Rješenje zadataka (jedno od ...)

1. [1 bod] Gdje se sve nalaze kopije jedne datoteke koja je u sustavu koji koristi *git*? Pretpostaviti da se radi o jednom projektu – jednom git repozitoriju na poslužitelju te jednog korisnika koji ga koristi (dohvatio ga je sa `git clone ...`).

   Rj.
   a) u repozitoriju na poslužitelju
   b) u lokalnom repozitoriju (.git direktoriju)
   c) u lokalnoj kopiji ("radna inačica")

2. [1 bod] Radi provjere ispravnog rada u kod se ugrađuju dodatne provjere. Neke od njih se izvode samo u ispitnom pokretanju ('DEBUG' načinu), a neke uvijek. Pokazati na primjeru potrebu korištenja oba načina provjera.

   Rj.
   Ispitno pokretanje izvodi se pri razvoju sustava, dok još možda postoje neke greške u kodu (iako vjerojatno postoje i kasnije). Stoga se u tim pokretanjima koriste dodatne direktive pri prevođenju (DEBUG) te se uključuju dodatne provjere, pogotovo na početku funkcija dodatno se provjeravaju poslani parametri. Primjeri takvih ispitivanja su:

```
int neka_funkcija ( tip1 p1, tip2 p2, ... ) {
    ASSERT ( "provjera p1" ); //ili assert
    ...
}
```

   U fazi pravog rada sustava i dalje se mogu provjeravati neki parametri, pogotovo oni koji dolaze iz programa. Takve provjere moraju biti ostvarene običnim kodom (ne makroima ASSERT i sl.). Dodatne provjere tijekom pravog rada uglavnom se svode na probleme nedostatka sredstava i grešaka u radu (koje najčešće nisu rezultat krivog programa već ulaza i sl.). Primjerice, svaki bi zahtjev za stvaranje nekog objekta (npr. dretve) ili zahtjev za dijelom spremnika (malloc) trebalo provjeriti.

```
int neka_druga_funkcija ( tip1 p1, tip2 p2, ... ) {
    if ( "provjera p1" ) {
        "ili vrati grešku, ili prekini dretvu, ili ...";
    }
    ...
   x = malloc (...);
    if ( x == NULL ) {
        "prijavi grešku u neki dnevnik ili korisniku preko zaslona, ..."
        "ili vrati grešku, ili prekini dretvu, ili ...";
    }
    ...
}
```

3. [1 bod] Napraviti makro `POSTAVI(tip, podaci, duljina, poruka)` koji će popuniti varijablu strukture `struct poruka { short tip; char data[1]; }`. Pretpostaviti da je spremnički prostor odgovarajuće duljine za poruku već zauzet.

   Rj.

```
#define POSTAVI(tip, podaci, duljina, poruka)  \
do{ poruka.tip = tip; memcpy ( poruka.data, poruka, duljina ); }while(0)
```

4. [2 boda] Funkcije u nekoj datoteci koriste se i za jezgru i za programe (npr. `memset`). Međutim, obzirom na korištenje sklopovlja za pretvorbu logičkih adresa u fizičke za programe, dok jezgra koristi fizičke adrese (apsolutne), isti se kod dva puta prevodi i kasnije zajedno povezuje ('linka'). Stoga se pri prevođenju trebaju koristiti različita imena funkcija (npr. `memcpy` za programe te `kmemcpy` za jezgru). Napisati dio koda koji definira makro `FUNKCIJA(ime_funkcije)` koji treba koristiti pri deklaraciji funkcija. Preko makroa `JEZGRA` može se doznati je li funkciji treba dodati prefiks `k` (kada je `JEZGRA=1`) ili ne. Npr. korištenje makroa bi izgledalo:

   ```
   void * FUNKCIJA(memcpy) ( void *dest, const void *src, size_t num )
   ```

   što se u početnoj fazi prevođenja treba prevesti u:

   ```
   void * memcpy ( void *dest, const void *src, size_t num )
   ```

   ili

   ```
   void * kmemcpy ( void *dest, const void *src, size_t num )
   ```

   Rj.

   ```
   #if JEZGRA == 1
       #define FUNKCIJA(ime_funkcije)   k ## ime_funkcije
   #else
       #define FUNKCIJA(ime_funkcije)   ime_funkcije
   #endif
   ```

5. [1 bod] Navesti primjere gdje jedan program (dretva) zbog greške može narušiti cijeli sustav. Kojim mehanizmima se navedeni problemi mogu lokalizirati (da greške ne utječu na ostatak sustava)?

   Rj.
   Zbog mijenjanja podataka drugih procesa ili jezgre OS-a. Rješenje: koristiti upravljanje spremnikom koje će onemogućiti procesu da piše van ograđenog prostora.

   Zbog instrukcija koje upravljaju nekim dijelovima sustava. Primjerice, program može zabraniti prekide i time onemogućiti prihvat prekida svih naprava (a time i sata koji bi tu dretvu maknuo s procesora). Rješenje: procese izvoditi u korisničkom načinu rada u kojem ne mogu pokretati takve instrukcije.

6. [2 boda] U pseudokodu prikazati ostvarenje prekidnog podsustava. Ostvariti sve neophodne funkcije za korištenje iz jezgre. Rješenje mora biti potpunije, npr. nije dovoljno napisati 'spremni kontekst dretve' već 'spremni kontekst dretve u opisnik dretve'. Pretpostaviti da se pri prihvatu prekida automatski na stog pohranjuje programsko brojilo i identifikator prekida (broj) te da se u programsko brojilo upisuje vrijednost 10. Pretpostaviti da za svaki broj (do `BR_PREKIDA`) postoji samo jedan mogući izvor prekida.

   Rj.

```
//struktura podataka:
polje_kazaljki_na_funkciju obrada[BR_PREKIDA];

//sučelja za jezgru
inicijalizacija_prekidnog_podsustava() {
    za i = 0 do BR_PREKIDA-1
        obrada[i] = NULL;
    omogući prekidanje;
}

registriraj_prekid ( id, funkcija ) {
    ako je ( id > 0 && id <= BR_PREKIDA )
        obrada[id] = funkcija;
}

//obrada prekida
10: //na adresi 10
    pohrani sve korisničke registre procesora na stog;
    pozovi "obrada_prekida"
    obnovi sve korisničke registre procesora sa stoga;
    vrati se iz prekida; //učitaj PC, makni id sa stoga, dozvoli
       prekidanje

obrada_prekida() {
    x = dohvati_opisnik_aktivne_dretve ();
    kopiraj_kontekst_u_opisnik_dretve;
    id = dohvati_id_prekida_sa_stoga;
    ako je ( obrada[id] != NULL )
        obrada[id] (id);
    x = dohvati_opisnik_aktivne_dretve ();
    kopiraj_kontekst_iz_opisnika_dretve_x_na_stog;
}
```

7. [2 boda] Neki sklop detektira otkucaje sata te preko prikladnog sučelja zapisuje broj 1 na adresu `BEAT`. Korištenjem tog podatka ostvariti sustav koji će na zaslonu uređaja prikazivati:
   a) ukupan broj otkucaja (`b`)
   b) trenutnu frekvenciju otkucaja (broj otkucaja u minuti `bpm`)
   c) procjenjenu potrošnju kalorija po minuti (računati preko `cpm=fun1(bpm)`, `fun1` postoji)
   d) procjenjeni ukupan broj potrošenih kalorija (`cals=fun2(b)`, `fun2` postoji).

   Sustav posjeduje 16-bitovno brojilo koje odbrojava frekvencijom FREQ. Brojilo nije moguće promijeniti (resetirati) niti ono izaziva prekide, već nastavlja s nulom nakon što dosegne najveću vrijednost. Pretpostaviti da se ispis vrijednosti na zaslon zbiva jednostavnim upisom odgovarajućih vrijednosti na zasebne lokacije u spremniku (adrese: `B`, `BPM`, `CPM`, `CALS`). Broj otkucaja po minuti izračunavati na osnovu zadnja četiri otkucaja: $bpm = 60 \cdot 3/(t_4 - t_1)$. Upravljanje ostvariti upravljačkom petljom (beskonačnom petljom).

   Rj.

```
//struktura podataka:
#define MAXCNT  (1<<16)
int b, bpm;
int *pb = B, *pbpm = BPM, *pcpm = CPM, *pcals = CALS;
int *beat = BEAT, *brojilo = BROJILO;
int t[4] = {0,0,0,0}, T;

//upravljačka petlja
void upravljanje () {
    while (1) {
        if ( *beat ) {
            *beat = 0;
            b++;

            t[0] = t[1]; t[1] = t[2]; t[2] = t[3];
            t[3] = *brojilo;

            T = 0;
            if ( t[1] > t[0] )
                T += t[1] - t[0];
            else
                T += t[1] + MAXCNT - t[0];

            if ( t[2] > t[1] )
                T += t[2] - t[1];
            else
                T += t[2] + MAXCNT - t[1];

            if ( t[3] > t[2] )
                T += t[3] - t[2];
            else
                T += t[3] + MAXCNT - t[1];

            bpm = 60 * 3 * FREQ / T;

            *pb = b;
            *pbpm = bpm;
            *pcpm = fun1(bpm);
            *pcals = fun2(b);
        }
    }
}
```

8. [4 boda] U nekom računalnom sustavu postoji 64-bitovno brojilo koje odbrojava frekvencijom od 1 GHz. Korištenjem tog brojila ostvariti:
   a) praćenje dodijeljenog procesorskog vremena pojedinoj dretvi
   b) raspoređivanje podjelom vremena.

   Pretpostaviti da svaki poziv jezgrine funkcije započinje operacijom `deaktiviraj_dretvu` iz koje se poziva funkcija `ažuriraj_vremena` koju treba ostvariti (za a) dio).
   Nadalje, pretpostaviti da u sustavu postoji i drugi satni mehanizam koji periodički izaziva prekide (dovoljno velikom, ali nepoznatom i nestalnom frekvencijom). Iz tih se funkcija poziva `raspored ivanje_podjelom_vremena` koju treba ostvariti (za b) dio). Neka sve dretve trebaju dobiti kvant vremena `T`. Za samo raspoređivanje ne koristiti dodatne alarme već samo poziv `raspoređivanje _podjelom_vremena` koji je ugrađen u sve potrebne funkcije. Za praćenje dobivenog vremena po dretvi koristiti podatke prikupljene u a) dijelu zadatka. Zbog poziva iz drugih funkcija poneka će dretve dobiti i više vremena od T, ali manje ne smije. Za upravljanje dretvama koristiti pozive: `stavi_ u_pripravne(dretva)`, `prva=uzmi_prvu_pripravnu()`, `aktivna=dohvati_aktivnu()` te `postavi_aktivnu(dretva)`.
   Proširiti opisnike dretve po potrebi. Korištenje vremena pojednostaviti (npr. koristiti vrijeme u jedinicama nanosekunde u 64-bitovnim varijablama).

   Rj. a)
   1) uz pretpostavku da jezgrine funkcije traju kratko (zanemarivo).

```
#define BROJILO neka_adresa //neka brojilo odbrojava prema većim gore
#define FREQ    1000000000
#define MAX 0xffffffffffffffff  //2^64-1
long zadnje_očitanje_brojila = 0;
long *brojilo = BROJILO;

ažuriraj_vremena () {
    aktivna = dohvati_aktivnu();
    if ( *brojilo > zadnje_očitanje_brojila )
        t = *brojilo - zadnje_očitanje_brojila;
    else
        t = MAX - zadnje_očitanje_brojila + 1 + *brojilo;

    zadnje_očitanje_brojila = *brojilo;
    aktivna->dobiveno_vremena += t;
}
```

   2) uz pretpostavku da jezgrine funkcije traju duže (nezanemarivo) (informativno)

```
#define BROJILO neka_adresa //neka brojilo odbrojava prema gore
#define FREQ    1000000000
#define MAX 0xffffffffffffffff  //2^64-1
long *brojilo = BROJILO;

ažuriraj_vremena () {
    aktivna = dohvati_aktivnu();
    if ( *brojilo > aktivna->zadnje_očitanje_brojila )
        t = *brojilo - aktivna->zadnje_očitanje_brojila;
    else
        t = MAX - aktivna->zadnje_očitanje_brojila + 1 + *brojilo;

    aktivna->dobiveno_vremena += T;
}
prije_povratka_u_dretvu (aktivna) {
    aktivna->zadnje_očitanje_brojila = *brojilo;
}
```

Rj. b)

```
raspoređivanje_podjelom_vremena () {
    ažuriraj_vremena ();
    aktivna = dohvati_aktivnu();
    t = aktivna->dobiveno_vremena - aktivna->pocetak_kvanta;
    if ( t >= T ) {
        stavi_u_pripravne ( aktivna );
        aktivna = uzmi_prvu_pripravnu ();
        postavi_aktivnu ( aktivna );
        aktivna->pocetak_kvanta = aktivna->dobiveno_vremena;
    }
}
```

9. [3 boda] Neki ugrađeni sustav sadrži sljedeće spremnike:
   a) ROM na adresi 0x100000
   b) RAM na adresi 0x200000
   c) priručni spremnik na adresi 0xF000000.

   Programska potpora sastoji se od jezgre OS-a (u direktoriju `jezgra`) te programa P_1, P_2, ... P_6 koji se nalaze u direktorijima `programi/p1`, `programi/p2`, `...programi/p6`. Slika sustava će se posebnim alatima spremiti u ROM. Pri pokretanju sustava posebnim sklopovljem sve će se kopirati u RAM. Potom će tek započeti s radom izgrađena programska komponenta. Jezgrine funkcije i podaci jezgre najprije će se kopirati u priručni spremnik (i tamo ostati i od tamo koristiti). Programi P_1, P_2 i P_3 pokretati će se iz RAM-a (za njega ih treba pripremiti), dok će ostali iz priručnog spremnika, kamo će se kopirati pri pokretanju. Programi P_4, P_5 i P_6 pokreću se pojedinačno – nikad nisu dva istovremeno aktivna (a i ne bi stalo više od jednog u priručni spremnik, uz jezgru). Dakle, programe P_4, P_5 i P_6 pripremiti za istu početnu adresu u priručnom spremniku (odmah iza jezgre). Sustav nema sklopovlje za dinamičko pretvaranje adresa – sve adrese moraju biti pripremljene za lokacije s kojih će se koristiti. Napisati skripte za povezivača.

Rj.

```
ROM = 0x100000; RAM = 0x200000; CACHE = 0xF000000;
SECTIONS {
    kernel_1st_load = RAM;
    kernel_copy_to = CACHE;
    .kernel kernel_copy_to : AT ( kernel_1st_load ) {
        jezgra* ( * )
    }
    p123_1st_load = RAM + SIZEOF ( .kernel );
    .p123 p123_1st_load : AT ( p123_1st_load ) {
        programi/p1* ( * )
        programi/p2* ( * )
        programi/p3* ( * )
    }
    p456_copy_to = kernel_copy_to + SIZEOF ( .kernel );
    p4_1st_load = p123_1st_load + SIZEOF ( .p123 );
    .p4 p456_copy_to : AT ( p4_1st_load ) {
        programi/p4* ( * )
    }
    p5_1st_load = p4_1st_load + SIZEOF ( .p4 );
    .p5 p456_copy_to : AT ( p5_1st_load ) {
        programi/p5* ( * )
    }
    p6_1st_load = p5_1st_load + SIZEOF ( .p5 );
    .p6 p456_copy_to : AT ( p6_1st_load ) {
        programi/p6* ( * )
    }
}
```

10. [3 boda] U sustavu koji koristi sklopovsku potporu za dinamičko upravljanje spremnikom, programi koriste logičke adrese dok u se u jezgrinim funkcijama koriste fizičke (apsolutne). Ostvariti jezgrinu funkciju `j_najveća` koja za zadana imena datoteka vraća ime i veličinu najveće datoteke (od zadanih imena). Neka je funkcija koja se poziva iz programa:

```
int najveća ( char **imena, size_t *veličina, char **najveća ) {
    izazovi_programski_prekid;
}
```

Iz obrade programskog prekida poziva se jezgrina funkcija: `j_najveća(void *parametri)` gdje je jedini parametar adresa (fizička) stoga pozivajuće dretve (adresa gdje se nalazi prvi parametar `imena`). Izvođenje jezgrine funkcije obavlja se korištenjem fizičkog načina adresiranja (pretvaranje adresa je isključeno). Uz pretpostavku da postoji pomoćna funkcija `vrati_veličinu(ime)` koja vraća veličinu zadane datoteke, ostvariti jezgrinu funkciju `j_najveća`. Zanemariti povratnu vrijednost funkcije (ono što `najveća` vraća kao povratnu vrijednost, ne preko parametara). Adresu početka spremničkog prostora trenutnog procesa može se dohvatiti sa `početna_adresa_procesa (NULL)`.

Rj.

```
int j_najveća ( void *parametri ) {
    char **imena;
    size_t *veličina;
    char **najveća;

    imena = *((char ***) parametri); parametri += sizeof (char **);
    veličina = *((size_t **) parametri); parametri += sizeof (size_t *);
    najveća = *((char ***) parametri);

    //adrese su u logičkom obliku, pretvori ih u apsolutni
    početna = početna_adresa_procesa ( NULL );
    imena += početna;
    veličina += početna;
    najveća += početna;

    max = -1;
    imax = 0;
    for ( i = 0; imena[i] != NULL; i++ ) {
        ime = imena[i] + početna; //pretvori u fiz. adresu
        vel = vrati_veličinu ( ime );
        if ( vel > max ) {
            imax = i;
            max = vel;
        }
    }
    *veličina = max;
    *najveća = imena[imax]; //ovo već je u logičkim adresama
}
```