1. [2 boda] U zaglavlju koje se uključuje iz datoteke `test.c` nalazi se i makro:

```
#define LOG(LEVEL, format, ...) printf("[" #LEVEL ":%s:%d] " format \
"\n", __FILE__, __LINE__, ##__VA_ARGS__)
```

U datoteci `test.c` u liniji 25 makro se poziva sa:

```
LOG ( WARN, "X je izvan granica [10-100]! X=%d", X );
```

Prikazati kako će ta linija izgledati nakon prve faze prevođenja (nakon "preprocesinga", nakon evaluacije makroa).

```
printf("[" "WARN" ":%s:%d] " "X je izvan granica [10-100]! X=%d" \
"\n", "test.c", 25, X);
tj. nakon spajanja konstantnih stringova:
printf("[WARN:%s:%d] X je izvan granica [10-100]! X=%d\n", "test.c", 25, X);
```

2. [3 boda] Sklop za prihvat prekida ima dva registra: KZ (kopija zastavica) i TP (tekući prioritet). Bitovi različiti od nule u registru KZ označavaju da jedna ili više naprava spojenih na taj ulaz traži obradu prekida, dok nule označavaju naprave koje ne traže prekid ili je njihov zahjev prihvaćen (u obradi). Naprava spušta svoj zahtjev kada se pozove funkcija za obradu prekida te naprave. U sustavu ima mnogo naprava i neke (one ista prioriteta) dijele ulaz sklopa za prihvat prekida. Prekidi većeg prioriteta trebaju prekidati obradu prekida manjeg prioriteta. U registru TP bitovi postavljeni u 1 označavaju da je dotični prekid u obradi (procesor postavlja i briše registar TP – to treba ugraditi u kod). Npr. ako je vrijednost KZ $= 00010011_2$ i TP $= 00100100_2$ tada naprave s prioritetima 4, 1 i 0 imaju postavljen zahtjev za prekid dok se trenutno obrađuje zahtjev naprave prioriteta 5, a prekinuta je obrada naprave prioriteta 2 (koja se treba nastaviti po završetku prioritetnijih). Neka postoji funkcija $msb(x)$ koja vraća indeks najznačajnije jedinice (npr. $msb(001000_2) = 3$). Ostvariti prekidni podsustav (`void inicijaliziraj()`, `void registriraj_prekid (int irq, void (*obrada)())` te `void prihvat_prekida()` (bez argumenata!) koja se poziva svaki puta kad se prekid prihvati, za opisani sustav uz pretpostavku da će sklop proslijediti zahtjev većeg prioriteta od tekućeg prema procesoru, dok će one istog ili manjeg prioriteta zadržati (sklop uspoređuje KZ i TP). Pri prihvatu prekida, prije poziva `prihvat_prekida` kontekst prekinuta posla spremljen je na stog, a nakon povratka iz iste funkcije sa stoga se obnavlja kontest (ne treba ga programski spremati/obnavljati). Pretpostaviti postojanje sustava dinamičkog upravljanja gomilom (`kmalloc/kfree`) te operacija za rad s listom (`dodaj_u_listu(&lista,&element)` i `uzmi_prvi_iz_liste(&lista)`).

```
lista_t lista[N];
void inicijaliziraj() {
    int i;
    TP = 0;
    for ( i = 0; i < N; i++ )
        inicijaliziraj_listu ( &lista[i] );
}
void registriraj_prekid (int irq, void (*obrada)()) {
    // registracija NIJE POJAVA prekida !!!
    dodaj_u_listu ( &lista[irq], obrada );
}
void prihvat_prekida() {
    irq = msb (KZ);
    TP = TP | (1<<irq);
    void (*obrada)();
    obrada =  uzmi_prvi_iz_liste ( &lista[irq] );
    while ( obrada ) {
        omogući_prekidanje();
        obrada();
        zabrani_prekidanje();
        obrada =  uzmi_prvi_iz_liste ( &lista[irq] );
    }
    TP = TP ^ (1<<irq);
}
```

3. [3 boda] Neki sustav posjeduje 16 bitovno brojilo na adresi `CNT` koje odbrojava frekvencijom od 50 kHz. Upisom neke vrijednosti u brojilo započinje odbrojavanje prema nuli. Kada brojilo dođe do nule, izaziva prekid te se učitava zadnja upisana vrijednost pa ponovno kreće s odbrojavanjem. Izgraditi sustav upravljanja vremenom koji treba imati sučelja:

   a) inicijalizacija podsustava: `void inicijaliziraj()`
   b) obrada prekida brojila: `void prekid_sata()`
   c) dohvat trenutna sata: `long dohvati_vrijeme()` (vraća vrijeme u ms)
   d) promjena trenutna sata: `void postavi_vrijeme(long novo_vrijeme_ms)`
   e) postavljanje alarma: `void alarm(long za_koliko_ms, void (*obrada)())`

   Sučelje koristi vrijednost sata u milisekundama (pretpostaviti da je tip `long` dovoljan za prikaz sata u milisekundama). Međutim, interno, obzirom da zahtjevi za postavljanjem alarma mogu doći u bilo kojem trenutku, preciznost treba biti veća (u rezoluciji brojila). Promjena sata i postavljanje alarma briše prethodno postavljeni alarm (on se ne poziva). Rješenje neka izaziva prekide što rijeđe, samo kada je to potrebno (kad je to moguće učitavati najveću vrijednost u brojilo). Pretpostaviti da postoje makroi: `OTK_US(O)` koji pretvara broj otkucaja brojila u mikrosekunde i obratno `US_OTK(US)` (vrijeme u mikrosekundama u potreban broj otkucaja brojila).

```
//makroi - nije ih trebalo ostvariti
#define OTK_US(O)     ((O)*20)
#define US_OTK(US)    ((US)/20)

#define MAX_BR 0xffff          //najveći broj koji stane u brojilo
#define MAX_US OTK_US(MAX_BR) //najveći interval između prekida sata
#define MAX_MS (MAX_US/1000)

long sat_ms, sat_us, ucitano, *cnt, odgoda_ms;
void (*alarm)();

void inicijaliziraj () {
    sat_ms = sat_us = odgoda_ms = 0;
    postavi_brojilo ( MAX_MS );
}
void prekid_sata () {
    // pretpostavka je da će se prekid prihvatiti prije nego li brojilo
    // odbroji još koji otkucaj; inače bi trebalo ažurirati drukčije
    azuriraj_sat ();
    if ( odgoda_ms > 0 ) {
        odgoda_ms -= OTK_US ( ucitano ) / 1000;
        if ( odgoda_ms <= 0 ) {
            postavi_brojilo ( MAX_MS );
            alarm();
        }
        else {
            postavi_brojilo ( odgoda_ms );
        }
    }
}
long dohvati_vrijeme () {
    return sat_ms + ( sat_us + OTK_US ( ucitano - (*cnt) ) ) / 1000;
}
void postavi_vrijeme ( long novo_vrijeme_ms ) {
    sat_ms = novo_vrijeme_ms;
    sat_us = odgoda_ms = 0;
    postavi_brojilo ( MAX_MS );
}
```

(nastavak na idućoj stranici)

```
void alarm ( long za_koliko_ms, void (*obrada)() ) {
    azuriraj_sat();
    odgoda_ms = za_koliko_ms;
    postavi_brojilo ( odgoda_ms );
}
void azuriraj_sat () {
    long us = sat_us + OTK_US ( ucitano - (*cnt) );
    sat_ms += us / 1000;
    sat_us = us % 1000;
}
void postavi_brojilo ( long ms ) {
    long usec;
    if ( ms >= MAX_MS )
        ucitano = MAX_BR;
    else
        ucitano = US_OTK ( ms * 1000 );
    *cnt = ucitano;
}
```

4. [3 boda] Izvorni kod neka sustava sastoji se od datoteka u direktorijima: `arch`, `kernel`, `modules` i `programs`. Sliku sustava nastalu prevođenjem treba učitati u ROM na adresi 0x10000. Instrukcije i konstante treba pripremiti tako da ostaju u ROM-u dok ostale varijable treba pripremiti za RAM na adresi 0x30000, osim za programe. Programe pripremiti za logičke adrese, počevši s adresom 0. Obzirom da se koristi straničenje, instrukcije i konstante koje programi koriste moći će se koristiti iz ROM-a, ali će podatke operacijski sustav pri pokretanju programa prekopirati u RAM (na neku slobodnu adresu). Napisati skriptu za povezivača, u koju ugraditi i potrebne varijable, a da bi se kopiranje podataka jezgre i programa moglo obaviti.

```
ROM = 0x10000;
RAM = 0x30000;

SECTIONS {
    .rom1 ROM : AT(ROM) {
        arch* kernel* modules* (.text .rodata)
    }
    adr1 = ROM + SIZEOF(.rom1);
    .rom2 0 : AT(adr1) {
        programs* (.text .rodata)
    }
    adr2 = adr1 + SIZEOF(.rom2);
    .ram1 RAM : AT (adr2) {
        arch* kernel* modules* (.data .bss)
    }
    adr3 = adr2 + SIZEOF(.ram1);
    .ram2 SIZEOF(.rom2) : AT (adr3) {
        programs* (.data .bss)
    }
    adr4 = adr3 + SIZEOF(.ram2);
}
```

5. [3 boda] Neki pisač je spojen na računalo preko pristupna sklopa koji se u računalu upravlja s dva registra: podatkovnim na adresi `POD` te upravljačkim na adresi `UPR`. Spremnost za prihvat podataka pisač iskazuje postavljanjem nule u `UPR`. Bilo koja druga vrijednost označava da pisač trenutno ne može prihvaćati nove podatke za ispis. Napisati upravljački program za pisač koji će u slučaju da pisač ne može prihvaćati nove znakove koristiti i međuspremnik veličine 64 KB. Funkcije koje upravljački program treba ostvariti da bi se one mogle uključiti u sustav su:

```
void *inicijaliziraj (); //vraća opisnik naprave - potrebne podatke
int posalji ( void *podaci, size_t duljina, void *opisnik );
int procitaj ( void *podaci, size_t duljina, void *opisnik );
int dohvati_status ();

#define VMS (64*1024)

struct podaci {
    char ms[VMS];
    size_t prvi, zadnji, ima;
};

void *inicijaliziraj () {
    struct podaci *ms = kmalloc ( sizeof(struct podaci) );
    ms->prvi = ms->zadnji = ms->ima = 0;
    return ms;
}
int posalji ( void *podaci, size_t duljina, void *opisnik ) {
    char *p = podaci;
    size_t jos = duljina;

    //ako ima još nešto u međuspremniku, prvo probaj to slati
    salji_iz_medjuspremnika ( opisnik );

    while ( *UPR == 0 && jos > 0 ) {
        *POD = *p++;
        jos--;
    }

    while ( jos > 0 && op->ima < VMS ) {
        op->ms[op->zadnji] = *p++;
        op->zadnji = ( op->zadnji + 1 ) % VMS;
        op->ima++;
        jos--;
    }

    return duljina - jos; //koliko je poslao i stavio u ms
}
int procitaj ( void *podaci, size_t duljina, void *opisnik ) {
    return 0;
}
int dohvati_status () { return *UPR; }

// ako naprava izaziva prekid, ova funkcija bi se trebala pozvati
void salji_iz_medjuspremnika ( void *opisnik ) {
    struct podaci *op = opisnik;
    while ( *UPR == 0 && op->ima ) {
        *POD = op->ms[op->prvi];
        op->prvi = ( op->prvi + 1 ) % VMS;
        op->ima--;
    }
}
```

6. [3 boda] U nekom sustavu jezgrine funkcije treba pozivati mehanizmom prekida. Ostvariti dio takva sustava i prikazati ga na primjeru poziva `status = ČekajSemafor(s)`. Pokazati ostvarenje te funkcije, ostvarenje prihvata takva prekida i poziv jezgrine funkcije `jf_ČekajSemafor(j_semafor *s)` (prikazanu ispod) i povratak iz jezgrine funkcije u program. Pretpostaviti da postoje funkcije: `izazovi_programski_prekid()`, `dohvati_adresu_prvog_parametara()`, `postavi_povratnu_vrijednost_za_prekinutu_dretvu(vrijednost)` (i ostale koje su bile prikazane u okviru predavanja, ako su potrebne). Pretpostaviti da se struktura `j_semafor` sastoji od vrijednosti semafora i reda za blokirane dretve, dok je `s` tipa `void *` i sadrži fizičku adresu odgovarajuće strukture `j_semafor` (iako se iz procesa toj adresi ne može pristupiti). Dodatno potrebne operacije na "nižoj" razini prikazati pseudokodom (dozvoljeno je "miješanje" pseudokoda i C-a).

```
int jf_ČekajSemafor ( j_semafor *s ) {
    if ( s->vrijednost > 0 ) {
        s->vrijednost--;
    } else {
        blokiraj_trenutnu_dretvu_u_redu(s->red);
        postavi_aktivnu_dretvu()
    }
    return 0;
}

ČekajSemafor(semafor *s) {
    vrati pozovi_j_funkciju ( ČEKAJ_SEMAFOR, s )
}

pozovi_j_funkciju ( int id, ... ) { //u asembleru
    izazovi_programski_prekid() //od tuda ide obrada prekida
    vrati vrijednost iz R0; //prvi registar opće namjene
}

u prihvatu prekida:
    ...
    ako je uzrok_prekida == PROGRAMSKI_PREKID tada
        a = dohvati_adresu_prvog_parametara()
        id = ((int*)a)
        ako je id == ČEKAJ_SEMAFOR tada
            j_semafor *s = a + 1 //idući argument iza id
            pv = jf_ČekajSemafor(s)
            postavi_povratnu_vrijednost(pv)
    ...
```

7. [3 boda] Funkcije iz niže zadanih datoteka pozivaju se i iz drugih datoteka. Opisati što sve ne valja u zadanome kodu. Prepisati te datoteke (ili samo dijelove koje treba promijeniti) ispravnim rješenjem.

```
/* a.c */
# include "sva potrebna zaglavlja su uključna"
int brojac = 0;
static char poruka[] = "12345";
int dodaj ( char *p ) {
    strcat ( poruka, p );   brojac += strlen(poruka) + 1;
    return brojac;
}
/* b.c */
# include "sva potrebna zaglavlja su uključna"
int brojac = 0;
char *u_mala () {
    extern char poruka[]; // varijabla iz datoteke a.c
    int i;
    for ( i = 0; i < strlen(poruka); i++ )
        if ( poruka[i] >= 'A' && poruka[i] <= 'Z' ) {
            poruka[i] = poruka[i] - 'A' + 'a';  brojac++;
        }
    return poruka;
}
int pretvorbi() { return brojac; }
```

```
/* a.c */
# include "sva potrebna zaglavlja su uključna"
static int brojac = 0; //dodan static
char poruka[MAXVEL] = {'1','2','3','4','5',0};
// maknut static, dodana veličina i inicijalizacija
// ili:  char poruka[MAXVEL] = {0}; + kod u dodaj
int dodaj ( char *p ) {
    //if ( poruka[0] == 0 ) strcat ( poruka, "12345" );
    if ( strlen(p) + strlen(poruka) > MAXVEL ) return -1;
    strcat ( poruka, p );   brojac += strlen(poruka) + 1;
    return brojac;
}
/* b.c */
# include "sva potrebna zaglavlja su uključna"
static int brojac = 0;
char *u_mala () {
    extern char poruka[]; // varijabla iz datoteke a.c
    int i;
    for ( i = 0; i < strlen(poruka); i++ )
        if ( poruka[i] >= 'A' && poruka[i] <= 'Z' ) {
            poruka[i] = poruka[i] - 'A' + 'a';  brojac++;
        }
    return poruka;
}
int pretvorbi() { return brojac; }
```