Pismeni ispit iz predmeta *Operacijski sustavi za ugrađena računala*, 9.6.'10.

1. Iz navedene skripte povezivača navesti:

```
SECTIONS {
        .pocetak 0x100000 : {
                code = .;
                *(.pocetak_koda)
                . = ALIGN(4096);
        }
        .kod : {
                *(.text)
                . = ALIGN(4096);
        }
        .podaci : {
                data = .;
                *(.data)
                *(.rodata.*)
                . = ALIGN(4096);
        }
        .din_spr : {
                bss = .;
                *(.bss)
                . = ALIGN(4096);
        }
        end = .;
}
```

   a) (1 bod) imena ulaznih odjeljaka:

```
        .pocetak_koda, .text, .data,
.rodata, .bss
```

   b) (1 bod) imena izlaznih odjeljaka:

```
    .pocetak, .kod, .podaci, .din_spr
```

   c) (1 bod) sve varijable definirane u skripti

```
        code, data, bss, end
```

   d) (2 boda) vrijednosti svih varijabli (po potrebi koristiti odgovarajuće makroe i prethodno izračunate varijable)

```
        code = 0x100000,
        data = code + SIZEOF(.pocetak) + SIZEOF(.kod)
        bss = data + SIZEOF(.podaci)
        end = bss + SIZEOF(.din_spr)
```

2. (3 boda) U nekom kodu (.c) nalazi se poziv funkcije:

```
a = zbroji ( m, n, p );
```

   Funkcija zbroji definirana je kao:

```
int zbroji ( int m, int n, int p ) {
      int suma;
      suma = m + n + p;
      return suma;
}
```

   Skicirajte instrukcije koje će prevoditelj napraviti prevođenjem zadanog koda za poziv funkcije, te za samu funkciju (uz pretpostavku da se optimizacije neće koristiti). Koristiti instrukcije procesora ARM ili x86 (ili slične).

```
// a = zbroji ( m, n, p );
push p
push n
push m
call zbroji
add $12, %esp
mov %eax, a

ili, npr.
sub $12, %esp
mov m, (%esp)
mov n, 4(%esp)
mov p, 8(%esp)
call zbroji
add $12, %esp
mov %eax, a
```

```
// int zbroji ( int m, int n, int p ) { ...
push %ebp
mov %esp, %ebp

add $4, %esp //suma

mov 12(%ebp), %eax   //eax = m
add 16(%ebp), %eax
add 20(%ebp), %eax
mov %eax, (%esp)

mov (%esp), %eax

leave
ret
```

3. (6 bodova) Jedan projekt za ugrađene sustave sastoji se od tri datoteke: pocetak.c, jezgra.c i programi.c (te njihova zaglavlja, ekvivalentne .h datoteke). Sve se datoteke sastoje od .text, .rodata, .data i .bss odjeljaka. Skicirati skriptu za povezivača (*linkera*) koji će napraviti datoteku slike sustava koja će se učitati po pokretanju sustava u spremnik na adresu 0x100000. Kôd u pocetak.c će odmah po početku izvođenja premjestiti kôd i podatke „jezgre“ koji nastaju iz datoteke jezgra.c na spremničku lokaciju 0x10000. Iza „jezgre“ kopirati će se i „programi“ (svi odjeljci iz datoteke programi.c), ali na prvu iduću adresu poravnatu na veličinu stranice (zadnjih 12 bitova moraju biti nule). Isto tako "poravnati" i kraj "programa".
   „Jezgru“ (odjeljak `.jezgra`) treba pripremiti za fizičku adresu gdje će se ona nalaziti, tj. adresu 0x10000. „Programe“ (odjeljak `.programi`) treba pripremiti za logičku adresu 0x20000!
   Dodati u skriptu varijable `programi_pocetak` i `programi_kraj` koje će sadržavati fizičku adresu početka i kraja dijela spremnika gdje se nalazi odjeljak s „programima“ (nakon premještanja), a koji su potrebni podsustavu za upravljanje spremnikom pri stvaranju novih procesa (taj se kopira za svaki novi proces). Sve se objektne datoteke nalaze u istom direktoriju (neka filter bude ime objektne datoteke). Po potrebi dodati i dodatne varijable u skriptu.

Rješenje:

```
SECTIONS {

	.pocetak {
		pocetak.o (.text)
		pocetak.o (.rodata)
		pocetak.o (.data)
		pocetak.o (.bss)
	}

	pocetak_jezgre = .;
	.jezgra 0x10000 AT ( pocetak_jezgre ) {
		jezgra.o (.text)
		jezgra.o (.rodata)
		jezgra.o (.data)
		jezgra.o (.bss)
		. = ALIGN(4096);
	}

	programi_pocetak = pocetak_jezgre + SIZEOF ( .jezgra );
	.programi 0x20000 AT ( programi_pocetak ) {
		programi.o (.text)
		programi.o (.rodata)
		programi.o (.data)
		programi.o (.bss)
		. = ALIGN(4096);
	}
	programi_kraj = programi_pocetak + SIZEOF ( .programi );

}
```

4. (3 boda) Nadopuniti navedene datoteke odgovarajućim ključnim riječima (`static`, `extern`, ...) i sl. tako da navedeno ima smisla. Sve što ne treba biti „globalno“ označiti lokalnim.
   (3 boda) Napisati odgovarajući „`Makefile`“ (pretpostaviti da nikakve dodatne zastavice i sl. nisu potrebne).

"jezgra.h"
```
int j_dohvati_znak ();
int postavi_alarm (...);
int obrisi_alarm (...);
```

"jezgra.c"
```
#include "jezgra.h"
#include "arh.h"

extern int br_zn;
int j_dohvati_znak () {
	//čekaj dok nema znakova u međuspremniku
	//arh sloja za tipkovnicu
	while ( br_zn == 0 )
		asm ( "hlt" );

	return arh_dohvati_znak ();
}

int postavi_alarm (...) { /* kod */ }
int obrisi_alarm (...) { /* kod */ }
```

"program.h"
```
void jeka ();
```

"program.c"
```
#include "program.h"
#include "jezgra.h"
static
int br_zn = 0;
static
char ms[BR_ZN] = { 0 };
static
char znak = 0;

static
void ispisi ( format, ... ) { /* kod */ }

static
void obrada_alarma () {
	ispisi ( "Stanje: ms=%s, br_zn=%d, zadnji_znak=%c\n", ms, br_zn, znak );
}
void jeka () {

	int id = postavi_alarm ( "svake 3 sekunde pozovi funkciju", obrada_alarma );

	do {
		znak = j_dohvati_znak ();
		ms[br_zn++] = znak;
	}
	while ( znak != 'q' );

	obrisi_alarm ( id );
}
```

"arh.h"
```
int arh_dohvati_znak ();
void arh_prekid_tipk ();
```

"arh.c"
```
int br_zn = 0;
static
int prvi = 0;
static
char ms[VEL_MS];


int arh_dohvati_znak () {
	if ( br_zn == 0 )
		return 0;

	char znak = ms[prvi++];

	if ( prvi > VEL_MS )
		prvi = 0;

	br_zn--;

	return znak;
}
static
int uzmi_znak_iz_tipk (){ /* kod */ }

void arh_prekid_tipk () {
	char znak = uzmi_znak_iz_tipk ();

	prvi++;
	br_zn++;

	if ( prvi > VEL_MS )
		prvi = 0;

	ms[prvi] = znak;
}
```

5. (3 boda) Pri prihvatu prekida (potpuni) kontekst prekinute dretve sprema se u dva koraka (za većinu arhitektura). Zašto su potrebna dva koraka (ili više)?

procesor „sam" pohranjuje samo par registara, ostale treba programski (za potpuni kontekst)

6. (3 boda) Prihvaćanje prekida kojima je uzrok izvan procesora može se privremeno onemogućiti odgovarajućom zastavicom registra stanja. Zašto se prekide generirane unutar procesora ne može onemogućiti (zašto je tako sustav napravljen)?

zato jer ti prekidi ili pokazuju grešku (dijeljenje s nulom, nepostojeća instrukcija, ...) pa trenutnu dretvu treba prekinuti, ili je to programski prekid kojim dretva želi pozvati jezgrinu funkciju
i jedno i drugo je neophodno obraditi prikladnim funkcijama

7. (3 boda) Navedite osnovne funkcije/operacije podsustava za upravljanje prekidima. Koja osnovna sučelja taj podsustav treba imati?

8. (3 boda) Navedite operacije/funkcije/sučelja koja mora pružati podsustav za upravljanje vremenom.

9. (3 boda) Je li u sustavima koji nemaju višedretvenost (višezadaćnost) potreban mehanizam programskog prekida? Obrazložiti. Navesti prednosti i nedostatke takvog mehanizma za navedeni sustav.

10. (2 boda) Funkcija označene sa `inline` će se najvjerojatnije „ugraditi" u kôd od kuda se pozivaju. Takve se funkcije vrlo često cijele navode u datotekama zaglavlja (.h). Međutim, nekad ih je ipak potrebno navesti u datotekama s kôdom. Opišite primjer kada je to neophodno.

kada se u njima koriste globalne varijable definirane u .c datoteci (mogu biti i „static")

11. (2 boda) Navedite osnovne elemente opisnika dretvi.

12. (2 boda) Opišite princip sinkronizacije mehanizmom barijere.

13. (3 boda) Opišite mehanizme operacijskog sustava za upravljanje asinkronim događajima (generiranih izvan procesora).

14. (3 boda) Što su to *callback* funkcije (funkcije s povratnim pozivom)? U prikazanom sustavu, koji se elementi sve mogu nazvati takvim funkcijama i zašto?

15. (3 boda) U sustavu sa straničenjem koristi se sklopovska potpora te funkcije operacijskog sustava. Što radi sklopovlje a što operacijski sustav u upravljanju spremničkim prostorom?

Pismeni ispit iz predmeta *Operacijski sustavi za*

1. (3 boda) Nadopuniti skriptu povezivača prikazanu desno. Odrediti vrijednosti svih varijabli korištenih u skripti.

```
SECTIONS {
        .jezgra 0x100000 : {
                code = .;
                jezgra*(.text)
                data = .;
                jezgra*(.data)
                jezgra*(.rodata.*)
                bss = .;
                jezgra*(.bss)
                . = ALIGN(4096);
        }
        programi = .;

        .program1 0 : AT (programi)
        {
                prog1*(.text)
                prog1*(.rodata.*)
                prog1*(.data)
                prog1*(.bss)
                . = ALIGN(4096);
        }
        .program2 0 :
        AT(programi + SIZEOF(program1))
        {
                prog2*(.text)
                prog2*(.rodata.*)
                prog2*(.data)
                prog2*(.bss)
                . = ALIGN(4096);
        }

        ostatak = programi +
                  SIZEOF(program1) +
                  SIZEOF(program2)
}
```

```
code = 0x100000
data = code + SIZEOF(jezgra*(.text))  ***
bss = data + SIZEOF(jezgra*(.data))   ***
programi = code + SIZEOF(.jezgra)
ostatak = programi + SIZEOF(program1) +
          SIZEOF(program2)
```

2. (3 boda) U nekom kodu (.c) nalazi se funkcija:

```
int zbroji ( int m, int n, int *p ) {
      int suma, paran;
      suma = m + n + p;
      paran = ~(suma & 1);
      *p = paran;
      return suma;
}
```

Prikažite stanje stoga (što se sve nalazi na stogu) dretve koja je pozvala navedenu funkciju u trenutku prije izvođenja zadnje linije koda (`return suma`) (uz pretpostavku da se optimizacije neće koristiti).

```
(vrh stoga)                paran
                           suma
                           stara kazaljka „okvira" stoga (ebp, nije potrebno)
                           povratna adresa (kamo se vraća nakon funkcije zbroji)
                           m
                           n
                           p
(tu je na stogu nešto što ovisi o širem kontekstu - ne da se iz primjera odrediti)
```

3. (3 boda) Što sve spada u (puni) kontekst dretve? Navedite primjer za x86 ili ARM7 sustave.

```
svi registri procesora koja dretva koristi:

za x86: eax, ebx, ..., registri „koprocesora", mmx registri, segmentni registri i ostali koji
se koriste za upravljanje spremnikom (cr0, cr3, cr4, ...) te registar stanja (eflags)

za arm7: r0-r15 + CPSR (reg. stanja)
```

4. (6 bodova) Jedan projekt za ugrađene sustave sastoji se od tri datoteke: `pocetak.c`, `jezgra.c` i `programi.c` (te njihova zaglavlja, ekvivalentne `.h` datoteke). Sve se datoteke sastoje od `.text`, `.rodata` i `.data` odjeljaka. Datoteke je potrebno prevesti i povezati (*kompajlirati i linkati*) tako da se slika sustava može staviti u ROM na adresu 0. Obzirom da je ROM dovoljno velik samo za pohranu slike, kôd u `pocetak.c` (koji treba postaviti na adresu 0, a koji nema `.data` odjeljak) će odmah po početku izvođenja (*reseta*) premjestiti podatke „jezgre“ (`.data` odjeljak koji nastaje iz datoteke `jezgra.c`) na spremničku lokaciju 0x10000 (prva adresa RAM-a). Pri pokretanju programa (koji nastaju iz datoteke `programi.c`) njihovi se podaci (`.data` odjeljak) moraju kopirati u RAM. Za upravljanje spremnikom potrebno je pripremiti odgovarajuće dijelove sustava za predviđene lokacije. Za programe se koristi straničenje, tj. programe treba pripremiti tako da započinju s logičkom adresom nula. Jezgra radi u apsolutnim adresama.

   Pored toga, potrebne su slijedeće adrese:

`j_pod_rom`:	adresa podataka jezgre u ROMu (za kopiranje u RAM)
`j_pod_ram`:	adresa podataka jezgre u RAMu (za kopiranje iz ROMa)
`prog_rom`:	adresa instrukcija programa u ROMu (`.text` i `.rodata` odjeljaka)
`prog_d_rom`:	adresa podataka programa u ROMu (`.data` odjeljak)
`slobodno`:	prva adresa iza podataka jezgre, a koja će se koristiti za straničenje (za podatke programa, u RAM-u).

   Skicirati skriptu za povezivača (*linkera*) koji će napraviti datoteku slike sustava koja zadovoljava navedene zahtjeve. Sve se objektne datoteke nalaze u istom direktoriju (neka filter bude ime objektne datoteke). Po potrebi dodati i dodatne varijable u skriptu.

```
SECTIONS {

        .pocetak 0 : AT (0) {
              pocetak.o (.text .rodata)
        }

        .jezgra_kod : {
              jezgra.o (.text .rodata)
        }

        jezgra_pod_rom = .;
        j_pod_ram = 0x10000;
        .jezgra_pr j_pod_ram : AT ( jezgra_pod_rom ) {
              jezgra.o (.data)
              . = ALIGN(4096);
        }

        prog_rom = jezgra_pod_rom + SIZEOF ( jezgra_pr );
        .prog_kod 0 : AT ( prog_rom ) {
              programi.o (.text .rodata)
        }

        prog_d_rom = prog_rom + SIZEOF( prog_kod );
        .prog_pod 0 + SIZEOF(.prog_kod) : AT ( prog_d_rom ) {
              programi.o (.data)
        }

        slobodno = j_pod_ram + SIZEOF ( .jezgra_pr );

}
```

5. (3 boda) Nadopuniti navedene datoteke odgovarajućim ključnim riječima (`static`, `extern`, ...) i sličnо tako da se navedeno može prevesti i da ima smisla. Sve što ne treba ili ne smije biti „globalno“ označiti lokalnim. Dati i ostale „korisne preporuke“ prevoditelju.
   (2 boda) Napisati odgovarajući „`Makefile`“.

N.h:
```
inline
void dodaj_N ( int broj );
inline
int suma_N ();
```

R.h:
```
inline
void dodaj_R ( double broj );
inline
double suma_R ();
```

C.h:
```
typedef struct _C_ {
   double r, i;
} C;
inline
void dodaj_C ( C broj );
inline
C suma_C ();
```

N.c:
```
static
int suma = 0;

extern double naj;

inline static
void zbroji ( int broj ) {
   suma += broj;
   if ( broj > naj )
      naj = broj;
}
inline static
int provjeri ( int broj ) {
   if ( broj>0 && broj<1000 )
      return 1;
   else
      return 0;
}
inline
void zbroji_N ( int broj ) {
   if ( provjeri ( broj ) )
      zbroji ( broj );
}
inline
int suma_N () {
   return suma;
}
```

R.c:
```
static
double suma = 0;

extern double naj;

inline static
void zbroji ( double broj ) {
   suma += broj;
   if ( broj > naj )
      naj = broj;
}
inline static
int provjeri ( double broj ) {
   if ( broj>0 && broj<1000 )
      return 1;
   else
      return 0;
}
inline
void zbroji_R ( double broj ) {
   if ( provjeri ( broj ) )
      zbroji ( broj );
}
inline
double suma_R () {
   return suma;
}
```

C.c:
```
static
C suma = { 0, 0 };


inline static
void zbroji ( C broj ) {
   suma.r += broj.r;
   suma.i += broj.i;
}
inline static
int provjeri ( C broj ) {
   if(broj.r>0 && broj.r<1000)
      return 1;
   else
      return 0;
}
inline
void zbroji_C (  C broj ) {
   if ( provjeri ( broj ) )
      zbroji ( broj );
}
inline
C suma_C () {
   return suma;
}
```

brojke.c:
```
#include „N.h“ i „R.h“ i „C.h“
double naj = 0;

int main () {
   C broj;
   broj.r = 0;
   broj.i = 100;
   while ( broj.r < broj.i ) {
      zbroji_N ( (int)broj.i );
      zbroji_R ( broj.r );
      zbroji_C ( broj );
      broj.r += 1;
      broj.i -= 1;
   }

   broj = suma_c ();
   printf ("%d:%g:%g+%g:%g\n",
       suma_N (), suma_R (),
       broj.r, broj.i, naj );

   return 0;
}
```

Makefile:
```
brojke: brojke.o N.o R.o C.o
    gcc brojke.o N.o R.o C.o -o brojke

brojke.o: brojke.c N.h R.h C.h
    gcc -c brojke.c

brojke.o: N.c N.h
    gcc -c N.c

brojke.o: R.c R.h
    gcc -c R.c

brojke.o: C.c C.h
    gcc -c C.c
```

6. (5 bodova) Skicirajte ostvarenje nadzornog alarma (*watchdog timer*) korištenjem ili sučelja prikazanog sustava (okvirno) ili sučelja iz „standardnih sustava“. Prikazati korištenje sučelja na primjeru.

```
void istek_periode ( void *param )
{
    int id_dretve = (int) param;
    ispisi ( „istek nadzornog alarma za dretvu %d - dretva se prekida!“,
             id_dretve );
    prekini_dretvu ( id_dretve );
}

int postavi_nadzorni_alarm ( int id_alarma, vrijeme_t perioda, int id_dretve )
{
    return postavi_alarm ( id_alarma, perioda, istek_periode,
                           (void *) id_dretve, NEISTINA);
}

void neka_dretva ( void *param )
{
    ...
    id_na = stvori_nadzorni_alarm ( STVORI_NOVI, perioda, id_dretve () );
    ...
    (u nekoj petlji u kojoj dretva radi i mora raditi periodički poziva:)
    stvori_nadzorni_alarm (id_na, perioda, id_dretve () );
    (produžava se alarm za periodu)
    ...
}
```

7. (5 bodova) Neki sustav posjeduje sklop koji svakih 10ms generira prekid iz kojeg se poziva funkcija *j_prekid_sata*. Skicirati tu funkciju, funkciju *j_trenutno_vrijeme* (koja vraća trenutno vrijeme u preciznosti od 10ms – vrijeme od pokretanja sustava) te dodati potrebnu podatkovnu strukturu tako da se navedene funkcije mogu ostvariti. Vrijeme treba iskazati u milisekundama.

```
static int j_sat; //vrijeme sustava

void j_prekid_sata ()
{
    j_sat += 10;
}

int j_trenutno_vrijeme ()
{
    return j_sat;
}
```

8. (2 boda) Koje su prednosti i nedostaci upravljanja prekidima na niskoj razini (u „arh“ sloju) u odnosu na upravljanje na višoj razini (u „jezgri“, dok je u „arh“ sloju samo osnovno upravljanje kao u prikazanom sustavu)?

prednosti: brže (izravno i u IDT za x86; manje kućanskih poslova)
nedostaci: manja prenosivost (treba više mijenjati pri prijenosu na druge sustave), moguće više koda – veća jezgra (svaki prekid ima svoju pohranu i obnovu konteksta...)

9. (3 boda) Čemu služe programski prekidi? Može li se ista funkcionalnost napraviti i drugim načinima/mehanizmima (bar u posebnim slučajevima)? Kada i kako (ako da)?

za poziv jezgrine funkcije koja se obavlja pod većim privilegijama

sustavi koji nemaju „korisnički“ i „nadgledni“ način rada mogu istu funkcionalnost ostvariti izravnim pozivima funkcija (ne preko „skupih“ prekida)

10. (3 boda) Gdje se sve može koristiti ključna riječ *static*? Opišite značenja svake od uporaba.

- izvan funkcija, tada navedena varijabla je „globalna“ ali samo za navedenu datoteku (simboličko ime nije dostupno za druge .c datoteke, ni sa extern)
- funkcija može biti static – dostupna je (kao simboličko ime) samo u navedenoj datoteci
- varijabla u funkciji može biti static – ta se varijabla ponaša kao globalna – njena vrijednost se ne alocira na stogu kao ostale lokalne varijable, već statički je za nju rezervirano mjesto, ali za razliku od globalne njoj se može pristupiti samo iz te funkcije (slični je princip u objektnim jezicima s statičkim varijablama)

11. (2 boda) Uobičajene funkcije sinkronizacije uključuju neke inačice *čekaj* i *postavi*. Koja proširenja tih funkcija susrećemo u „standardnim“ sučeljima? Kratko opišite funkcionalnosti tih proširenja.

probaj_čekati - ne blokira dretvu ako ne može „proći normalno“ (npr. semafor nije prolazan)
čekaj_ograničeno - ako se dretva blokira, trajanje blokiranog stanja je ograničeno na vrijeme zadano kao parametar funkcije

12. (2 boda) Što su to signali (na razini OS-a, ne na električnoj razini)? Kako dretva može definirati svoje ponašanje po primitku signala (npr. u UNIX-u i sličnim sustavima)?

signali – sredstvo dojave asinkronih događaja koji se tiču dretve; sredstvo komunikacije između dretvi te između OS-a i dretvi
ponašanje za signal:
ignoriraj signal
obradi signal pretpostavljenom funkcijom (definirano OS-om, tj. bibliotekom pri stvaranju programa)
obradi signal prethodno definiranom funkcijom (dretva definira)
privremeno zadrži (blokiraj) signal (ne odbacuje se već čeka promjenu ponašanja za njega)

13. (3 boda) Operacijski sustav omogućava korisničkim dretvama *registraciju* na pojedine asinkrone događaje (npr. na istek alarma, na primitak signala) tako da se za te događaje pozovu odgovarajuće funkcije. Kako implementirati te funkcionalnosti u OS-u? Koji se problemi tu javljaju i kako ih riješiti (npr. kako su riješeni u prikazanom sustavu)?

Treba pozvati zadanu funkciju koju je dretva definirala, ali u kontekstu dretve! Problem je to implementirati kada u sustavu postoje različiti načini rada procesora i različiti procesi jer tada treba pripremiti okruženje za poziv zadane funkcije.
Jedan od načina je stvoriti novu dretvu unutar istog procesa, s istim svojstvima kao dotična dretva, te obradu funkcije obaviti u toj dretvi (ta je funkcija početna za dretvu).
Drugi je način „umetnuti“ obradu te funkcije zadanoj dretvi. Obzirom da se to radi u jezgrinoj funkciji kada je kontekst dretve na njenom stogu (ili drugom mjestu), može se taj kontekst pohraniti, a umjesto njega napraviti novi, tako da kada se vratimo u tu dretvu najprije idemo u obradu zadane funkcije (zadanog događaja)

14. (3 boda) Prijelaz s „flat“ modela upravljanja spremnikom („bez upravljanja“, svi vide sve u apsolutnim adresama, k0-k11) na straničenje zahtijeva znatne izmjene u implementaciji. Kratko opisati potrebne izmjene (gdje i koje).

za svaki proces treba izgraditi tablicu prevođenja
promjene su:
u pohrani konteksta i obnovi konteksta (treba pohraniti/ažurirati potrebne registre koji upravljaju straničenjem i pokazuju na tablicu prevođenja);
podsustav za upravljanje tablicama prevođenja (stvaranje, dodavanje zapisa, ...)

komunikacija jezgra-dretva i obratno je „otežana“ jer treba interno prevoditi adrese (programski)

15. (2 boda) Što sve spada u opisnik procesa?

1. id procesa, id korisnika, ...
2. opis adresnog prostora procesa
3. opis načina raspoređivanja i prioritete
4. popis dretvi
5. popis korištenih resursa sustava
6. maske za signale (mada to je bolje u opisnike dretvi)

Pismeni ispit iz predmeta *Operacijski sustavi za ugrađena računala*, 2. 6. 2011.

1. (8) Jedan projekt za ugrađene sustave sastoji se od tri datoteke: `startup.c`, `mkernel.c` i `programs.c` (te njihova zaglavlja, ekvivalentne `.h` datoteke). Pri prevođenju sve će se izlazne (.o) datoteke sastojati samo od `.text`, `.rodata`, `.data` i `.bss` odjeljaka. Izlazna datoteka `kernel.bin` koja nastaje povezivanjem sve tri izlazne objektne datoteke (`startup.o`, `kernel.o` i `programs.o`) posebnim se alatom upisuje u stalni spremnik (ROM) na adresi 0x20000. Pretpostaviti da će program u `startup.c` prekopirati podatke jezgre (`.rodata`, `.data` i `.bss` iz `mkernel.o`), instrukcije i podatke programa (sve iz `programs.c`) u radni spremnik (RAM) na adresu 0x100000. Nadalje, pretpostaviti da se koristi neki oblik upravljanja spremnikom (dinamičko ili straničenje) te da se *programi* nalaze u logičkom prostoru koji započinje s adresom 0x30000. Napisati skriptu `skripta.ld` za povezivača (*linkera*) tako da navedeno bude moguće (uključujući dodavanje potrebnih varijabli).

2. (4) Napisati `Makefile` za prevođenje prethodnog projekta uz pretpostavku da su zastavice za prevođenje: `-c -O -m32 -nostdlib` te zastavice za povezivanje: `-O -T skripta.ld -s`.

3. (3) Napisati jedan makro koji pozive:
   `CNT_INC(sent, num)` zamjenjuje sa `dataset.stat.m_sent +=num`,
   `CNT_INC(recv, num)` sa `dataset.stat.m_recv +=num` te
   `CNT_INC(errors, num)` sa `dataset.stat.m_errors +=num`.

4. (3) Zašto makro: `#define NEG(X) -X` nije općenito dobar, ako želimo da on vraća vrijednost koja po iznosu odgovara X, ali je suprotnog predznaka? Kako treba popraviti makro da bude uvijek upotrebljiv?

5. (4) Operacije (funkcije `op1-op6`) koje se ostvaruju u datotekama `op.h` i `op.c` koriste se: `op1` do `op3` samo unutar datoteke, `op4-op6` unutar i izvan datoteke. Funkcije `op2`, `op4` i `op6` koriste globalne varijable `var1` i `var2` definirane u datoteci `op.c`. Funkcije `op5` i `op6` te varijabla `var2` se koriste i iz datoteke `op2.c`, dok `var1` samo u `op.c`. Varijabla `var1` koristi se u funkcijama `op4` i `op6`, s time da se `op4` poziva iz obrade prekida (`op6` uvijek samo izravno iz dretve). Funkcije `op3`, `op4` i `op5` su kratke te ih treba ugraditi na mjesto poziva (`inline`). Skicirati `op.h`, `op.c` i `op2.c` uz pretpostavku da su sve funkcije tipa `int op (int p);` te da sve što nije potrebno izvan datoteke se i ne može koristiti izvan datoteke. Pri deklaraciji varijable (neka su sve tipa `int`) koristiti dodatno potrebne ključne riječi radi postizanja navedenih zahtjeva. Skica datoteka koje treba nadopuniti je u nastavku.

```
op.c:                                op.h:                          op2.c:
int var1;
int var2;

int op1 (int p) {„kod"}
int op2 (int p) {„kod"}
int op3 (int p) {„kod"}
int op4 (int p) {„kod"}
int op5 (int p) {„kod"}                                             „ostatak op2.c"
int op6 (int p) {„kod"}
```

6. (3) Koja je prednost a koji nedostatak u korištenju makroa u usporedbi s funkcijama? Gdje ima smisla koristiti makro a gdje nema (već je bolje koristiti funkciju)?

7. (6) Skicirajte ostvarenje osnovnog nadzornog alarma (*watchdog timer*) korištenjem sučelja za upravljanje vremenom: `postavi_alarm (id, vrijeme, funkcija)` (`id` je opisnik alarma koji vraća navedena funkcija; prvi puta se poziva s `NULL`). Prikazati korištenje sučelja na primjeru.

8. (8) U sustavu gdje se jezgrine funkcije izravno pozivaju (npr. sve prije *Chapter_08*) treba ostvariti sučelje `sem_timed_wait (sem, max_wait)`. Opišite što je sve potrebno promijeniti u jezgri da se to može ostvariti. Skicirajte tu funkciju i sve dodatne potrebne.

9. (3) Za upravljanje vremenom potrebno nam je odgovarajuće sklopovlje. Koje su osnovne mogućnosti sklopovlja potrebne za to?

10. (8) Neki procesor ima samo jednu prekidnu liniju (žicu) na koju su spojene sve naprave. Skicirati potrebnu strukturu podataka te jezgrine funkcije za ostvarenje prekidnog podsustava.

Pismeni ispit iz predmeta *Operacijski sustavi za ugrađena računala*, 20. 6. 2011.

1. (8) Sustav za koji se programira sastoji se od ROM-a na adresi 0 te RAM-a na adresi 0x100000. Jedan projekt sastoji se od datoteka: `startup.c`, `mkernel.c`, `p1.c`, `p2.c` i `p3.c`. Pri prevođenju sve će se izlazne (.o) datoteke sastojati samo od `.text`, `.rodata`, `.data` i `.bss` odjeljaka. Izlazna datoteka `kernel.bin` koja nastaje povezivanjem svih izlaznih objektnih datoteka posebnim se alatom upisuje u ROM.

   Napisati program u `startup.c` koji treba prekopirati podatke jezgre i sve iz programa (`.rodata`, `.data` i `.bss` iz `mkernel.o` te `.text`, `.rodata`, `.data` i `.bss` iz `p*.c`) u radni spremnik (RAM) na adresu 0x100000. Pritom, obzirom da se koristi straničenje *programe* (svaki zasebno) treba pripremiti za adresu 0 (a podatke jezgre za adresu 0x10000, kôd za ROM gdje se prvotno i učitava).

   Napisati skriptu `skripta.ld` za povezivača (*linkera*) tako da navedeno bude moguće.

2. (6) Napisati `Makefile`, `zaglavlje.h` te nadopuniti datoteke `int.c` i `double.c` tako da sustav radi ispravno.

`int.c`:
```
int suma = 0;
int brojac = 0;
inline int zbroji ( int a, int b )
{ return a + b; }
inline int podijeli ( int a, int b )
{ return a / b; }
inline void dodaj_i ( int broj )
{ suma = zbroji ( suma, broj ); brojac++; }
inline int sv_i ()
{ return podijeli ( suma, brojac ); }
```

`prekid.c`:
```
#include „zaglavlje.h"
static int dohvati_pod_i ( int broj ) { … }
static double dohvati_pod_d ( int broj ) { … }

void prekidna_rutina ( int broj ) {
   if ( broj > 0 )
      dodaj_i ( dohvati_pod_i (broj) );
   else
      dodaj_d ( dohvati_pod_d (broj) );
}
```

`double.c`:
```
double suma = 0;
int brojac = 0;
inline int zbroji ( double a, double b )
{ return a + b; }
inline int podijeli ( double a, double b )
{ return a / b; }
inline void dodaj_d ( double broj )
{ suma = zbroji ( suma, broj ); brojac++; }
inline int sv_d ()
{ return podijeli ( suma, brojac ); }
```

`main.c`:
```
#include „zaglavlje.h"
int main () {
   dodaj_i ( 0 );
   dodaj_d ( 0 );
   inicijaliziraj_prekide ( prekidna_rutina );
   for ( ; ; ) {
      sleep (1);
      printf ( „%d:%lf\n", sv_i(), sv_d() );
   }
   return 0;
}
```

3. (8) Ostvariti podsustav za upravljanje vremenom, tj. samo osnovnu (i jedinu) jezgrinu funkciju `void k_postavi_alarm ( kada, akcija )`. Željena funkcionalnost ne uključuje naknadnu promjenu već postojećih alarma ili njihovo brisanje već se oni nakon aktivacije (nakon poziva zadane funkcije `akcija`) tada miču (tu funkcionalnost također ostvariti). Pretpostaviti da na raspolaganju stoje operacije za rad s listom (`list_init`, `list_dodaj_sort (list, elem, funkc_usporedbe)`, `list_prvi`, …) te sučelje *arch* sloja `arch_postavi_alarm ( kada, akcija )` (sučelje *arch* sloja pamti samo zadnji alarm, prethodno postavljeni se prebriše novim).

4. (3) Napraviti makro koji povećava dvije varijable za jedan, tj. popraviti idući makro tako da on bude uporabljiv u svim smislenim dijelovima programa: `#define INC(A,B) A++;B++`

5. (3) Zadan je makro i njegov poziv iz kôda:
   definicija: `#define LOG(LEVEL, X) printf (#LEVEL „-%d", status->m_ ## X)`
   poziv iz kôda: `LOG ( ERROR, sent );`

   Napisati kako će izgledati kôd nakon preprocesorske obrade, tj. kako će se poziv zamijeniti s makroem (a prije prevođenja „čistog" C-a).

6. (4) U predlošcima za razne elemente korišteno je posebno oblikovano „sučelje", primjerice:

`device_t`, `arch_timer_t`, i `arch_ic_t`. Kako se ta sučelja koriste u jezgri, a kako im se pristupa iz programa (prije *Chapter_8*)? Opišite osnovni princip (može na primjeru).

7. (8) Proširiti monitorske funkcije (i ostale potrebne podatkovne strukture monitora i/ili opisnika dretve) da se omogući mehanizam rekurzivnog zaklučavanja. U nastavku su navedene monitorske funkcije (iz skripte).

```
j_funkcija Zaključaj_monitor ( m )
{
        ako ( Mon[m].v == 1 )
        {
                Mon[m].v = 0;
        }
        inače {
                stavi_u_red ( Mon[m].r, Aktivna_dretva );
                odaberi_aktivnu_dretvu();
        }
}
j_funkcija Otključaj_monitor ( m )
{
        ako ( red Mon[m].r je prazan )
        {
                Mon[m].v = 1;
        }
        inače {
                stavi_u_red ( Pripravne_dretve, uzmi_prvu_iz_reda ( Mon[m].r ) );
                odaberi_aktivnu_dretvu();
        }
}
j_funkcija Čekaj_u_redu_uvjeta ( m, red )
{
        pohrani_pripadajući_monitor ( Aktivna_dretva, m );
        stavi_u_red ( Red_uvjeta[red].r, Aktivna_dretva );

        ako ( red Mon[m].r je prazan )
        {
                Mon[m].v = 1;
        }
        inače {
                stavi_u_red ( Pripravne_dretve, uzmi_prvu_iz_reda ( Mon[m].r ) );
                odaberi_aktivnu_dretvu();
        }
}
j_funkcija Propusti_iz_reda ( red )
{
        ako ( red Red_uvjeta[red].r nije prazan )
        {
                prva = uzmi_prvu_iz_reda ( Red_uvjeta[red].r );
                m = dohvati_pripadajući_monitor (prva);

                ako ( Mon[m].v == 1 ) //dretva se propušta u monitor
                {
                        Mon[m].v = 0;
                        stavi_u_red ( Pripravne_dretve, prva );
                        odaberi_aktivnu_dretvu();
                }
                inače { //neka druga dretva je u monitoru, treba pričekati da izađe
                        stavi_u_red ( Mon[m].r, prva );
                }
        }
}
```

8. (3) Navesti prednosti i nedostatke izravnog pozivanja jezgrinih funkcija (kao što je to do *Chapter_7*) naspram poziva preko programskih prekida (*Chapter_8*).
9. (3) Navesti probleme dinamičkog upravljanja spremnikom (dinamičko upravljanje kao što je ostvareno u `malloc/free`, ne upravljanje procesima). Navesti nekoliko algoritama te njihova dobra i loša svojstva.
10. Navesti potrebnu jezgrinu strukturu podataka za opis procesa ako se za upravljanje spremnikom koristi: a) (2) segmentacija, b) (2) straničenje.

Operacijski sustavi za ugrađena računala - ponovljeni završni ispit

1. (5) Neki projekt sastoji se od dva programa `p1.c` i `p2.c` te zajedničkog dijela `kernel.c`. Napisati skriptu za povezivača (*linkera*) tako da se, osim za kod, konstante i podatke (`.text`, `.rodata*`, `.data*`, `.bss*`) za svaki program zasebno dodatno rezervira i 16 KB za stog te 300 KB za gomilu (*heap*). Počeci dodatnih dijelova neka budu poravnati na 4 KB (zadnjih 12 bitova početne adrese moraju biti nule). Napisati *Makefile* koji će to prevesti i povezati (pretpostaviti da nisu potrebne posebne zastavice, osim `-c` za prevođenje te `-T ime_skripte -o slika.elf`).

2. (3) Navedite primjer gdje se makro:
   `#define INC(A,B) { (A)++; (B)++; }` ne može koristiti (daje krivi rezultat ili grešku pri prevođenju, dok se (bolji) makro:
   `#define INC(A,B) do { (A)++; (B)++; } while (0)` može.

3. (4) Neku napravu (`uredjaj_X`) treba dodati u sustav preko `device_t` sučelja:

```
struct _device_t_;
typedef struct _device_t_ device_t;

struct _device_t_
{
   /* device interface */
   int (*init) ( uint flags, void *params, device_t *dev );
   int (*destroy) ( uint flags, void *params, device_t *dev );
   int (*send) ( void *data, size_t size, uint flags, device_t *dev );
   int (*recv) ( void *data, size_t size, uint flags, device_t *dev );
};
```

   Neka su operacije nad napravom ostvarene funkcijama (te funkcije postoje!):

```
int uredjaj_X_posalji ( void *sto, size_t koliko );
int uredjaj_X_primi ( void *kamo, size_t koliko );
```

   Definirati sučelje (varijablu `uredjaj_x` tipa `device_t`) za tu napravu te ju popuniti (statički). Sve dodatne potrebne funkcije za to sučelje također napisati. Pretpostaviti da nije potrebna nikakva inicijalizacija naprave.

4. (3) Funkcija `int print ( char *format, ... )` prima varijabilan broj parametara. Pretpostavimo da se u ostvarenju te funkcije poziva druga funkcija `int print_dev ( device_t *dev, __________ )` (prema skici ispod) kojoj treba proslijediti iste parametre. Kako to napraviti? Nadopuniti slijedeći kod i komentirati.

```
int print ( char *format, ... ) {
   device_t *dev = dohvati_stdout (); //funkcija postoji negdje

   return print_dev ( dev, ____________________________________ );
}
```

5. (4) Prihvat i obrada prekida može se napraviti na nekoliko načina (prekidni podsustav). Navedite ih te ih kratko opišite (prednosti i nedostaci).

6. (3) Pri odabiru algoritma za dinamičko upravljanje spremnikom treba uzeti u obzir nekoliko svojstava algoritama. Koja?

7. (5) Raspoređivanje prema prioritetu kao prvom te prema prispijeću (FIFO) kao drugom

kriteriju najčešći je oblik raspoređivanja ostvaren u RT sustavima. Koja je struktura podataka potrebna da se odabir jedne od pripravnih dretvi obavi u složenosti O(1), tj. da ne ovisi o broju pripravnih dretvi (npr. odabir se uvijek obavlja u 2 koraka). Prikažite na primjeru sustava koji ima 32 prioriteta te sve potrebne instrukcije za manipulaciju nad 32-bitovnim riječima.

8. (3) Neki mikroprocesor ima registre opće namjene R0 do R7 te programsko brojilo (PC), kazaljku stoga (SP) i registar stanja (SR). Procesor **nema** više načina rada, tj. i pri pojavi prekida ostaje u istom načinu rada, ali na trenutni stog sprema minimalni kontekst. Navedite **instrukcije** koje procesor prvo mora obaviti u **proceduri za obradu prekida** te sadržaj stoga **nakon** izvođenja tih instrukcija (nakon prihvata prekida i tih instrukcija). Pretpostaviti generički procesor s uobičajenim instrukcijama.

9. (3) Što je to latentna (idle) dretva, koja je njena uloga i koja su njena (poželjna) svojstva?

10. (5) U pseudokodu prikazati ostvarenje semafora koji (atomarno) povećavaju i smanjuju vrijednost semafora i za veće vrijednosti od jedan, tj. ostvariti **jezgrine** funkcije `ČekajSemafor(id, broj)` i `PostaviSemafor(id, broj)` (`broj` je uvijek veći od nule). Pretpostaviti da je red blokiranih dretvi ostvaren po redu prispijeća (sve dretve imaju isti prioritet) te da se **UVIJEK** najprije oslobađa prva dretva (kada se za to stvore uvjeti). Primjerice, ako dretva X pozove `ČekajSemafor(sem, 10)`, a trenutna vrijednost semafora je 7, dretva X će se blokirati. Ako tada dretva Y pozove `ČekajSemafor(sem, 1)` i ona će se blokirati! Ako tada dretva Z pozove `PostaviSemafor(sem, 5)` propuštaju se obje dretve. Pretpostaviti da se jezgrine funkcije izravno pozivaju (kao do uključivo *Chapter_7*). Po potrebi proširiti opisnik semafora i dretvi (nije potrebno za najjednostavnije rješenje!).

11. U nekom sustavu zbivaju se događaji koji traže obradu. Dijelovi obrada traže sredstva zaštićena binarnim semaforima. Obrada događaja obavlja se u zasebnim dretvama s odgovarajućim prioritetima (pojava događaja višeg prioriteta istiskuje obradu manjeg prioriteta, tj. odgovarajuće dretve). Neka su događaji (za promatrani vremenski interval) zadani vremenima pojave (javljaju se samo jednom) i pseudokodom:

    t_1: t_1=10; { zauzmi(s_1); radi(15); zauzmi(s_2); radi(5); oslobodi(s_1, s_2); radi(5); }
    t_2: t_2=20; { zauzmi(s_2); radi(10); zauzmi(s_1); radi(5); oslobodi(s_1, s_2); radi(5); }
    t_3: t_3=25; { zauzmi(s_2); radi(5); zauzmi(s_1); radi(5); oslobodi(s_1, s_2); radi(5); }

    (Zadatak t_1 pojavljuje se u 10 jedinici vremena s zadanim poslom, ...)
    Zadatak t_3 ima najveći prioritet. Prikazati što će se dogoditi u sustavu ako se koristi:
    a) (3) samo raspoređivanje *prema prioritetu,*
    b) (2) *protokol stropnog prioriteta* – jednostavnija inačica (navesti stropne prioritete).

12. (4) Signali se dretvi mogu uputiti u bilo kojim trenucima (kada radi ali i kada čeka). Kako se taj problem rješava u operacijskim sustavima (primjerice UNIX/Linux)?

13. (3) Kada se procesima upravlja dinamičkom metodom upravljanja spremnikom, procesi ostaju u logičkom adresnom prostoru. Međutim, iz jezgrinih funkcija treba dohvaćati i mijenjati podatke u procesu. Navedite strukture podataka (dio opisnika procesa) te funkcije koje pretvaraju logičku adresu u fizičku i obratno (a koje su potrebne!).

Operacijski sustavi za ugrađena računala – *dekanski rok*

1. (4) Neki projekt sastoji se od dva programa `p1.c` i `p2.c` te zajedničkog dijela `kernel.c`. Napisati skriptu `skripta` za povezivača (*linkera*) tako da se, osim za kod, konstante i podatke (`.text`, `.rodata*`, `.data*`, `.bss*`) za svaki program zasebno dodatno rezervira i 16 KB za stog te 300 KB za gomilu (*heap*). Počeci dodatnih dijelova neka budu poravnati na 4 KB (zadnjih 12 bitova početne adrese moraju biti nule). Napisati *Makefile* koji će to prevesti i povezati (pretpostaviti da nisu potrebne posebne zastavice, osim `-c` za prevođenje te `-T skripta -o slika.elf`).

2. (1) Napisati makro INC2(A,B) koji će obje varijable povećati za jedan. Makro napraviti tako da se može pozivati od bilo kuda (kao da je funkcija, uz razliku što ne vraća vrijednost).

3. (2) Neku napravu (`uredjaj_X`) treba dodati u sustav preko `device_t` sučelja:

```
struct _device_t_;
typedef struct _device_t_ device_t;

struct _device_t_
{
   /* device interface */
   int (*init) ( uint flags, void *params, device_t *dev );
   int (*destroy) ( uint flags, void *params, device_t *dev );
   int (*send) ( void *data, size_t size, uint flags, device_t *dev );
   int (*recv) ( void *data, size_t size, uint flags, device_t *dev );
};
```

   Neka su operacije nad napravom ostvarene funkcijama (te funkcije postoje i trebaju se koristiti za ostvarenje „upravljačkog programa”):

```
int uredjaj_X_posalji ( void *sto, size_t koliko );
int uredjaj_X_primi ( void *kamo, size_t koliko );
```

   Definirati sučelje („upravljački program”, tj. varijablu `uredjaj_x` tipa `device_t`) za tu napravu te ju popuniti (statički). Sve dodatne potrebne funkcije za to sučelje također napisati. Pretpostaviti da nije potrebna nikakva inicijalizacija naprave.

4. (1) Funkcija `int print ( char *format, ... )` prima varijabilan broj parametara. Pretpostavimo da se u ostvarenju te funkcije poziva druga funkcija `int print_dev ( device_t *dev, __________ )` (prema skici ispod) kojoj treba proslijediti iste parametre. Kako to napraviti? Nadopuniti slijedeći kod i komentirati.

```
int print ( char *format, ... ) {
   device_t *dev = dohvati_stdout (); //funkcija postoji negdje

   return print_dev ( dev, ____________________________________ );
}
```

5. (2) Raspoređivanje prema prioritetu kao prvom te prema prispijeću (FIFO) kao drugom kriteriju najčešći je oblik raspoređivanja ostvaren u RT sustavima. Koja je struktura podataka potrebna da se odabir jedne od pripravnih dretvi obavi u složenosti O(1), tj. da ne ovisi o broju pripravnih dretvi (npr. odabir se uvijek obavlja u 2 koraka). Prikažite na primjeru sustava koji ima 32 prioriteta te sve potrebne instrukcije za manipulaciju nad 32-bitovnim riječima.

6. (2) Neki mikroprocesor ima registre opće namjene R0 do R7 te programsko brojilo (PC),

kazaljku stoga (SP) i registar stanja (SR). Procesor **nema** više načina rada, tj. i pri pojavi prekida ostaje u istom načinu rada, ali na trenutni stog sprema minimalni kontekst. Navedite **instrukcije** koje procesor prvo mora obaviti u **proceduri za obradu prekida** te sadržaj stoga **nakon** izvođenja tih instrukcija (nakon prihvata prekida i tih instrukcija). Pretpostaviti generički procesor s uobičajenim instrukcijama.

7. (5) Ostvariti semafore u jednostavnom modelu jezgre (definirati strukture podataka i funkcije). Pretpostaviti da postoji struktura podataka `kthread_q` (za red dretvi), te da postoje funkcije:
   `k_threadq_init(kthread_q *red)` - inicijalizira red za dretve,
   `k_release_thread(kthread_q *red)` - prvu dretvu iz zadanog reda prebacuje je u red pripravnih,
   `k_enqueue_thread(kthread_q *red)` - pozivajuću dretvu stavlja u navedeni red (aktivnu dretvu blokira u taj red) te
   `k_schedule_threads()` - među pripravnim dretvama i trenutno aktivnom dretvom odabire dretvu najveća prioriteta i aktivira ju (ta postaje aktivna i nastavlja s radom).
   Pretpostaviti da se te funkcije za ostvarenje semafora (`sem_wait`, `sem_post`, `sem_init` i `sem_destroy`) izravno pozivaju (ne mehanizmom prekida). Pretpostaviti postojanje i drugih potrebnih funkcija (`kmalloc`, `kfree`, `enable/disable_interrutps`).

8. (1) Signali se dretvi mogu uputiti u bilo kojim trenucima (kada radi ali i kada čeka). Kako se taj problem rješava u operacijskim sustavima (primjerice UNIX/Linux)?

9. (2) Kada se procesima upravlja dinamičkom metodom upravljanja spremnikom, procesi ostaju u logičkom adresnom prostoru. Međutim, iz jezgrinih funkcija treba dohvaćati i mijenjati podatke u procesu. Navedite strukture podataka (dio opisnika procesa) te funkcije koje pretvaraju logičku adresu u fizičku i obratno (a koje su potrebne!).

1. kratka provjera znanja iz OSZUR-a, 22. 3. 2012. _____

ime i prezime

1. (0,5) Koji je zajednički razlog podjele OS-a na podsustave te podjele izvornog koda OS-a na slojeve (arch/kernel/programs)?

2. (0,5) Neka je zadan Makefile:

```
1: hello: hello.c hello.h print.h
2:        gcc hello.c -o hello
```

Čemu služe imena datoteka navedena u 1. liniji, nakon `hello:`? Zašto su potrebna?

3. (1) Neka je zadan program:

```
#include <stdio.h>
int a;
int b = 5;

int main () {
      char poruka[] = "Suma=%d\n";
      int c = 3;

      printf ( poruka, a = b + c );

      return a;
}
```

Koji će dijelovi gornjeg programa biti u kojim odjeljcima (.text, .data, .rodata, .bss) prevedenog programa (npr. sa `gcc prog.c`)? Navedite barem dva dijela koja spadaju u različite odsječke.

4. (1) Nakon što se zahtjev za prekid pojavio, nakon što je procesor dovršio tekuću instrukciju, uz omogućeno prihvaćanje prekida od strane procesora, u postupku prihvata prekida procesor će dalje raditi slijedeće:

a) _____

_____

_____

_____