Međuispit iz predmeta *Operacijski sustavi za ugrađena računala*, 20. 4. 2023.

Rješavati na košuljici/dodatnim papirima. Pisati ČITKO, nečitka rješenja su kriva.

1. (1) Popraviti makroe:

   a) `#define MNOZI1(X,Y)` `X * Y`

   b) `#define MNOZI2(X,Y)` `X.x *= Y.x; X.y *= Y.y;`

```
#define MNOZI1(X,Y)          ((X) * (Y))
#define MNOZI2(X,Y)          do { (X).x *= (Y).x; (X).y *= (Y).y; while(0)
Ovdje ne pomažu typeof() jer se i dalje X i Y moraju pojaviti dva puta
Krivo rješenje (iako je bodovano kao ispravno):
do { \
    typeof(X) _X_ = (X); //_X_ je lokalna varijabla, nestaje izlaskom iz {} \
    typeof(Y) _Y_ = (Y); //_Y_ je lokalna varijabla, nestaje izlaskom iz {} \
    _X_.x *= _Y_.x;      //nije dovoljno, mora slijediti (X).x = _X_.x \
    _X_.y *= _Y_.y;      //nije dovoljno, mora slijediti (X).y = _X_.y \
} while(0)
```

2. (1) Ako se makro
```
#define LOG(LEVEL, format, ...) \
    printf("[" #LEVEL ":%d]" format "\n", __LINE__, ##__VA_ARGS__)
```
   pozove sa: `LOG(GRESKA, "A je prevelik! A=%d", A);` iz linije 75 kako će se on prevesti u prvom koraku prevođenja (u preprocessing fazi)?

```
printf("[GRESKA:%d]A je prevelik! A=%d\n", 75, A);
```

3. a) (4) Zadane funkcije, varijable i konstante postaviti u odgovarajuće datoteke tako da sve s prefiksom `a_` budu u datotekama `a.h` i `a.c`, s prefixom `b_` u `b.h` i `b.c` te `main()` u `main.c`. Pritom koristiti uobičajenu praksu (što ide u .h treba tamo ići, koristiti oznake `static`, `inline`, `extern` i slično ako i gdje je to potrebno, uključiti potrebna zaglavlja u odgovarajuće .h i .c datoteke). Ono što nije potrebno (ne koristi se izravno) izvan `a_` dijela neka ne bude vidljivo izvan njega. Isto vrijedi i za `b_` dio.

   b) (2) Napisati `Makefile` za prevođenje navedenih datoteka.

```
float a_rez = 1;

float a_kratki(float x)
{
    return x / 42;
}

float a_skaliranje(float x)
{
    if (a_rez > x)
        return 10;
    return 1;
}

float a_dugi(float x)
{
    neki dugi dio koda
    float y = rezultat proračuna
    a_rez += a_kratki(y)
    return a_rez * a_skaliranje(x);
}
```

```
float b_rez = 0;

float b_kratki(float x)
{
    return a_kratki(x) * b_rez;
}

float b_dugi(float x)
{
    "neki dugi dio koda"
    float y = rezultat proračuna
    b_rez += a_kratki(x) * b_kratki(y);
    return b_rez * b_skaliranje(y);
}

int main()
{
    float rez = a_dugi(b_dugi(25));
    return rez < 37;
}
```

Funkcije `a_kratki`, `a_skaliranje` i `b_kratki` su očito kratke i poželjno ih je ugraditi na mjesto poziva umjesto pozivati.

Funkcija `a_kratki` se koristi iz `a_dugi` te iz `b_dugi` – mora biti vidljiva i van `a.c`: rješenje je postaviti ju u `a.h`, ali označiti sa `static inline` (jer će biti prisutna u više datoteka, a `inline` je samo preporuka prevoditelju koju on može ignorirati).
Funkcija `a_skaliranje` se koristi samo u `a_dugi`, tj. može se ostvariti u `a.c` bez oglašavanja u `a.h`. Slično je i s funkcijom `b_kratki` koja treba biti samo u `b.c`.
Zaglavlja `a.h` i `b.h` (barem `a.h`) moraju biti zaštićena od višestrukog uključivanja s `#pragma once` ili `#ifndef _A_H_` i slično. U ovom primjeru se vrlo opreznim uključivanjem to može i izbjeći, ali zašto?

a.h
```
#pragma once

static inline float a_kratki(float x) {
    return x / 42;
}
float a_dugi(float x);
```

a.c
```
#include "a.h"

static float a_rez = 1; //nikako u .c!

static inline float a_skaliranje(float x){
    if (a_rez > x)
          return 10;
    return 1;
}
float a_dugi(float x) {
    neki dugi dio koda
    float y = rezultat proračuna
    a_rez += a_kratki(y)
    return a_rez * a_skaliranje(x);
}
```

b.h
```
#pragma once

float b_dugi(float x);
```

b.c
```
#include "b.h"
#include "a.h"

static float b_rez = 0; //nikako u .c!

static inline float b_kratki(float x) {
    return a_kratki(x) * b_rez;
}
float b_dugi(float x) {
    "neki dugi dio koda"
    float y = rezultat proračuna
    b_rez += a_kratki(x) * b_kratki(y);
    return b_rez * b_skaliranje(y);
}
```

main.c
```
#include "a.h"
#include "b.h"

int main() {
    float rez = a_dugi(b_dugi(25));
    return rez < 37;
}
```

b)
```
CFLAGS = -MMD
main: main.o a.o b.o
      gcc main.o a.o b.o -o main
-include *.d
```

ili

```
main: main.o a.o b.o
      gcc main.o a.o b.o -o main
a.o: a.c a.h
b.o: b.c b.h a.h
main.o: main.c a.h b.h
```

4. Zadan je **`Makefile`**:

```
CFLAGS = -Iinclude -MMD
LDFLAGS = -O2
LDLIBS = -lm
CC = cc

OBJEKTI = main.o zaprimi.o obradi.o ispisi.o
PROGRAM = program

$(PROGRAM): $(OBJEKTI)
    $(CC) $(LDFLAGS) $(OBJEKTI) $(LDLIBS) -o $(PROGRAM)

-include *.d
```

Uz pretpostavku da odgovarajuće `.c` i `.h` datoteke postoje, napisati koje će se naredbe pokretati sa naredbom `make`:
a) (2) pri prvom prevođenju (nema ni jedne `.o` i `.d` datoteke)

```
cc -Iinclude -MMD -c main.c
cc -Iinclude -MMD -c zaprimi.c
cc -Iinclude -MMD -c obradi.c
cc -Iinclude -MMD -c ispisi.c
cc -O2 main.o zaprimi.o obradi.o ispisi.o -lm -o program
```

b) (1) nakon barem jednog prevođenja, kada sve „pomoćne“ datoteke (`.o`, `.d`) već postoje, ali se naknadno promijenio kod u datoteci `zaprimi.c`.

```
cc -Iinclude -MMD -c zaprimi.c
cc -O2 main.o zaprimi.o obradi.o ispisi.o -lm -o program
```

5. (5) Izvorni kod za neki sustav podijeljen je u direktorije: `bootloader`, `kernel` i `programs`. Napisati skriptu za povezivanje tako da se sustav pripremi za upisivanje u ROM na adresi 0x10000, ali da ispravno radi tek kad se premjesti:

   a) sve iz `kernel` u blok memorije na adresi 0x20000

   b) sve iz `programs` u blok memorije na adresi 0x30000

   U skriptu ugraditi potrebne varijable koje će biti potrebne u funkciji `premjesti()` (nju ne pisati) koja se nalazi u početnom kodu u `bootloader` – kod i podatke iz tog direktorija ne premještati, on se koristi samo pri pokretanju.

```
ldscript.ld
SECTIONS {
    .boot 0x10000 :
    {
          bootloader* (*)
    }
    kernel_rom_start = 0x10000 + SIZEOF(.boot);
    .kernel 0x20000 : AT(kernel_rom_start)
    {
          kernel* (*)
    }
    kernel_rom_size = SIZEOF(.kernel);
    programs_rom_start = 0x10000 + kernel_rom_size;
    .programs 0x30000 : AT(programs_rom_start)
    {
          kernel* (*)
    }
    programs_rom_size = SIZEOF(.programs);
}
```

6. Što je sve potrebno za korištenje prekida od:
   a. (2) procesora – kada i kako on mora prihvaćati prekide

      Procesor mora na kraju svake instrukcije provjeriti je li na prekidnom ulazu postoji signal zahtjeva za prekid. Ako postoji onda ga prihvaća: zabrani daljnje prekidanje; prebaci se u prekidni način rada; na stog pohrani PC i RS; u PC stavi adresu prekidnog potprograma

   b. (2) sklopa za prihvat prekida – što on mora raditi te dodatno omogući preko „programiranja“

      Sklop za prihvat prekida prihvaća zahtjeve od naprava te ih prosljeđuje procesoru zajedno s identifikacijom (prekidnim brojem).

      Dodatno treba omogućiti programiranje – koji zahtjevi se prosljeđuju procesoru a koji ne.

      Opcionalno može imati i prioritete, pa prosljeđivati nove zahtjeve samo ako su prioritetniji od onog što procesor trenutno radi.

   c. (2) programske potpore – koje sve funkcije treba ostvariti?

      Funkciju za registraciju prekida – povezivanje prekidnog broja s obradom

      Prekidni potprogram ili za svaki prekid svoji, koji se pozivaju na prekide – utvrđuje se tko je dao zahtjev i što dalje – koju registriranu funkciju pozvati

7. (2) Što je sve potrebno da bi se upravljano nekom napravom?
   a. Kako pripremiti upravljački program?

      Upravljački program mora implementirati sučelje za taj sustav (npr. device_t)

   b. Što OS mora imati?

      OS mora imati „podsustav za UI“ koji treba omogućiti ugradnju i povezivanje s upravljačkim programima naprava

8. (6) Neki sustav ima 20-bitovno brojilo na adresi `BROJILO` koje odbrojava frekvencijom od 500 KHz. Kad dođe do nule izaziva prekid (na koji se poziva funkcija `prekid_brojila`) i u brojilo učitava zadnju poslanu vrijednost. Definirati potrebnu strukturu podataka te napisati funkcije kojima će biti ostvaren podsustav za upravljanje vremenom s jednim satom u sekundama i mikrosekundama te mogućnošću postavljanja jednog alarma. Pretpostaviti da postoje slijedeće pomoćne funkcije za rad s vremenom u obliku strukture s dva elementa `{.s,.us}`: `usporedi(t1, t2)` (vraća „t1-t2“), `zbroji(t1,t2)` (t1+=t2) i `oduzmi(t1,t2)` (t1-=t2). Najprije ostvariti pomoćne funkcije `brojilo_u_vrijeme(br=>t)`, `vrijeme_u_brojilo(t=>br)` te `postavi_u_brojilo(t)` koja preračunava vrijeme u otkucaje i postavlja ih u brojilo ako tamo stanu, inače u brojilo postavlja najveću vrijednost (a može se tamo i druge varijable postaviti, npr. varijable koje pamte što je učitano u brojilo, u broju otkucaja i/ili vremenu).

Konstante i varijable (s početnim vrijednostima)

```
F = 500000 //Hz
T1 = 1/F = 2 µs po otkucaju, T1US=2
BMAX = 0xfffff = 1048575
//TMAX = BMAX * T1 = BMAX/F = 2,09715 s
//     = 2 s + 97150 µs
TMAX = {.s = 2, .us = 97150}

sat = {.s = 0, .us = 0} - sat
b_učitano = BMAX - zadnje učitano u brojilo
t_učitano = TMAX - zadnje učitano, u vremenu
t_alarm = {.s=0, .us=0} - kada aktivirati
f_alarm = NULL - funkcija koju pozvati

//pomoćne funkcije - zadane
usporedi(t1, t2)
{//još optimiranije
   tmp = t1.s - t2.s
   ako je tmp != 0
      vrati tmp
   inače
      vrati t1.us - t2.us
}
zbroji(t1, t2)
{
   t1.s += t2.s
   t1.us += t2.us
   ako je t1.us > 1000000 {
      t1.s += 1
      t1.us -= 1000000
   }
}
oduzmi(t1, t2) (pretp. t1>=t2)
{
   t1.s -= t2.s
   t1.us -= t2.us
   ako je t1.us < 0 {
      t1.s -= 1
      t1.us += 1000000
   }
}

brojilo_u_vrijeme(br => t)
{
   t.s = br / F
   t.us = (br % F) * T1US
}
vrijeme_u_brojilo(t => br)
{
   br = t.s * F + t.us / T1US
}
```

```
postavi_u_brojilo(t)
{
   ako je usporedi(t, TMAX) >= 0 onda {
      b_učitano = BMAX
      t_učitano = TMAX
   }
   inače {
      br = vrijeme_u_brojilo(t)
      b_učitano = br
      t_učitano = t
   }
   BROJILO = b_učitano
}
inicijaliziraj()
{
   sat.s = sat.us = 0
   postavi_u_brojilo(TMAX)
   t_alarm = {0,0}
}
dohvati_sat(t) {
   brojilo_u_vrijeme(b_učitano-BROJILO,t)
   zbroji(t, sat)
}
postavi_sat(t) {
   init()
   sat = t
}
postavi_alarm(kada, alarm)
{
   brojilo_u_vrijeme(b_učitano-BROJILO,tmp_t)
   zbroji(sat, tmp_t)
   f_alarm = alarm
   t_alarm = kada
   tmp_t = t_alarm
   oduzmi(tmp_t, sat)
   postavi_u_brojilo(tmp_t)
}
prekid_brojila()
{
   zbroji(sat, t_učitano)
   ako je t_alarm.s + t_alarm.us > 0 onda
   {
      ako je usporedi(t_alarm, sat) <= 0 onda
      {
         postavi_u_brojilo(TMAX)
         t_alarm = {0,0}
         f_alarm()
      }
      inače {
         tmp_t = t_alarm
         oduzmi(tmp_t, sat)
         postavi_u_brojilo(tmp_t)
      }
   }
}
```