(Početni dio za zadatke 1-5) Neki sustav za nadzor i upravljanje ostvaruje se pomoću ugrađenog računala. Izvorni kod nalazi se u direktorijima `kernel` (`boot.c`, `kernel.c`, `interrupts.c`, `time.c`, `net.c` i `startup.c`) i `programs` (`p1.c`, `p2.c` i `p3.c`).

1. (4) Ugrađeno računalo ima ROM na adresi 0x10000 te RAM na adresi 0x100000. Napisati skriptu za prevođenje `ldscript.ld`, kod datoteke `boot.c` te `Makefile`. Instrukcije i konstante pripremiti za učitavanje i korištenje iz ROM-a, a sve ostalo za učitavanje u ROM ali za korištenje iz RAM-a – kod u `boot.c` treba te dijelove kopirati u RAM pri pokretanju te potom pozvati funkciju `start_kernel()`. Pretpostaviti da su u zastavice `CFLAGS`, `LDFLAGS` i `LDLIBS` već postavljene potrebne vrijednosti, a mogu se koristiti i implicitna pravila za određene ciljeve.

```
ldscript.ld
-----------
ROM = 0x10000;
RAM = 0x100000;

SECTIONS {
    .boot ROM : AT (ROM) {
        kernel/boot.o (*)
        * (.text .rodata)
    }
    start = ROM + SIZEOF(.boot);
    .ostalo RAM : AT (start) {
        * (*)
    }
    kraj = start + SIZEOF(.ostalo);
}

boot.c
------
void boot() {
    extern char start, kraj, ROM, RAM;
    size_t i;
    char *a, *b;

    for (a = &start; a < &kraj; a++)
        *a = *b++;

   start_kernel();
}
Makefile
------
PROJECT = project
OBJECTS = boot.o kernel.o interrupts.o time.o net.o startup.o \
p1.o p2.o p3.o

$(PROJECT): $(OBJECTS)
    $(CC) $(LDFLAGS) $OBJECTS -o $(PROJECT) -T ldscript.ld $(LDLIBS)

# teoretski, uz sve zastavice već u *FLAGS dovoljna bi bila i jedna
# linija, npr. boot.elf: boot.o kernel.o ...(itd.)
```

2. (2) Za upravljanje prekidima sustav raspolaže sklopom za prihvat prekida koji ima 8 ulaza: P0-P7. Sklop ima interno dva 8-bitovna registra IR i CP te polje HF[8] (s početkom na adresi 0x4FE2210) u koje treba zapisati adrese funkcije za obradu odgovarajućih prekida. Zahtjev za prekid neke naprave, spojen na neki od ulaza Px, prosljeđuje se u registar IR – odgovarajući bit tog registra se postavlja u 1. Sklop uspoređuje poziciju najviše postavljenog bita iz IR i CP, te ako je najviše postavljeni bit u IR na većoj poziciji od onog u CP te ako je odgovarajuće polje u HF[x] postavljeno (različito od nule) započinje postupak prihvata prekida. U postupku prihvata prekida procesor pohranjuje registre PC i SR na stog te dohvaća adresu iz odgovarajućeg HF[x] u PC (započinje obradu prekida). Istovremeno, sklop za prihvat prekida miče najviši postavljeni bit iz IR u CP. Po završetku obrade prekida, osim što procesor obnavlja PC i SR sa stoga daje signal sklopu za prihvat prekida koji tada briše najviši postavljeni bit iz CP. Ostvariti podsustav za upravljanje prekidima (u `kernel/interrupts.c`) preko sučenja `void init()` i `void register(int irqn, void *hf)`. Korištenjem tog sučelja u funkciji `void start_kernel()` (u `kernel/startup.c`) registrirati funkcije: `time_interrupt()` koji je spojen na P4, `net_interrupt()` koji je spojen na P6 te `sensor_interrupt()` koji je spojen na P7.

```
interrupts.c
------------
void init()
{
    void *HF = (void *) 0x4FE2210;
    int i;
    for (i = 0; i < 8; i++)
        HF[i] = NULL;
}
void register(int irqn, void *hf)
{
    void *HF = (void *) 0x4FE2210;
    HF[irqn] = hf;
}

startup.c
---------
void start_kernel()
{
    register (4, time_interrupt);
    register (6, net_interrupt);
    register (7, sensor_interrupt);
}
```

3. (4) Za upravljanje vremenom na raspolaganju je 24 bitovno brojilo na adresi 0x1CA880 koje frekvencijom od 50 MHz odbrojava od učitane vrijednosti do nule (pa izaziva prekid). Ostvariti podsustav za upravljanje vremenom (sat u `struct timespec {.tv_sec, .tv_nsec}` formatu, jedan alarm) nadopunom datoteke `kernel/time.c`:

```
#define MAXCNT  0xFFFFFF
#define T1NS    ___________ //koliko traje 1 otkucaj u ns - navesti!
#define TIME2NS(X)      ( (X).tv_sec * 1000000000 + (X).tv_nsec )
#define TIME2CNT(X)     ( TIME2NS(X) / T1NS )
#define CNT2TIMENS(X)   ( (X) * T1NS )
#define CNT             ((int*)0x1CA880)
#define SETCNT(X)       do *CNT = (X); while(0)
#define GETCNT()        (*CNT)
static struct timespec clock;
```

```
static struct timespec alarm_delay;
static unsigned last_load;
static void (*alarm_func)();

void time_init() {/* ostvariti */}
void time_set(struct timespec *t) {/* ostvariti */} //... i obr. alarm
struct timespec time_get() {/* ostvariti */}
void time_alarm(struct timespec *t, void (*alarm)()) {/* ostvariti */}
void time_interrupt() {/* ostvariti */}
```

Zanemariti eventualne probleme s preljevom pri množenju cijelih brojeva.

```
...
#define T1NS    20
...
void time_init()
{
    clock.tv_sec = clock.tv_nsec = 0;
    alarm_delay.tv_sec = alarm_delay.tv_nsec = 0;
    alarm_func = NULL;
    last_load = MAXCNT;
    SETCNT(last_load);
}
void time_set(struct timespec *t)
{
    time_init();
    clock = *t;
}
struct timespec time_get()
{
    struct timespec t = clock; //trenutna vrijednost zapisana u satu
    t.tv_nsec += CNT2TIMENS(GETCNT()); //proteklo od ažuriranja sata
    t.tv_sec += t.tv_nsec / 1000000000;
    t.tv_nsec = t.tv_nsec % 1000000000;
    return t;
}
void time_alarm(struct timespec *t, void (*alarm)())
{
    clock.tv_nsec += CNT2TIMENS(last_load - GETCNT());
    clock.tv_sec += t.tv_nsec / 1000000000;
    clock.tv_nsec = t.tv_nsec % 1000000000;

    alarm_delay = *t;
    last_load = TIME2CNT(alarm_delay);
    if (last_load > MAXCNT)
        last_load = MAXCNT;
    SETCNT(last_load);
    alarm_func = alarm;
}
void time_interrupt()
{
    clock.tv_nsec += CNT2TIMENS(last_load);
    clock.tv_sec += t.tv_nsec / 1000000000;
    clock.tv_nsec = t.tv_nsec % 1000000000;
```

```
    if (TIME2NS(alarm_delay) > 0) {
        unsigned remain = TIME2NS(alarm_delay)-CNT2TIMENS(last_load);
        if (remain > 0) {
            alarm_delay.tv_sec = remain / 1000000000;
            alarm_delay.tv_nsec = remain % 1000000000;
            last_load = TIME2CNT(alarm_delay);
            if (last_load > MAXCNT)
                last_load = MAXCNT;
            SETCNT(last_load);
        }
        else {
            last_load = MAXCNT;
            SETCNT(last_load);
            alarm_func();
        }
    }
}
```

4. (4) Za komunikaciju s drugim uređajima ugrađeno računalo ima mrežnu karticu. Kada stigne novi paket preko mreže, on se korištenjem izravnog pristupa spremniku (DMA) prebacuje u radni spremnik na adresu koja treba biti zapisana u mrežnu karticu na adresi 0x72F4180 (maksimalna veličina paketa je 128 B). Po prebacivanju paketa mrežna kartica izaziva prekid i upisuje veličinu paketa na adresu 0x72F4184. Tek po obradi prekida i upisivanja vrijednosti nula na adresu 0x72F4184 mrežna kartica može zaprimati iduće pakete s mreže. Slanje paketa može se obaviti tek ako je na adresi 0x72F4188 nula i to tako da se na adresu 0x72F418C upiše duljina paketa u oktetima te na adresu 0x72F4190 upiše početna adresa podataka u spremniku. Naprava će kraj slanja dojaviti prekidom i postavljanjem nule na adresi 0x72F4188 (tek tad se može osloboditi memorija u kojoj je paket). Korištenjem opisane mrežne kartice ostvariti podsustav za upravljanje mrežom koji će dodatno koristiti dva reda poruka, jedan za dolazne pakete a jedan za odlazne. Potrebno sučelje koje treba ostvariti u `kernel/net.c` jest:

```
//već definirano (npr. u include/net.h)
struct packet {size_t size; char data[1];};
//samo se "data" dio šalje/prima preko mreže
//ostvariti u kernel/net.c
void net_init() {}
void net_send(struct packet *packet) {}
int net_recv(struct packet **packet) {} //vraća veličinu primljena
//paketa, u packet stavlja adresu primljena paketa koju net_recv treba
//rezervirati s kmalloc
void net_interrupt() {}
```

Pretpostaviti da na na raspolaganju stoje operacije `memcpy`, `kmalloc`, `kfree` te operacije za rad s redom poruka: `id = mq_create()`, `void mq_send(id, void *data)`, `int mq_recv (id, void **data)` koje rade s kazaljkama (u red stavljaju i vraćaju kazaljku, ne kopiraju podatke!).

```
struct packet {size_t size, char data[1]};
static id_r, id_s;
static char input_buffer[128];

void net_init()
{
    id_r = mq_create();
```

```
    id_s = mq_create();
    *((int*)0x72F4180) = input_buffer;//kamo da sprema dolazeći paket
}
void net_send(struct packet *packet)
{
    if (*((int*)0x72F4188) == 0) {
        *((int*)0x72F418C)  = packet->size;
        *((int*)0x72F4190)  = packet->data;
    }

    //u ovom rješenju uvijek se "zapamti" paket, čak i ako je stavljen
    //za slanje; na prekid, kad je paket poslan, onda se miče i briše
    mq_send(id_s, packet);
}
int net_recv(struct packet **packet)
{
    return mq_recv(id_r, &packet);
}
void net_interrupt()
{
    size_t size = *((int*)0x72F4184);
    if (size) {//ima novog
        struct packet *p = kmalloc(sizeof(struct packet)+size);
        p->size = size;
        memcpy(p->data, input_buffer, size);
        mq_send(id_r, p);
    }
    if (*((int*)0x72F4188) == 0) {//spreman za (novo) slanje
        struct packet *packet;
        if ( mq_recv(id_s, &packet) ) {//prvi u redu je poslan
            kfree(packet); //sad ga obriši
            if ( mq_recv(id_s, &packet) ) { //ima li još?
                *((int*)0x72F418C)  = packet->size;
                *((int*)0x72F4190)  = packet->data;
            }
        }
    }
}
```

5. (3) U programu `p1.c` ostvariti funkciju `sensor_interrupt()` koja će na prekid senzora poslati poruku sadržaja "PING" (korištenjem opisanog sučelja `net_send` u prethodnom zadatku). Program u `p2.c` svakih 5 sekundi treba provjeriti ima li pristiglih poruka i ako ima njihov sadržaj poslati na LCD upisivanjem njena sadržaja na adresu 0x328910 (koristiti ostavarena sučelja za rad s vremenom). Program u `p3.c` u petlji učitava vrijednost s treće naprave, s adrese 0x9871230 te ako je pročitana vrijednost veća od 90 šalje poruku "HOT" preko mreže te na LCD ispisuje "FLUID TOO HOT". Program u `p2.c` se pokreće samoj jednom (jer se njegova funkcionalnost stavlja u alarm), dok program u `p3.c` radi cijelo vrijeme (tj. prekida se samo prekidima mreže, senzora, brojila).

`p1.c`:

```
void sensor_interrupt()
{
    struct packet *p = kmalloc(sizeof(struct packet)+4);
    p->size = 4;
```

```
    memcpy(p->data, "PING", 4);
    net_send(p);
}
p2.c:
void init_alarm()
{
    struct timespec t;
    t.tv_sec = 5;
    t.tv_nsec = 0;
    time_alarm(&t, alarm);
}
void alarm()
{
    struct packet *p;
    if (net_recv(&p)) {
        size_t i;
        for (i = 0; i < p->size; i++)
            *((int*)0x328910)   = packet->data[i];
            //ovdje ne može memcpy jer on mijenja obje adrese!
    }
}
p3.c:
void p3()
{
    while(1) {
        if (*((int*)0x9871230) > 90) {
            struct packet *p = kmalloc(sizeof(struct packet)+3);
            p->size = 3;
            memcpy(p->data, "HOT", 3);
            net_send(p);

            size_t i;
            char msg[] = "FLUID TOO HOT";
            for (i = 0; i < 13; i++)
                *((int*)0x328910)   = msg[i]; //ne memcpy!
        }
    }
}
```

6. (3) U nekom sustavu koji koristi virtualni spremnik (procesi se nalaze u logičkim adresama) nalazi se jezgrina funkcija `int ktest_strings(char **array)`. Zadatak je funkcije zasebno provjeravati nizove znakova (string) iz polja nizova `array` već postojećom funkcijom `int ktest_string(char *string)`. Zadnji element polja ima vrijednost `NULL` (veličina polja nije drukčije zadana). Kazaljka `array` kao i njeni elementi su u logičkim adresama. Pretpostaviti da postoje funkcije `void *adr_u2k(void *logical_address)` i `void *adr_k2u(void *physical_address)` koje pretvaraju logičku u fizičku adresu i obratno (za trenutno aktivan proces). Funkcija `int ktest_strings` treba vratiti 0 ako su sve povratne vrijednosti od `ktest_string` također bile nule. U protivnom potrebno je vratiti indeks prvog elementa iz polja `array` za koji nije dobivena nula.

```
int ktest_strings(char **array)
{
    char **ka = adr_u2k(array); //u fizičku - adresa početka polja s
```

```
                                    //kazaljkama
    char *string;
    int i;

    for (i = 0; ka[i] != NULL; i++) {
        string = adr_u2k(ka[i]); //svaku kazaljku opet u fizičku adr.
        if (ktest_string(string) != 0)
            return i;
    }
    return 0;
}
```