INTBR1: i = 0; break;
          case INTBR2: i = 1; break;
          case INTBR3: i = 2; break;
          case INTBR4: i = 3; break;
          case INTBR5: i = 4; break;
          default: return;
    }
    if (i == 0) {
          sat += ucitano[0];
    }
    else {
          if (odgoda[i] == 0)
                return;
          odgoda[id] -= ucitano[id];
          if (odgoda[id] > 0) {
                ucitano[id] = min(odgoda[id], MAXBR); //ili s if
                *br[id] = ucitano[id];
          }
          else {
                odgoda[id] = ucitano[id] = 0;
                *br[id] = ucitano[id];
                alarm[id]();
          }
    }
}
```

9. (4) U nekom sustavu sve kritične poslove rade se ili u obradi prekida ili s periodičkim dretvama koje vrlo brzo obave svoj posao. Stoga je raspoređivač prema obrnutom redu prispijeća (prva se izvodi ona dretva koja je zadnja došla u red pripravnih) dovoljno dobar. Ostvariti takav raspoređivač kroz sučelja:
   `void dodaj_u_pripravne(dretva_t *dretva);`
   `dretva_t *uzmi_prvu_iz_pripravnih();` //pretpostaviti da je uvijek tamo bar jedna kad se poziva ovo
   `void odaberi_aktivnu();` //ako aktivna ne postoji, uzeti prvu iz reda pripravnih
   Definirati sve ostale potrebne strukture podataka. Pretpostaviti da je kontekst dretvi definiran strukturom `kontekst_t` uz postojeću funkcija `zamjeni_dretvu(kontekst_t *stara, kontekst_t *nova)`.

```
opisnik_t *dretva {
    int stanje;
    kontekst_t kontekst;
    opisnik_t *iduca; //za ostvarenje jednostruko povezane liste
}
opisnik_t *aktivna = NULL;
opisnik_t *pripravne = NULL;

void dodaj_u_pripravne(opisnik_t *dretva)
{
    dretva->stanje = PRIPRAVNA;
    dretva->iduca = pripravne;
    pripravne = dretva;
}
opisnik_t *uzmi_prvu_iz_pripravnih() {
    opisnik_t *prva = pripravne;
    pripravne = prva->iduca;
    return prva;
}
```

```
void odaberi_aktivnu() {
    if (aktivna == NULL || aktivna->stanje != AKTIVNA) {
          opisnik_t *stara = aktivna;
          aktivna = uzmi_prvu_iz_pripravnih();
          aktivna->stanje = AKTIVNA;
          zamjeni_dretvu(stara, aktivna);
    }
}
```

10. (4) Ostvariti jednostavni cjevovod koji će koristiti dretve različitih procesa. Neka postoji samo jedan cjevovod u sustavu koji se koristi funkcijama (koje dretve pozivaju):

```
int postavi(size_t velicina_cijevi);
size_t citaj(char *buf, size_t koliko, char **prazno)
size_t pisi(char *buf, size_t koliko, char **neupisano).
```

Neka su funkcije neblokirajuće – čitaju/pišu najviše koliko mogu (do `koliko`) i veličinu pročitanih/upisanih podataka vraćaju preko povratne vrijednosti. U `prazno` funkcija `citaj` stavlja adresu prvog elementa `buf` u koji nije ništa stavila. Slično, u `neupisano` funkcija `pisi` stavlja adresu prvog elementa `buf` koji nije poslan u cijev. Iz ostvarenja gornjih funkcija (koja postoje) mehanizmom prekida pozivaju se ekvivalentne jezgrine funkcije (s prefiksom) `k` koje treba ovdje ostvariti. Ove jezgrine funkcije dobivaju identične argumente od početnih, jedino što su adrese u logičkom zapisu pozivajućeg procesa a jezgrine se funkcije obavljaju s fizičkim adresama. Pretvaranje logičke adrese aktivnog procesa u fizičku i obratno obaviti postojećim funkcijama:

```
void *logička_u_fizičku(void *logička_adresa);
void *fizička_u_logičku(void *logička_adresa);
```

Memoriju za cijev zauzeti s `kmalloc`.

```
static char *buffer = NULL; //kružni međuspremnik
static int velicina = 0, ulaz = 0, izlaz = 0, ima = 0;

int kpostavi(size_t velicina_cijevi) {
    velicina = velicina_cijevi;
    buffer = kmalloc(velicina);
    ulaz = izlaz = ima = 0;
}
size_t kcitaj(char *buf, size_t koliko, char **prazno) {
    char *b = logička_u_fizičku(buf);
    char **p = logička_u_fizičku(prazno);

    size_t kopirati = min(koliko, ima);//ili s if
    int i;
    for (i = 0; i < kopirati; i++) {
          b[i] = buffer[ulaz];
          ulaz = (ulaz + 1) % velicina;
          ima--;
    }
    *p = buf + kopirati; //ili fizicka_u_logicku(b+kopirati)

    return kopirati;
}
size_t pisi(char *buf, size_t koliko, char **neupisano) {
    char *b = logička_u_fizičku(buf);
    char **p = logička_u_fizičku(neupisano);

    size_t kopirati = min(koliko, velicina - ima);
    int i;
    for (i = 0; i < kopirati; i++) {
          buffer[izlaz] = b[i];
          izlaz = (izlaz + 1) % velicina;
          ima++;
    }
```

```
    *p = buf + kopirati;
    return kopirati;
}
```