Međuispit iz predmeta Operacijski sustavi za ugrađena računala
26. 4. 2022. Student: _____

Zadan je program u nekoliko datoteka (za zadatke 1-3):

fib.c
```
#include <stdlib.h>
#include <string.h>

static int zadnji = 1, predzadnji = 1;
static void ispisi();

int daj_max_fib_do_n (int n) {
    int iduci = zadnji + predzadnji;
    while (iduci < n) {
        predzadnji = zadnji;
        zadnji = iduci;
        iduci = zadnji + predzadnji;
    }
    ispisi();
    return zadnji;
}
static void ispisi() {
    int i;
    char buf[15];
    volatile char *serija = (char*) 0x1234;

    memset(buf, 0, 15);
    itoa(zadnji, buf, 10);
    for (i = 0; buf[i]; i++)
        *serija = buf[i];
}
```

sqrt.c
```
#include <stdlib.h>
#include <string.h>
#include <math.h>

static double zadnji = (A);
static double sqrt_zadnji = 215.3;

static void ispisi();

double daj_max_sqrt_do_n (int n) {
    double iduci = zadnji + 1;
    double sqrt_iduci = sqrt(iduci);
    while (sqrt_iduci < n) {
        zadnji = iduci;
        sqrt_zadnji = sqrt_iduci;
        iduci = zadnji + 1;
        sqrt_iduci = sqrt(iduci);
    }
    ispisi();
    return sqrt_zadnji;
}
static void ispisi() {
    int i;
    char buf[15];
    volatile char *serija = (char*) 0x123C;

    memset(buf, 0, 15);
    ftoa(zadnji, buf, 10);
    for (i = 0; buf[i]; i++)
        *serija = buf[i];
}
```

(samo **bold crveno** je neophodno za bodove)

prim.c
```
#include <stdlib.h>
#include <string.h>

static int zadnji = (A);
static void ispisi();

int daj_max_prim_do_n (int n) {
    int i, j;
    for (i = zadnji; i < n; i++) {
        for (j = 2; j < i/2; j++)
            if (i % j == 0) break;
        if (j == i/2) zadnji = i;
    }
    ispisi();
    return zadnji;
}
static void ispisi() {
    int i;
    char buf[15];
    volatile char *serija = (char*) 0x1238;

    memset(buf, 0, 15);
    itoa(zadnji, buf, 10);
    for (i = 0; buf[i]; i++)
        *serija = buf[i];
}
```

main.c
```
#include <time.h>
int daj_max_fib_do_n (int n);
int daj_max_prim_do_n (int n);
double daj_max_sqrt_do_n (int n);

//#define A 13 => preko Makefile-a za sve
#define B    (A) * 37
static void ispisi();
static int maxn;

int main() {
    int i, fib, prim;
    double sqrtn;
    struct timespec t;

    t.tv_sec = t.tv_nsec = 0;
    for (i = A; i < B + 1; i++) {
        fib = daj_max_fib_do_n (i);
        prim = daj_max_prim_do_n (i);
        sqrtn = daj_max_sqrt_do_n (i);
        if (fib > prim) maxn = fib;
        else maxn = prim;
        if ((int)sqrtn > maxn)
            maxn = (int)sqrtn;
        ispisi();
    }
    return 0;
}
static void ispisi() {
    volatile char *serija = (char*) 0x1230;
    *serija = '0' + maxn % 7;
}
void premjesti() {…}
```

1. (2) **Popraviti** zadani kod dodavanjem odgovarajućih ključnih riječi, … (riješiti na ovom papiru).

2. (4) Napisati pripadni **`Makefile`** prema uobičajenim pravilima uz korištenje uobičajenih implicitnih varijabli (`CFLAGS, LDFLAGS, LDLIBS`) i implicitna pravila za prevođenje (za povezivanje napisati potpune upute). U kodovima se koristi vrijednost `A` koju treba pomoću `Makefile`-a zamijeniti s vrijednošću zbroja varijabli okoline `BROJ1` i `BROJ2` (ako nisu postavljene, postaviti ih u `Makefile`-u na vrijednosti 7 i 77). Izlazni program neka se zove `program`. Neka se kod optimira obzirom na veličinu (zastavica `-Os`). Pri povezivanju koristiti skriptu za povezivanje (dodati `-T skripta.ld`, skriptu napisati u okviru slijedećeg zadatka).

```
Makefile
BROJ1 ?= 7      # može i bez ?
BROJ2 ?= 77     # može i bez ?
CFLAGS = -D A=($(BROJ1)+$(BROJ2))
LDFLAGS = -Os -T ldscript.ld
LDLIBS = -lm
OBJS = main.o sqrt.o prim.o fib.o
program: $(OBJS)
   $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@
```

3. (6) Napraviti **skriptu za povezivanje** te funkciju **`premjesti()`** tako da se program pripremi za učitavanje na adresu 0x10000, ali da ispravno radi tek kad se dijelovi premjeste na druge adrese, prema podjeli:

   a. sve iz datoteke `fib.c` u blok memorije na adresi 0x20000

   b. sve iz datoteke `prim.c` u blok memorije na adresi 0x30000

   c. sve iz datoteke `sqrt.c` u blok memorije na adresi 0x40000

   d. sve iz datoteke `main.c` u blok memorije na adresi 0x50000, osim funkcije `premjesti()` koja treba ostati tamo gdje je početno učitana (u bloku koji počinje s 0x10000).

   U skriptu ugraditi potrebne varijable koje koristiti u funkciji `premjesti()` koja se poziva prva, prije `main`, a koja treba premjestiti zadane dijelove na navedene adrese. Po potrebi dodatno označiti tu funkciju u kodu.

```
SECTIONS {
  fib_start = 0x10000;
  .fib 0x20000 : AT(fib_start)
  {
    fib.o(*)
  }
  fib_size = SIZEOF(.fib);

  prim_start = fib_start + fib_size;
  .prim 0x30000 : AT(prim_start)
  {
    prim.o(*)
  }
  prim_size = SIZEOF(.prim);

  sqrt_start = prim_start + prim_size;
  .sqrt 0x40000 : AT(sqrt_start)
  {
    sqrt.o(*)
  }
  sqrt_size = SIZEOF(.sqrt);
  premjesti_start = sqrt_start + sqrt_size;
  .premjesti premjesti_start: AT(premjesti_start)
  {
    main.o(.premjesti)
  }
  premjesti_size = SIZEOF(.premjesti);

  main_start = premjesti_start + premjesti_size;
  .premjesti 0x50000 : AT(main_start)
  {
    main.o(*)
  }
  main_size = SIZEOF(.main);
}
```

```
void premjesti()
__attribute__((section(".premjesti")))
{
  extern char
  fib_start, fib_size,
  prim_start, prim_size,
  sqrt_start, sqrt_size,
  main_start, main_size;
  char *od, *kamo;
  size_t koliko, i;

  koliko = (size_t) &fib_size;
  od = &fib_start;
  kamo = (char *) 0x20000;
  for (i = 0; i < koliko; i++)
    kamo[i] = od[i];

  koliko = (size_t) &prim_size;
  od = &prim_start;
  kamo = (char *) 0x30000;
  for (i = 0; i < koliko; i++)
    kamo[i] = od[i];

  koliko = (size_t) &sqrt_size;
  od = &sqrt_start;
  kamo = (char *) 0x40000;
  for (i = 0; i < koliko; i++)
    kamo[i] = od[i];

  koliko = (size_t) &main_size;
  od = &main_start;
  kamo = (char *) 0x50000;
  for (i = 0; i < koliko; i++)
    kamo[i] = od[i];
}
```

4. (3) Neki procesor ima integriran sklop za prihvat prekida. On se programira na način da se u 32-bitovni registar `IRQE` postavi broj čije jedinice predstavljaju omogućene prekidne ulaze (one koji se prihvaćaju) te da se u registar `IRQT` postavi adresa u memoriji gdje se nalazi tablica s adresama funkcija za obradu prekida. Ulazi su numerirani od 0 do 31. Npr. ako je bit 7 u registru `IRQE` postavljen u 1 onda će se prekid naprave koja je spojena na ulaz 7 prihvatiti te će se pozvati funkcija za obradu tog prekida koja se treba nalaziti na adresi `IRQT+7*sizeof(void*)`. Ostvariti **prekidni podsustav** sa sučeljima `void inicijaliziraj()` i `void registriraj_funkciju(int irq, void *funkcija)`. Dok se neka funkcija ne registrira za neki prekid on mora biti onemogućen u sklopu. Također, ako je argument `funkcija` jednak `NULL` u pozivu `registriraj_funkciju`, onda se zadani prekid treba zabraniti.

```
void *irqt[32]={0};

void inicijaliziraj()
{
   IRQE = 0;
   IRQT = irqt;
}
```

```
void registriraj_funkciju(int irq, void *funkcija) {
   irqt[irq] = funkcija
   if (funkcija != NULL)
         IRQE = IRQE | (1<<irq);
   else
         IRQE = IRQE & ~(1<<irq);
}
```

5. (2) Popraviti sljedeće makroe (ovdje ili na papirima):

```
#define MAX(X,Y)                  X > Y ? X : Y
//primjer poziva: t = MAX(i+1, j+k, k*t); //krivi primjer s 3 argumenta

#define POVECAJ(A,B,C,D)          A++ ; B++ ; C++ ; D = A + B + C ;
// primjer poziva: if (x > y)
//                     POVECAJ(x, y, *(z+j), w);
//                 else
//                     x = y;
```

```
#define MAX(X,Y)            ((X)>(Y)?(X):(Y)) //ili MAX(X,Y,Z) (rj. s 3 arg.)
#define POVECAJ(A,B,C,D)    do { (A)++;(B)++;(C)++;(D)=(A)+(B)+(C); } while(0)
```

6. (3) Programeri X i Y rade zajedno na nekom projektu za koji koriste alat git. U nekom trenutku sadržaj datoteke `radno` koja se nalazi u zajedničkom repozitoriju jest:

```
A
B
C
D
```

Programeri sada rade paralelno (lokalno, nad svojom kopijom repozitorija): X dodaje redak sa znakom `X` na početak datoteke, a Y mijenja redak sa znakom `C` u znak `Y`. Programer X tada prvi pokreće naredbe:

```
git add radno
git commit -m "+X"
git push
```

Programer Y nakon toga iste naredbe (uz `C=>Y` umjesto `+X` u komentaru naredbe `commit`).

a) Hoće li X uspjeti napraviti zadano bez grešaka? Ako ima grešaka što mora X napraviti da ih otkloni?

b) Hoće li Y uspjeti napraviti zadano bez grešaka? Ako ima grešaka što mora Y napraviti da ih otkloni?

c) Koji je konačni sadržaj datoteke `radno` (nakon otklanjanja svih grešaka i unosa obje promjene)?

a) X će uspjeti bez grešaka

b) Y neće uspjeti napraviti push; najprije treba napraviti pull koji će zahtijevati dodatni commit, jer se promjene spajaju s onima napravljenim od X-a, pa tek onda push

c) X A B Y D (svaki u svom redu)