Završni ispit iz predmeta *Operacijski sustavi za ugrađena računala* 27. 6. 2022.

1. Izvorni kod jezgre za neki ugrađeni sustav nalazi se u direktoriju `kernel`, dok se programi nalaze u direktoriju `programs` (u direktoriju `kernel` se nalaze datoteke `io.c`, `core.c`, `sched.c`, `time.c`; u `programs`: `prog1.c`, `prog2.c`, `prog3.c`).

   a. (3) Napraviti skriptu za povezivanje `ldscript.ld` tako da se pripremi slika sustava koja će se posebnim alatom učitati u ROM koji je na adresi `0x10000`. Sliku pripremiti tako da se sve prvotno učita u ROM, ali da je za ispravan rad potrebno kopirati instrukcije jezgre, podatke jezgre uključujući konstante te podatke programa (bez konstanti) u RAM na adresi `0x50000`. Instrukcije programa i konstante koje program koristi trebaju se koristiti iz ROM-a (pripremiti za rad s tih adresa). Početna funkcija je `startup` (dodati `ENTRY(startup)` u skriptu).

ldscript.ld
```
ENTRY(startup)
ROM = 0x10000;
RAM = 0x100000;

SECTIONS{
     .rom_ram RAM : AT(ROM) {
          kernel* (*)
          programs* (.data .bss)
     }

     rom_ram_end = ROM + SIZEOF(.rom_ram);
     .rom_rom rom_ram_end : AT(rom_ram_end) {
          programs* (.text .rodata)
          //ili *(*) ALI NE MOŽE OVAKO NEŠTO GORE U PRVI ODJELJAK!
     }
     rom_rom_end = rom_ram_end + SIZEOF(.rom_rom);
}
```

   b. (2) Napisati `Makefile` za izgradnju slike sustava `slika.elf`. Mogu se koristiti i implicitna pravila te po potrebi i „recepti“. Pri povezivanju koristiti skriptu za povezivanje (dodati `-T ldscript.ld`). Za samo prevođenje pretpostaviti da su implicitne varijable već postavljene u datoteci `config.ini` koji treba uključiti na početku s `include config.ini`.

```
Makefile
include config.ini
IMG = slika.elf

OBJS = kernel/io.o kernel/core.o kernel/sched.o kernel/time.o \
     programs/prog1.o programs/prog2.o programs/prog3.o

$(IMG): $(OBJS)
     $(LD) -o $(IMG) $(OBJS) $(LDFLAGS) -T ldscript.ld $(LDLIBS)

#implicitna pravila su dovoljna za ostalo, ali može se npr. receptima:
#%.o: %.c
#    $(CC) -c $< $(CFLAGS)
```

2. (4) Neki procesor ima integriran sklop za prihvat prekida. On se programira na način da se u 32-bitovni registar `INT_EN` postavi broj čije jedinice predstavljaju omogućene prekidne ulaze (one koji se prihvaćaju mehanizmom prekida). Ulazi su numerirani od 0 do 31, gdje veći broj označava veći prioritet naprave. Kada naprava sklopu postavi zahtjev za prekid upiše se jedinica na odgovarajuće mjesto u registar `INT_RQ` (npr. naprava spojena za 7. ulaz postaviti će 7. bit). Po obradi prekida, potrebno je obrisati taj bit. Ako je prihvat prekida te naprave dozvoljen, zahtjev će se proslijediti procesoru, odmah, ako je ovaj zahtjev najvećeg prioriteta, ili kasnije, kad se zahtjev većeg prioriteta obradi. Procesor mora u registar `INT_CP` postaviti prioritet trenutne obrade, tj. odgovarajući bit tog registra treba biti postavljen u 1 (uspoređuje se samo najviše postavljeni bit). Procesor prihvaća prekid (automatsko ponašanje procesora pri prihvatu) tako da: 1. sprema sve registre procesora na stog te 2. u PC stavi adresu 0x10000 na koju treba postaviti prekidni program. Napisati prekidni program koji treba tamo postaviti uz pretpostavku da je sustav inicijaliziran, da se neki prekidi prihvaćaju a neki ne (što je programirano u registru `INT_EN`). Zahtjeve naprava koji ne izazivaju prekide, obraditi tek nakon obrade svih naprava koje izazivaju prekide (na kraju tih obrada provjeriti jesu li ostali samo ti zahtjevi i onda ih obraditi proizvoljnim redoslijedom). Pretpostaviti da su funkcije za obradu već postavljene u polje `hndl[]`. Dohvat najznačajnijeg postavljenog bita obaviti funkcijom (instrukcijom) `msb(x)`. Pomoćne operacije: postavi i-ti bit u 1: `X|=1<<i`; obrisi i-ti bit: `X&=^(1<<i)`.

```
prekidni_potprogram //na adresi 0x10000
{
   rq = msb(INT_RQ & INT_CP); //najprioritetniji zahtjev koji izaziva prekid
   INT_CP |= 1 << rq;         //postavi prioritet obrade u CP
   //“dozvoli_prekidanje“ nije potrebno jer ga procesor niti ne zabranjuje!
   hndl[rq]();
   INT_CP &= ^(1 << rq); //više se ne obrađuje ovaj prioritet
   INT_RQ &= ^(1 << rq); //zahtjev je obrađen, više ne čeka

   while (INT_RQ && (INT_RQ & INT_EN == 0)) {
        //ima zahtjeva koji čekaju, ali niti jedan od njih ne izaziva prekid
         rq = msb(INT_RQ); //proizvoljno, ovdje je uzet najvećeg prioriteta
         hndl[rq]();
         INT_RQ &= ^(1 << rq); //prekid obrađen
   }
}
```

3. (4) Neko ugrađeno računalo ima dva brojila: prvo BR1 odbrojava od 0xFFFFF do nule frekvencijom od 1 kHz, a drugo BR2 broji od 0 do 999 frekvencijom od 1 MHz. Prvo brojilo, kad dođe do nule izaziva prekid, dok drugo kad dođe do kraja ponovno kreće od nule (ne izaziva prekid). Oba brojila se mogu čitati i pisati. Ostvariti podsustav za upravljanje vremenom koji će imati sat u preciznosti 1 μs te jedan alarm u granulaciji 1 ms: `inicijaliziraj()`, `dohvati_sat()`, `postavi_sat(novi_sat)`, `postavi_alarm(kada, obrada)`, `prekid_BR1()`. Funkcije za dohvat i postavljanje sata koriste vrijeme u mikrosekundama (npr. 1234567890 μs, dok `kada` predstavlja apsolutno vrijeme, izraženo u milisekundama, npr. 1234567 ms).

```
sat = 0 //u mikrosekundama
obrada = NULL
kada = 0 //u milisekundama
MAX_BR1 = 0xFFFFF
učitano = MAX_BR1

inicijaliziraj () {
   sat = 0
   obrada = NULL
   kada = 0
   učitano = MAX_BR1
   BR1 = učitano
   BR2 = 0
}
```

```
postavi_sat(novi_sat) {
   sat = novi_sat
   kada = 0 //obriši alarm
   učitano = MAX_BR1
   BR1 = učitano
   BR2 = 0
}

dohvati_sat() {
   t = sat
   t += (učitano - BR1) * 1000 + BR2
   vrati t
   //ne ovdje mijenjati sat!!!
}
```

```
postavi_alarm(kada2, obrada2){
   kada = kada2
   obrada = obrada2
   sat += (učitano - BR1) * 1000 + BR2
   učitano = min(MAX_BR1, kada - sat/1000)
   BR1 = učitano
   BR2 = 0
}

prekid_BR1 {
   sat += učitano * 1000 //BR2 == 0 u tom trenutku!
   ako je (kada > 0) {
         ako je sat >= kada {
               kada = 0
               učitano = MAX_BR1
               BR1 = učitano
               obrada()
         }
         inače {
               učitano = min(MAX_BR1, kada - sat/1000)
               BR1 = učitano
         }
   }
}
```

4. (4) Neko ugrađeno računalo ima sporu bežičnu komunikaciju izvedenu posebnim sklopom koji za svaki prihvat 32-bitovna podatka generira prekid. Procesor tada mora prebaciti primljeni podatak iz registra sklopa na adresi `0xFFF30`, a da bi sklop mogao primiti sljedeći podatak. Pri slanju, podatak treba upisati na adresu `0xFFF40`. Ima li nepročitani novi podatak provjerava se preko statusnog registra na adresi `0xFFF50`, preko bita 0 (ako je postavljen). Slično, može li se poslati novi znak vidi se preko bita 1 (ako je postavljen). Odgovarajući bitovi statusna registra se automatski brišu nakon čitanja/slanja podataka. Sklop će generirati prekid i kad pošalje podatak, i kad može primiti sljedeći (na prekid treba provjeriti je li došao novi i/ili poslan idući). Podaci (na „višoj razini“) se šalju u paketima veličine do deset 32-bitovnih podataka omeđeni početnom i završnom vrijednošću (`0xF1F2F3F4` i `0x1F2F3F4F`) koji se računaju u veličinu paketa. Ostvariti upravljački program naprave, tj. ostvariti sučelja `init`, `send`, `recv`, `interrupt`. Funkcije `send` i `recv` se koriste za čitanje i slanje potpune informacije koju upravljački program treba zaštititi početnim i završnim brojem. Koristiti međuspremnike. U slučaju da prethodna poruka nije još poslana, idući `send` mora vratiti grešku (-1). Također, ako cijela poruka nije primljena, `recv` vraća -1. Odbaciti primljene podatke ako nisu započeli početnom vrijednošću. Slično ako prethodno primljen paket nije još pročitan s `recv`.

```
IN = 0xFFF30
OUT = 0xFFF40
STATUS = 0xFFF50
START = 0xF1F2F3F4
END = 0x1F2F3F4F
buf_in[10]          //10 brojeva (32-bitovnih)
buf_out[10]         //10 brojeva (32-bitovnih)
size_in = 0         //koliko je brojeva primljeno
size_out = 0        //koliko je brojeva poslano
size_to_send = 0    //koliko ukupno brojeva treba poslati

init() {
   *STATUS = 0
   size_in = 0
   size_out = 0
   size_to_send = 0
}
send(data, size) {
   ako je (size_to_send > 0)
         vrati -1 //prethodno nije još poslano

   buf_out[0] = START
   kopiraj(&buf_out[1], data, size)
   buf_out[size] = END
   size_to_send = 1 + size + 1
   size_out = 0

   ako je (*STATUS & 2 != 0) {
         *OUT = buf_out[size_out]
         size_out++
   }
}
```

```
recv(data, size) {
   ako je (size_in < 3 ILI buf_in[size_in - 1] != END)
         vrati -1

   size = size_in - 2
   kopiraj(data, &buf_in[1], size)
   size_in = 0
}

interrupt() {
   ako je (*STATUS & 2 != 0 I size_to_send > size_out) {
         *OUT = buf_out[size_out]
         size_out++
   }
   ako je (*STATUS & 1 != 0) {
         msg = *IN
         ako je (size_in < 10) {
               ako je ((size_in > 0 I buf_in[size_in] != END)
                        ILI (size_in == 0 I msg == START)) {
                     buf_in[size_in] = msg
                     size_in++
                     ako je (size_in == 10 I buf_in[9] != END)
                           size_in = 0 //odbaci podatke
               }
               //inače ignoriraj podatak
         }
   }
}
```

5. (4) Osmisliti i ostvariti raspoređivanje dretvi u jednostavnom sustavu koji koristi prioritetno raspoređivanje s 4 različita prioriteta (0-3). Definirati opisnik dretve i osmisliti strukturu podataka potrebnu za raspoređivanje (red pripravnih). Ostvariti funkcije za inicijalizaciju raspoređivača `void sched_init()`, za dohvat najprioritetnije dretve iz reda pripravnih `kthread_t *sched_get()`, za stavljanje dretve u red pripravnih `void sched_add(kthread *)`, za dohvat i micanje najprioritetnije dretve iz reda pripravnih `kthread_t *sched_pop()` te za odabir aktivne dretve (među trenutno aktivnom, ako takva postoji, i pripravnim dretvama) `void schedule()`. Raspoređivač neka radi prema prioritetu kao primarnom kriteriju te redu prispijeća kao sekundarnom (kao što to radi SCHED_FIFO). Dohvat opisnika trenutno aktivne dretve neka je moguć kroz `kthread_t *get_active()`, a postavljanje aktivne obaviti s `void set_active(kthread_t *)` (obje ove funkcije postoje, ne njih ostvarivati!). U `schedule()` dovoljno je na kraju postaviti aktivnu dretvu, povratak u nju će se ostvariti povratkom iz jezgrine funkcije – povratkom iz prekida (zato nije potrebno ništa drugo).

```
typedef struct kthread {
   int prio;
   struct kthread *next;
} kthread_t;

typedef struct rq {
```

```
   kthread_t *first;
   kthread_t *last;
} rq_t;

#define PRIO 4
rq_t rq[PRIO];

void sched_init() {
   int i;
   for (i = 0; i < PRIO; i++)
         rq[i].first = rq[i].last = NULL;
}
kthread_t *sched_get() {
   int i;
   for (i = PRIO-1; i >= 0; i++)
         if (rq[i].first)
               return rq[i].first;
   return NULL;
}
kthread_t *sched_pop() {
   int i;
   kthread_t *thread = NULL;
   for (i = PRIO-1; i >= 0; i++)
         if (rq[i].first) {
               thread = rq[i].first;
               rq[i].first = thread->next;
               if (rq[i].first == NULL)
                     rq[i].last = NULL;
               return thread;
         }
   return NULL;
}
void sched_add(kthread_t *thread) {
   if (rq[thread->prio].last == NULL) //nema tu dretvi
         rq[thread->prio].first = thread;
   else
         rq[thread->prio].last->next = thread;
   rq[thread->prio].last = thread;
   thread->next = NULL;
}
void schedule(){
   kthread_t *act = get_active();
   kthread_t *next = sched_get();
   if (!act || act->prio < next->prio) {
         next = sched_pop();
         set_active(next);
   }
}
```

6. (1) U sustavu koji koristi procese, tj. logički adresni prostor unutar njih, treba ostvariti jezgrinu funkciju `size_t sys_read(int fd, char *buf, size_t size)`, gdje je argument `buf` u logičkim adresama procesa. Pretpostaviti da se podaci interno u jezgri mogu dohvatiti iz datoteke/naprave funkcijom `kread()` s istim argumentima, ali gdje drugi argument mora biti fizička adresa. Skicirati ostvarenje te funkcije i navesti što je sve još potrebno od jezgre. Izostaviti provjeru argumenata, pretpostaviti da su oni već provjereni, da su prekidi zabranjeni i da će biti dozvoljeni nakon ove funkcije, prije povratka u dretvu (negdje drugdje).

```
size_t sys_read(int fd, char *buf, size_t size) {
   char *b = log_to_phy(buf); //potrebna je ta funkcija
   return kread(fd, b, size);
}
```