Operacijski sustavi za ugrađena računala

3. 5. 2024. Međuispit

_____
ime i prezime

1. (2) Funkcije mogu biti dodatno označene ključnim riječima `static`, `inline`, `static inline`. Navesti značenja pojedinog označavanja i kada se ono koristi.

   `static` – vidljivo samo unutar datoteke u kojoj je definirana

   `inline` – sugestija prevoditelju da ju stavi na mjesto poziva; za kratke funkcije

   `static inline` - sugestija prevoditelju da ju stavi na mjesto poziva, za kratke funkcije koje se mogu onda staviti i u zaglavlja te uključiti u više datoteka

2. (1) Napisati makro koji će vratiti prvi idući parni broj od zadanog. Npr. `PARNI(8)` treba vratiti `10`.

   ```
   #define PARNI(X)              (((int)(X) + 2) & ~1)
   ```

   ili `((X) + 2)` ako se zadatak shvati tako da je X paran cijeli broj …

3. (1) Napisati makro `DIJELI(C,O,B,N)` koji će u `C` postaviti cjelobrojni dio od `B/N`, a u `O` decimalni dio. Npr. `DIJELI(C,O,7.5,4)` => `7,5/4=1,875` => `C=1`, `O=0,875`. Makro mora raditi za sve tipove brojeva.

   ```
   #define DIJELI(C,O,B,N)     do{    \
       double x = 1.*(B)/(N);         \
       (C) = (int) x;                 \
       (O) = x - (int) x;             \
   }while(0)
   ```

4. Neki sustav ima `ROM1` (`BOOT`) na adresi 0x1000, `ROM2` (`PROG`) na adresi 0x10000 i `RAM` na adresi 0x50000. Veličina `RAM`-a je 8 KB. Izvorni kod sustava nalazi se u direktorijima `arch`, `kernel` i `progs`.
   a. (4) Napisati skriptu za povezivanje `ldscript.ld` tako da se sve pripremi za smještaj u `PROG` prema redoslijedu:
      1. instrukcije iz datoteke `kernel/startup.c` (tj. funkcija `startup()` koja je jedina u toj datoteci)
      2. instrukcije iz svih ostalih datoteka
      3. konstante iz svih datoteka
      4. podaci iz svih datoteka, ali da ispravno rade tek kad ih se premjesti u `RAM+512`
      5. polje `char stog[512] __attribute__((section(".stog")));` postaviti na kraj `RAM`-a (tj. samo pripremiti za tu adresu, ali postaviti iza ostalih dijelova u `PROG`).

      Prvih 512 B `RAM`-a ne koristiti (koristi se za d) dio zadatka). U skripti definirati varijable `premjesti_adr` i `premjesti_vel` koje trebaju imati adresu podataka u `ROM2` i njihovu veličinu.

      ldscript.ld:

      ```
      PROG = 0x10000;
      RAM = 0x50000;
      KRAJ_RAMa = 0x50000 + 0x2000;
      SECTIONS {
            .ostaje PROG : AT ( PROG ) {
                  kernel/startup.o (.text)
                  * (.text)
                  * (.rodata)
            }
            premjesti_adr = PROG + SIZEOF(.ostaje);
            premjesti_kamo = RAM + 512;
            .premjesti premjesti_kamo : AT ( premjesti_adr ) {
                  * (.data .bss)
            }
            premjesti_vel = SIZEOF(.premjesti);
            .stog KRAJ_RAMa - 512 : AT ( premjesti_adr + premjesti_vel ) {
                  * (.stog)
            }
      }
      ```

b. (1) Napisati kod za premještanje podataka iz ROM2 u RAM, tj. funkciju copy_ROM2_RAM() koja se prva poziva iz početne funkcije startup() u datoteci kernel/startup.c.

```
void copy_ROM2_RAM(){
	extern char premjesti_adr, premjesti_vel, premjesti_kamo;
	int i, vel = (int) &premjesti_vel;
	char *a = &premjesti_adr, *b = &premjesti_kamo;
	for (i = 0; i < vel; i++)
		*b++ = *a++;
}
```

c. (1) Napisati Makefile za prevođenje. Pretpostaviti da postoji datoteka config.ini u kojoj su definirane sve .o datoteke koje treba stvoriti preko varijable OBJS. Također, sve ostale implicitne varijable su tamo već postavljene (CFLAGS, LDFLAGS).

```
Makefile
include config.ini
TARGET = slika.elf
$(TARGET): $(OBJS)
	$(LINK) $(LDFLAGS) $(OBJS) -T ldscript.ld -o $(TARGET)
```

d. (2) U ROM1 se smješta program pokretač (*bootloader*). Njegov izvorni kod se nalazi u direktoriju bootloader s datotekama boot.c i remote.c. U datoteci remote.c nalazi se kod za komunikaciju koji će po detekciji spajanja s upravljačkim računalom (uz neki prekid) preuzeti kontrolu (npr. radi debuggiranja ili ažuriranja programa u ROM2). Kod u boot.c (funkcija boot()) radi:
1. osnovnu inicijalizaciju sustava (boot_init() koja postoji),
2. poziva inicijalizaciju iz remote.c (prvu funkciju iz te datoteke, nepoznata imena) te
3. pokreće sustav pripremljen u ROM2 (funkciju startup() iz kernel/startup.c).
Napisati datoteku boot.c te navesti kako se došlo do adresa potrebnih funkcija.

```
boot.c:
void boot()
{
	boot_init();
	extern char remote_start;  //u skripti koja se koristi za povezivanje boot-a
	void (*remote_init)() = (void *) &remote_start;
	remote_init();
	void (*startup)() = (void *) 0x10000;  //početak PROG-a
	startup();
}
```

5. (5) Neki sustav ima sklop za prihvat prekida sa 8-bitovnim registrima `IR` te `CR`. U registru `IR` su postavljeni bitovi onih naprava čiji zahtjev za prekid čeka početak obrade. Na početku obrade nekog zahtjeva za prekid mora se postaviti odgovarajući bit registra `IR` postaviti u nulu čime će sklop signalizirati napravi da se njen zahtjev započinje obrađivati. Pri upisu u `IR` samo se bitovi označeni s nulom postavljaju u nulu, ostali se ne mijenjaju. Npr. ako je `IR`=00011001, upisom vrijednosti 11110111 nova vrijednost `IR`-a je 00010001. U registru sklopa `CR` definiraju se zahtjevi koji se prosljeđuju procesoru. Npr. `CR`=10001000 označava da se prekidi s ulaza 7 i 3 prosljeđuju, a ostali ne. Ostvariti prekidni podsustav u kojem se prekidi prihvaćaju prema prioritetu gdje je prioritet određen rednim brojem ulaza – ulaz 0 ima najmanji prioritet (npr. 1), a ulaz 7 najveći (npr. 8). Ostvariti sučelja: `void inicijaliziraj(); void registriraj(int ulaz, void (*funkcija)());` te `void prihvat_prekida()` koji se poziva na prekid. Zabraniti prihvat prekida za one ulaze za koje još nije registrirana funkcija. Pri prihvatu prekida mora se ustanoviti prioritet prekida (preko `IR`-a) te manipulacijom registra `CR` omogućiti da se do završetka obrade trenutna prekida prihvaćaju samo prioritetniji prekidi. Indeks najznačajnije jedinice nekog broja neka se može dohvatiti funkcijom `msb(n)`. Postavljanje `i`-tog bita u varijabli `x` napraviti sa: `x = x | (1<<i)`. Brisanje `i`-tog bita u varijabli `x` napraviti sa: `x = x & ~(1<<i)`. Brisanje nižih `i` bitova varijable `x` ostvariti sa: `x = x & ~((1<<i)-1)`.

```
void (*obrada[8])();
int prio = 0, cr = 0;

void inicijaliziraj()
{
    int i;
    for (i=0; i<8;i++)
        obrada[i] = NULL;
    CR = 0;
    IR = 0;
}
void registriraj(int ulaz, void *f)
{
    obrada[ulaz] = f;
    cr |= 1 << ulaz;
    if (ulaz + 1 > prio)
        CR |= 1 << ulaz;
}
```

```
void prihvat_prekida()
{
    int stari_prio = prio;
    prio = msb(IR & CR) + 1;
    IR = ~(1 << prio);
    CR = CR & ~((1 << prio) - 1);
    omogući_prekidanje;
    obrada[prio-1]();
    zabrani_prekidanje;
    prio = stari_prio;
    CR = cr & ~((1 << prio) - 1);
}
```

Treba pamtiti koji su prekidi maskirani – zato se ovdje koristi globalna varijabla **cr**. Teoretski bi se u obradi prekida mogla pozvati funkcija za registraciju nekog drugog prekida…
U protivnom, minimalno treba pohraniti prethodnu vrijednost od **CR** prije brisanja manje značajnih bitova.

6. Neki sustav ima 32-bitovno brojilo na adresi `BROJILO` koje odbrojava frekvencijom f=100 KHz.

a) (3) Ostvariti makroe:

- `ZBROJI(T1,T2)` – zbraja vremena `T1=T1+T2`, oba u formatu `vrijeme_t` (sekunde i mikro-sekunde),
- `ODUZMI(T1,T2)` – oduzima vremena `T1=T1-T2`, pretpostaviti da je `T1>=T2`
- `USPOREDI(T1,T2)` – vraća negativnu vrijednost ako je `T1<T2`, 0 ako su jednaki te pozitivno inače
- `BROJILO_MAX` – najveća vrijednost koja stane u brojilo,
- `BROJ_U_VRIJEME(BROJ,VRIJEME)` – pretvara broj otkucaja zadan u `BROJ` u vrijeme i sprema ga u varijablu `VRIJEME` koje je tipa `vrijeme_t`,
- `VRIJEME_U_BROJ(VRIJEME,BROJ)` – pretvara vrijeme u broj otkucaja i sprema ga u varijablu `BROJ`; ako je `VRIJEME > Tmax` vratiti `BROJILO_MAX` (`Tmax` neka bude globalna varijabla koju izračunati u `inicijaliziraj` pozivom `BROJ_U_VRIJEME(BROJILO_MAX, Tmax)`),

U makroima se može pretpostaviti da su argumenti jednostavne varijable (ne treba ih omeđivati zagradama). Izbjegavati množenja i dijeljenja kad god je to moguće. Paziti na preciznost i prekoračenja pri računanju.

```
struct timespec { long sec, long usec };
```

```
#define ZBROJI(T1, T2)     do { \
    T1.sec += T2.sec;           \
    T1.usec += T2.usec;         \
    if (T1.usec >= 1000000) {   \
        T1.sec++;               \
        T1.usec -= 1000000;     \
    }                           \
}while(0)
```

```
#define ODUZMI(T1, T2)     do{   \
    T1.sec += T2.sec;           \
    T1.usec += T2.usec;         \
    if (T1.usec >= 1000000) {   \
        T1.sec++;               \
        T1.usec -= 1000000;     \
    }                           \
}while(0)
```

```
#define USPOREDI(T1, T2) \
(T1.sec == T2.sec ? T1.usec - T2.usec : T1.sec - T2.sec)

#define BROJILO_MAX         0xffffffff

#define BROJ_U_VRIJEME(BROJ, VRIJEME) do { \
    VRIJEME.sec = (BROJ / 100000);          \
    VRIJEME.usec = ((BROJ % 100000) * 10); \
} while(0)

#define VRIJEME_U_BROJ(VRIJEME, BROJ) do {                  \
    if (USPOREDI(VRIJEME, Tmax) > 0)                        \
        BROJ = BROJILO_MAX;                                 \
    else                                                    \
        BROJ = VRIJEME.sec * 100000 + VRIJEME.usec / 10;    \
} while(0)
```

b) (5) Ostvariti upravljanje vremenom sa sučeljem:

- `void inicijaliziraj(vrijeme_t pocetni_sat);`
- `vrijeme_t dohvati_vrijeme();`
- `vrijeme_t postavi_vrijeme(vrijeme_t novo_vrijeme);` //vrati prijašnje vrijeme, briši alarm
- `void postavi_alarm(void (*obrada)(), vrijeme_t kada);` //`kada` je apsolutno vrijeme
- `void prekid_brojila();`

```
struct timespec Tmax, sat, ucitano, alarm;
void (*aktivacija)();

void inicijaliziraj(vrijeme_t pocetni_sat) {
    sat = pocetni_sat;
    alarm.sec = alarm.usec = 0;
    BROJ_U_VRIJEME(BROJILO_MAX, Tmax);
    ucitano = Tmax;
    BROJILO = BROJILO_MAX;
}
```

```
vrijeme_t dohvati_vrijeme()
{
    vrijeme_t t;
    BROJ_U_VRIJEME(ucitano - BROJILO, t);
    ZBROJI(t, sat);
    vrati t;
}
vrijeme_t postavi_vrijeme(vrijeme_t novo_vrijeme)
{
    vrijeme_t t = dohvati_vrijeme();
    inicijaliziraj(novo_vrijeme);
    vrati t;
}
void postavi_alarm(void (*obrada)(), vrijeme_t kada)
{
    long brojilo;
    vrijeme_t t = kada;
    aktivacija = obrada;
    alarm = kada;
    sat = dohvati_vrijeme();
    ODUZMI(t, sat);
    VRIJEME_U_BROJ(t, brojilo);
    BROJILO = brojilo;
    BROJ_U_VRIJEME(brojilo, ucitano);
}
void prekid_brojila()
{
    ZBROJI(sat, ucitano);
    if (alarm.sec != 0 || alarm.usec != 0) {
        if (USPOREDI(alarm, sat) <= 0) {
            BROJ_U_VRIJEME(BROJILO_MAX, ucitano);
            alarm.sec = alarm.usec = 0;
            BROJILO = BROJILO_MAX;
            aktivacija();
        }
        else {
            long brojilo;
            vrijeme_t t = alarm;
            ODUZMI(t, sat);
            VRIJEME_U_BROJ(t, brojilo);
            BROJILO = brojilo;
            BROJ_U_VRIJEME(brojilo, ucitano);
        }
    }
}
```

7. U neko ugrađeno računalo treba pored postojećih aktivnosti dodati i nekoliko senzora, prikupljati podatke od njih i periodički ih slati. Prvi senzor je spojen na pin 3 i služi za očitanje vlažnosti u zemlji. Vrijednost očitati preko sučelja `float senzor_ucitaj_analogno(int pin)`, te transformirati funkcijom `float pretvori_vlagu(float ocitanje)`. Drugi senzor spojen je na pinove 4 i 5 i služi za očitanje temperature, vlažnosti i tlaka zraka. Očitanje se obavlja tako da se na pin 4 pošalje kodove 0x50, 0x60 ili 0x70 za očitanje redom temperature, vlažnosti ili tlaka preko sučelja `int senzor_posalji(int pin, int protokol, char naredba)` (uz `protokol=62` za ovaj senzor) te potom čita preko pina 5 sučeljem `int senzor_procitaj(int pin, int protokol)`. Dobivene vrijednosti transformirati funkcijama `float izracunaj_temp(int t)` za temperaturu, `float izracunaj_vlagu(int t)` za vlagu te `float izracunaj_tlak(int t)` za tlak.

   a. (3) Napisati upravljački program koji će na zahtjev (poziv `read`) prikupiti očitanja iz svih senzora u polje od četiri podatka tipa `float`, korištenjem sučelja za naprave:

```
struct naprava {
	char ime_naprave[16];
	void inicijaliziraj(struct naprava *n);
	int posalji(struct naprava *n, size_t velicina, void *ms);
	int procitaj(struct naprava *n, size_t velicina_ms, void *ms);
	void *podaci_upravljackog_programa;
}
```

   Sučelja koja upravljački program ne treba postaviti na `NULL`.

```
int procitaj(struct naprava *n, size_t velicina_ms, void *ms) {
	float f[4];

	f[0] = pretvori_vlagu(senzor_ucitaj_analogno(3));

	senzor_pošalji(4, 62, 0x50);
	f[1] = izracunaj_temp(senzor_procitaj(5, 62));

	senzor_pošalji(4, 62, 0x60);
	f[2] = izracunaj_temp(senzor_procitaj(5, 62));

	senzor_pošalji(4, 62, 0x70);
	f[3] = izracunaj_temp(senzor_procitaj(5, 62));

	memcpy(ms, f, 16);
	return 16;
}
struct naprava senzori = {
	.ime_naprave = "senzor341",
	.inicijaliziraj = NULL,
	.posalji = NULL,
	.procitaj = procitaj,
	.podaci_upravljackog_programa = NULL,
};
```

   b. (2) Napisati program koji će definirati periodički alarm koji se aktivira svakih 10 minuta preko postojećeg sučelja `int definiraj_periodicki_alarm(int period_u_sekundama, void (*obrada)())`. Napisati i funkciju za alarm u kojoj preko običnog sučelja `open/close/read` dohvatiti očitanja senzora te očitano poslati preko `int posalji_svima(char *podaci)`, gdje `podaci` mora biti polje od 24 B (korisnu informaciju treba nadopuniti do te veličine ako je manja).

```
void program() {
	definiraj_periodicki_alarm(10*60, akcija);
}
void akcija() {
	int fd = open("senzor341", O_READ);
	char podaci[24];
	memset(podaci, 0, 24);
	read(fd, podaci, 24);
	posalji_svima(podaci);
}
```