Operacijski sustavi za ugrađena računala, pismeni ispit, 8. 7. 2025.

Pisati čitko – nečitka rješenja su kriva. U mnogim zadatcima nisu zadane adrese memorija/registara – koristiti zadana imena kao da su adrese (kao da su prethodno definirane, npr. kroz `#define ADDR ((void *) 0x123123)`*).*
Ispit ima viška bodova – maksimalan broj bodova (35) može se ostvariti i bez da se sve riješi.

1. (6) Sustav ima sklop za prihvat prekida s nekoliko upravljačkih i statusnih registara. Sklop ima 16 ulaza na koje su spojeni zahtjevi različitih naprava. Preko 16-bitovnog registra `PIC_CR` omogućuje se prihvat prekida pojedinog ulaza (prosljeđivanje procesoru), preko 16-bitovnog registra `PIC_SR` može se pogledati koje su naprave tražile prekid, a preko 16-bitovnog registra `PIC_ACK` se napravama javlja da je njihov prekid prihvaćen na što one spuštaju svoj zahtjev. U datotekama `interrupt.c/h` ostvariti prekidni podsustav sa sučeljem:

   ```
   void register_interrupt(int irq, void (*handler)(int irqid));
   ```

   Ako je `handler=NULL` onda zabraniti prekidanje s ulaza `irq`. U protivnom ga dozvoliti i povezati s navedenom funkcijom. Procesor prihvaća prekid sklopa tako da u programsko brojilo postavlja adresu 0x50000 – prekidni potprogram treba postaviti na tu adresu (u kodu pripremiti sve što treba, a u zadnjem zadatku to ugraditi u sve potrebno). U prekidnom potprogramu treba ustanoviti izvor prekida i pozvati odgovarajuću registriranu funkciju. Sklop za prihvat prekida je pri pokretanju već u stanju zabrane svih prekida (u `PIC_CR` su sve nule).

2. (6) Sustav ima jednostavan sklop za upravljanje s osam ulazno-izlaznih pinova opće namjene. Upravljanje smjerovima pinova obavlja se preko 8-bitovnog registra `IO_CR1`: nula na pojedinoj poziciji označava da je taj bit ulazni, jedinica izlazni. Čitanje stanja ulaznih pinova obavlja se preko registra `IO_DR_IN` – čitaju se samo stanja pinova označenih kao ulazni, vrijednosti za izlazne pinove ne interpretirati (jer ne predstavljaju ništa). Npr. ako su pinovi 0-4 i 7 označeni kao ulazni i čitanjem s `IO_DR_IN` se dobila vrijednost 0x43. To interpretirati: ulaz0=1, ulaz1=1, ulaz2=0, ulaz3=0, ulaz7=1. Postavljanje izlaznih pinova obavlja se preko registra `IO_DR_OUT` s istom logikom kao i za ulaze – postavljaju se samo oni pinovi označeni kao izlazni. Registrom `IO_CR2` definira se za svaki ulazni pin da li promjena stanja izaziva prekid (na ulazu `IO_IRQ` sklopu za prihvat prekida): 1 označava generiranje prekida, 0 ne. Ostvariti upravljanje pinovima u `io.c/h` preko sučelja:

   ```
   void io_set_direction(int pin, int dir);
   void io_set_irq(int pin, int request);
   int io_get(int pin);  //vraća 0 ili 1, ovisno o stanju pina
   void io_register_interrupt(void (*handler)(int irqid));//koristiti sučelje  iz prethodnog zadatka
   ```

3. (6) Ostvariti upravljanje vremenom u jedinici milisekunde u `time.c/h` preko sučelja:

   ```
   void time_init(long time_now); //inicijalizira podsustav, postavlja vrijeme, briše postojeći alarm
   void time_alarm(long delay, void (*alarm)()); //postavlja alarm
   long time_get();//dohvaća vrijeme
   static void time_interrupt_handler();//interna funkcija koju ostvariti i povezati s prekidom brojila
   ```

   Sustav ima 16-bitovno brojilo na adresi `CNT` koje odbrojava s 50 KHz od učitane vrijednosti do nule, kada šalje zahtjev za prekid na ulaz `CNT_IRQ`, u brojilo učitava zadnju poslanu vrijednost i nastavlja s brojanjem.

4. (6) Ostvariti jednostavno upravljanje dinamičkom memorijom u `mm.c/h` kroz sučelja:

   ```
   void mm_init(void *start, size_t size); //inicijalizira podsustav
   void *mm_alloc(size_t size); //alocira memoriju ili vraća NULL ako nema memorije
   void mm_free(void *mem); //oslobađa alociranu memoriju
   ```

   Pretpostaviti da ostvarenje liste postoji (u `lib.c/h`, ne implementirati!) sa sučeljima:

   ```
   void list_init(list_t *header); //inicijalizira zaglavlje liste
   void list_add(list_t *header, chunk_t *chunk); //dodaje u listu složenu prema adresama
   void *list_next(list_t *chunk);  //vraća slijedeći element ili NULL ako je ovo zadnji
   void *list_prev(list_t *chunk);  //vraća prethodni element ili NULL ako je ovo prvi
   void list_del(list_t *chunk);  //miče zadani element iz liste
   ```

   Struktura `list_t` nije poznata niti je to potrebno – definirati svoju strukturu za blokove memorije (npr. `chunk_t`) i na prvo mjesto postaviti `list_t` element – tada je adresa `list_t` elementa i `chunk_t` jednaka, u prethodne funkcije može slati ista uz `(list_t*)` *cast* i obratno.

   Pri ostvarenju podijeliti slobodni blok ako bi ostatak bio barem 128 B. Pri oslobađanju spajati susjedne slobodne blokove. Mogu svi blokovi (slobodni i zauzeti) biti u zajedničkoj listi.

5. Programska potpora za neki sustav nalazi se u datotekama: `boot.c`, `io.c`, `mm.c`, `interrupts.c`, `time.c`, `threads.c`, `startup.c`, `lib.c`, `programs.c` te istoimenim zaglavljima – `.h` datotekama.

   a. (6) Napisati skriptu za povezivanje `ldscript.ld` koja će sve pripremiti za učitavanje u ROM na adresi 0x10000 i rad iz RAM-a na adresi 0x50000. Sadržaja iz `boot.o` pripremiti za učitavanje i korištenje iz ROM-a. Sve ostalo treba pri pokretanju premjestiti u RAM. Na adresu 0x50000 staviti odjeljak `.interrupt_handler` iz datoteke `interrupts.o` (u koji treba staviti samo funkciju za prihvat prekida u 1. zadatku). Iza toga sve ostalo.

      Napisati kod datoteke `boot.c` koja radi to kopiranje i po završetku poziva funkciju `startup()` ostvarenu u `startup.c`.

   b. (6) Napisati datoteku `Makefile` kojom se izgrađuje navedeni sustav. Pretpostaviti da se pri pokretanju prevođenja u naredbenoj liniju zadaju dvije varijable prema: `make VERSION=3 DEBUG=1` te da ih treba u odgovarajućem obliku predati programima pri prevođenju (koriste se u izvornom kodu). Neka prevoditelj koji se ovdje koristi, imena `compajler`, već pri pokretanju sam postavlja početne vrijednosti u varijable `CFLAGS`, kao i povezivač, imena `linker`, sam postavlja početne vrijednosti za `LDFLAGS` i `LDLIBS`.

   c. (6) Napisati datoteke `startup.c/h` i `programs.c/h`.
      U `startup.c` inicijalizirati sustav (sve što je potrebno iz prethodnih zadataka) te pozvati funkciju `blink()`. Pretpostaviti da sustav ima 1 MB memorije (zadnja adresa RAM-a je 0xFFFFF).
      U `programs.c` ostvariti funkciju `blink()` koja svake sekunde mijenja stanja pinova 0-5. Također, u toj funkciji postaviti da svaka 10-ta promjena na pinu 6 (koji je ulazni pin) uzrokuje promjenu izlaza pina 7 kojeg početno postaviti u nulu. Koristiti sučelja iz prethodnih zadataka.

6. (6) Ostvariti višedretveni podsustav u `thread.c/h` s vanjskim sučeljima:

   ```
   void thread_create(void *thread_function, void *param);
   void thread_exit(void *thread_function, void *param);
   void thread_yield();  //staviti ju na kraj reda pripravnih
   void enter_critical_section();  //samo je jedan KO u sustavu
   void leave_critical_section();
   ```

   Neka se dretve raspoređuju po redu prispijeća i neka se zamjena jedne dretve drugom može obaviti funkcijom `switch_context(context_t *from, context_t*to)` koja postoji (i `context_t` je već definiran). Sva ostala potrebna sučelja uzeti iz prethodnih zadataka.

1.

interrupt.h

```
void register_interrupt(int irq_id, void (*handler)(int irqid));
```

interrupt.c

```
#include "interrupt.h"

static void (*ih[16])(int) = {NULL};
static int *cr = PIC_CR;
static int *sr = PIC_SR;
static int *ack = PIC_ACK;

void register_interrupt(int irq, void (*handler)(int))
{
	ih[irq] = handler;

	if (handler != NULL)
		*cr |= 1 << irq;
	else
		*cr &= ~(1 << irq);
}

static void intterrupt_handler() __attribute__((section(".interrupt_handler")))
{
	int i;
	for (i = 0; i < 16; i++) {
		if ( (*sr & (1 << i)) != 0) {
			*ack = 1 << i;
			ih[i](i);
		}
	}
}
```

2.

io.h

```
void io_set_direction(int pin, int dir);
void io_set_irq(int pin, int request);
int io_get(int pin);
void io_register_interrupt(void (*handler)(int irqid));
```

io.c

```
#include "io.h"

void io_set_direction(int pin, int dir)
{
	int *x = IO_CR1;

	if (dir)
		*x |= 1 << pin;
	else
		*x &= ~(1 << pin);
}

void io_set_irq(int pin, int request)
{
	int *x = IO_CR2;

	if (dir)
		*x |= 1 << pin;
	else
```

```
            *x &= ~(1 << pin);
}

int io_get(int pin)
{
      int *x = IO_DR_IN;

      return (*x & (1 << pin)) >> pin; //ili (*x >> pin) & 1;
}

void io_register_interrupt(void (*handler)(int irqid))
{
      register_interrupt(IO_IRQ, handler);
}
```

3.
time.h

```
void time_init(long time_now);
void time_alarm(long delay, void (*alarm)());
long time_get();
```

time.c

```
#include "time.h"

//2^16-1 = 0xFFFF = 65535; obzirom na f=50000 može se kao max. vr. staviti i 50000,
//jednostavniji je proračun, ali može i max.vr.
#define MAX 50000
static long t = 0; //sat
static long delay = 0; //odgoda do alarma, u otkucajima sata
static void (*handler)() = NULL;
static long load = MAX;
static *counter = CNT;

void time_init(long time_now)
{
      t = time_now;
      delay = 0;
      load = MAX;
      *counter = load;
      register_interrupt(CNT_IRQ, time_interrupt_handler);
}

void time_alarm(long _delay, void (*alarm)()) //delay u ms
{
      delay = _delay * 50;
      handler = alarm;

      t += (load - *counter) / 50;
      load = min(MAX, delay);
      *counter = load;
}

long time_get()
{
      return t + (load - *counter) / 50;
}

static void time_interrupt_handler()
{
      t += load / 50;
```

```
	if (delay > 0) {
		delay -= load;
		if (delay <= 0) {
			delay = 0;
			*counter = load = MAX;
			 alarm_handler();
		}
		else {
			load = min(MAX, delay);
			*counter = load;
		}
	}
}
```

4.

mm.h

```
#include <lib.h>
void mm_init(void *start, size_t size);
void *mm_alloc(size_t size);
void mm_free(void *mem);
```

mm.c

```
#include "mm.h"

typedef struct _chunk_t_ {
	list_t list;
	size_t size; //negativne vrijednosti za zauzete blokove
}
chunk_t;

static list_t list;

//u sve bi ove funkcije trebalo dodati i zaključavanje, npr. prema sučelju iz zadnjeg zadatka:
//enter_critical_section() i void leave_critical_section(); ali je to ipak ovdje izostavljeno

void mm_init(void *start, size_t size) {
	chunk_t *chunk = start;//pretpostavka je da ne treba poravnanje
	chunk->size = size;

	list_init(&list);
	list_add(&list, (list_t *) chunk);
}

void *mm_alloc(size_t size)
{
	size += sizeof(chunk_t);//treba i zaglavlje uračunati

	list_t *item = list_next(&list);
	while (item != NULL) {
		chunk_t *chunk = (chunk_t *) item;
		if (chunk->size >= size) {
			if (chunk->size >= size + 128) {
				//podijeli ga
				chunk_t *c = (void *) chunk + size;
				c->size = chunk->size - size;
				list_add(&list, (list_t *) c);
				chunk->size = -size; //oznaka zauzetosti
				return (void *) (chunk + 1);
			}
```

```
            }
            item = list_next(&item);
      }
      return NULL;
}

void mm_free(void *mem)
{
      chunk_t *chunk = ((chunk_t *) mem) - 1;
      chunk->size = -chunk->size;

      list_t *item = list_prev((list_t *) chunk);
      if (item != NULL) {
            chunk_t *c = (chunk_t *) item;
            if(c->size > 0) {
                  c->size += chunk->size;
                  list_del(chunk);
                  chunk = c;
            }
      }
      item = list_next((list_t *) chunk);
      if (item != NULL) {
            chunk_t *c = (chunk_t *) item;
            if(c->size > 0) {
                  chunk->size += c->size;
                  list_del(c);
            }
      }
}
```

5. a.

ldscript.ld

```
SECTIONS {
      .boot 0x10000 : AT(0x10000) {
            boot.o (*)
      }

      pocetak = 0x10000 + SIZEOF(.boot);

      .ostalo 0x50000 : AT (pocetak) {
            interrupts.o (.interrupt_handler)
            * (*)
      }
      kraj = pocetak + SIZEOF(.ostalo);
}
```

boot.c

```
void boot() {
      extern char pocetak, kraj;
      char *a = &pocetak, *b = &kraj;
      char *c = (char *) 0x50000;
      while (a < b)
            *c++ = *a++;
      startup();
}
```

b. Makefile

```
CFLAGS += -D VERSION=3 -D DEBUG=1
```

```
OBJS = boot.o io.o mm.o interrupts.o time.o threads.o stratup.o lib.o programs.o

CC = compajler
LD = linker

IMG = image.elf

$(IMG): $(OBJS)
	(LD) $(LDFLAGS) $(OBJS) $(LDLIBS) -o $(IMG)
```

c.
startup.h

```
void startup();
```

startup.c

```
void startup() {
	extern char kraj;
	time_init(0);
	mem_init(&kraj, 0xFFFFF - &kraj);
	blink();
}
```

programs.h

```
void blink();
```

programs.c

```
static int pin7_change = 0, pin7_state = 0;
static int pins_state = 0;

static void pin6_interrupt(int irq)
{
	pin7_change = (pin7_change + 1) % 10;
	if (pin7_change == 0) {
		pin7_state = 1 - pin7_state;
		io_set_pin(7, pin7_state);
	}
}
static void change_pins()
{
	time_alarm(1000, change_pins);

	pins_state = 1 - pins_state;
	int i;
	for (i = 0; i < 6; i++)
		io_set_pin(i, pins_state);
}

void blink()
{
	int i;
	for (i = 0; i <= 5; i++) {
		io_set_direction(i, 1);
		io_set_pin(i, pins_state);
	}

	io_set_direction(6, 0);
	io_register_interrupt(pin6_interrupt);
	io_set_irq(6, 1);

	io_set_direction(7, 1);
	io_set_pin(7, pin7_state);
```

```
	time_alarm(1000, change_pins);

	for(;;);
}
```

6. thread.h

```
void thread_create(void *thread_function, void *param);
void thread_exit();
void thread_yield();
void enter_critical_section();
void leave_critical_section();
```

thread.c

```
#include "thread.h"

typedef struct _kthread_t_ {
	list_t list; //na prvom mjestu da se pojednostavi korištenje liste
	context_t context;
}
kthread_t;

static kthread_t *active;
static list_t ready;
static list_t cs;
static int cs_state = 1;

//sve bi ove funkcije trebale imati na početku zabrani_prekide() i dozvoli na kraju
//zabrana i dozvola preko statusnog registra procesora

void thread_create(void *thread_function, void *param)
{
	zabrani_prekide();
	kthread_t *thread = mm_alloc(sizeof(kthread_t));
	list_add(&ready, thread);
	dozvoli_prekide();
}

void thread_exit()
{
	zabrani_prekide();
	//aktivna ovo poziva
	kthread_t *old = active;
	active = list_next(&ready);
	list_del(&ready, active);
	mm_free(old);
	dozvoli_prekide();
	switch_context(NULL, active->context);
}

void thread_yield()
{
	zabrani_prekide();
	kthread_t *old = active;
	list_add(&ready, active);
	active = list_next(&ready);
	list_del(&ready, active);
	dozvoli_prekide();
	switch_context(old->context, active->context);
```

```
}

void enter_critical_section()
{
	zabrani_prekide();
	if (cs_state == 1) {
		cs_state = 0;
		dozvoli_prekide();
	}
	else {
		cs_state--;
		kthread_t *old = active;
		list_add(&cs, active);
		active = list_next(&ready);
		list_del(active);
		dozvoli_prekide();
		switch_context(old->context, active->context);
	}
}

void leave_critical_section()
{
	zabrani_prekide();
	if (cs_state == 0) { //nitko ne čeka
		cs_state = 1;
	}
	else {
		kthread_t *waiting = list_next(&cs);
		list_del(&cs, waiting);
		list_add(&ready, waiting);
	}
	dozvoli_prekide();
}
```