Pisati čitko – nečitak odgovor ne donosi bodove.

1. (1) Navesti prednosti i nedostatke korištenja sustava za upravljanje izvornim kodom (npr. git).

   + praćenje promjena
   + grananje
   + višekorisnički rad
   - složenost (za naučiti)

2. (1) Koja svojstva algoritama dinamičkog upravljanja memorijom treba razmatrati u različitim okruženjima? Koja su bitnija u "običnim operacijskim sustavima", koja u ugrađenim sustavima s vrlo malo memorije, a koja u sustavima za rad u stvarnom vremenu?

   - obični OS: optimirati za prosječnu učinkovitost
   - ugr.sust: fragmentacija, dodatno korištenje memorije za upravljanje
   - SRSV: složenost

3. (1) Koja je prednost korištenja jedinstvenog sučelja za upravljanje napravama – sučelja koje za naprave treba implementirati da bi ih mogli ugraditi u operacijski sustav?

   - prenosivost
   - korištenje više naprava za istu stvar

4. (1) Sinkronizacijske funkcije (npr. semafori) osim uobičajenih sučelja *čekaj* i *postavi*, imaju i dodatna sučelja proširenih mogućnosti. Navesti ta sučelja te opisati njihove proširene mogućnosti.

   - trywait – ne blokiraj ako je semafor* neprolazan
   - timedwait – blokiraj, ali ne beskonačno nego najviše koliko je zadano

5. (1) Ostvarenje jezgrinih funkcija u sustavu koji ima izolaciju preko procesa zahtjeva nekoliko sklopovskih mogućnosti procesora i nekoliko postupaka izgradnje programske potpore. Opišite te potrebne mogućnosti i postupke.

   - potrebne sklopovske mogućnosti:
     * privilegirani način rada (može se sve)
     * korisnički način rada (ograničeni pristupi)
     * prekidi
     * podrška upravljanju spremnikom (segmenti, straničenje)
   - potrebni postupci
     * poziv jezgrinih funkcija mehanizmom prekida
     * zasebna priprema jezgre (npr. za apsolutne adrese)
     * zasebna programa (logičke adrese)

6. (1) Zašto sljedeći makroi nisu dobri? Pokazati primjere kada se ne dobiva željeno ponašanje.

```
#define INV(X)          (1/X)
#define ROOT(a,b,c)     ((-(b)-sqrt((b)*(b)-4*(a)*(c)))/(2*(a)))

a = INV(b+c) => a = 1/b+c
x1 = ROOT(a=funA(a,b,c), b=funB(a,b,c), c--) => a i b se više puta
   koriste u makrou!
```

7. (1) Napraviti makro `PORUKA(TIP,DULJINA,PODACI)` koji će zauzeti memoriju za novu poruku (funkcijom `malloc` koja postoji), popuniti ju zadanim vrijednostima i vratiti kazaljku na nju. Prva dva argumeta makroa su tipa `short`, a treći je adresa međuspremika s podacima. Poruka je definirana strukturom:

```
struct poruka {
```

```
    short tip;
    short duljina;
    char podaci[1];
}

#define PORUKA(TIP, PODACI, DULJINA)                          \
do {                                                          \
   struct poruka *poruka;                                     \
   poruka = malloc ( sizeof(struct poruka) - 1 + DULJINA );   \
   poruka->tip = TIP;                                         \
   poruka->duljina = DULJINA;                                 \
   memcpy ( poruka->podaci, PODACI, DULJINA );                \
   poruka;                                                    \
}                                                             \
while(0)
```

8. (1) Za zadani dio koda navesti gdje će se nalaziti pojedini dijelovi, u kojim odjeljcima (`.text`, `.rodata`, `.data`, `.bss`) ili na stogu.

```
int a = 3;                                      a: .data
static int b = 5;                               b: .data
char rez[10];                                 rez: .bss
void funkcija (int x, int y) {                x, y: stog
    static int z = 0;                           z: .data (.bss)
    char *c = "0123456789";                     c: stog
    ...                                 "0123456789": .rodata
}                                          funkcija: .text
```

9. (3) Neki ugrađeni sustav ima ROM na adresi 0x10000 i RAM na adresi 0x100000. Programe treba pripremiti za učitavanje u ROM, ali za korištenje iz RAM-a – pri pokretanju sve treba najprije kopirati u RAM i tek potom nastaviti s radom. Napisati skriptu za povezivača te funkciju `prekopiraj()` koja će se prva pozvati (staviti ju na početak ROM-a) i napraviti kopiranje te potom pozvati funkciju `pokretanje()` (koja, kao i sve ostalo, treba biti pripremljena za pokretanje iz RAM-a).

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

SECTIONS {
    .boot ROM :
    {
        prekopiraj.o (*)
    }
    pocetak = ROM + SIZEOF (.boot); //ili samo .
    .sve RAM : AT ( pocetak )
    {
        * (*)
    }
    kraj = pocetak + SIZEOF (.sve);
}

prekopiraj.c:
void prekopiraj() {
```

```
    extern char pocetak, kraj, RAM;
    char *s = &pocetak, *d = RAM;
    for (; s != &kraj;)
        *d++ = *s++;
    pokretanje();
}
```

10. (3) Neki procesor posjeduje sklop za prihvat prekida s N prekidnih ulaza koji se može programirati, tj. zasebno definirati prioritete za svaki pojedini ulaz. Postavljanje prioriteta za pojedini ulaz `I` svodi se na upisivanje prioriteta na adresu `INT_PRIO+I` (npr. za K-ti ulaz treba prioritet upisati na adresu `INT_PRIO+K`). Upisivanjem vrijednosti 0 sklop onemogućava se prekid s zadanog ulaza. Kada se na nekom od ulaza pojavi zahtjev za prekid i taj ulaz ima veći prioritet od broja trenutno zapisanog na adresi `INT_PRIO`, sklop prosljeđuje zahtjev za prekid prema procesoru i u `INT_PRIO` upisuje prioritet novog zahtjeva. Ako novi zahtjev nema veći prioritet sklop neće odmah proslijediti zahtjev za prekid. Kada su prekidi u procesoru omogućeni, procesor pri primitku zahtjeva za prekid sprema kontekst prekinutog programa na stog i poziva funkciju `prihvat_prekida`. Dozvola i zabrana prihvata prekida na razini procesora postiže se instrukcijama `dozvoli_prekidanje` i `zabrani_prekidanje`. Definirati potrebnu strukturu podataka te ostvariti sljedeće funkcije prekidnog podsustava:
`void inicijaliziraj_prekidni_podsustav ();`
`void zabrani_prekide_od_x (int x);` `void dozvoli_prekide_od_x (int x);`
`void registriraj_za_prekid (int x, void (*funkcija)(int), int prio);`
`void prihvat_prekida (int x);` (`x`=ulaz s kojeg je došao zahtjev za prekid).
Obrade prekida treba obavljati prema prioritetima – najprije obraditi prekide većih prioriteta.

```
#define N                (nešto-zadano)
#define INT_PRIO         (nešto-zadano)

void (*obrada[N])(int);      //funkcije za obradu
int *PRIO = (int*) INT_PRIO; //lakše korištenje

void inicijaliziraj_prekidni_podsustav () {
    int i;
    for ( i = 0; i < N; i++ ) {
        obrada[i] = NULL;
        PRIO[i] = 0;
    }
    PRIO[N] = 0;
    dozvoli_prekidanje;
}
void zabrani_prekide_od_x (int x) {
    PRIO[x] = 0;
}
void registriraj_za_prekid (int x, void (*funkcija)(int), int prio) {
    PRIO[x] = prio;
    obrada[x-1] = funkcija;
}
void prihvat_prekida (int x) {
    int tp = PRIO[0]; //zapamti tekući prioritet

    dozvoli_prekidanje;
    obrada[x-1](x);
    zabrani_prekidanje;
    PRIO[0] = tp;
```

```
}
```

11. (3) U nekom sustavu satni mehanizam generira periodičke zahtjeve za prekid svakih 500 mikrosekundi. Ostvariti podsustav za upravljanje vremenom korištenjem samo tog prekida. Jedinica vremena za sva sučelja treba biti milisekunda. Potrebne funkcionalnosti ostvariti kroz sučelja:
`unsigned long int dohvati_sat ();`
`void postavi_sat (unsigned long int t);`
`void alarm (void (*funkcija_za_obradu)(void *), unsigned long int t);.`

```
unsigned long int sat = 0; //sat u jedinicama od 500 mikrosekundi
unsigned long int odgoda = 0;

unsigned long int dohvati_sat () {
    return sat / 2;
}
void postavi_sat (unsigned long int t) {
    sat = t * 2;
}
void alarm (void (*funkcija_za_obradu)(void *), unsigned long int t) {
    odgoda = t * 2;
    obrada = funkcija_za_obradu;
}
void prekid_sata () {
    sat++;
    if (odgoda > 0) {
        odgoda--;
        if ( odgoda <= 0 )
            obrada(NULL);
    }
}
```

12. (2) U nekom sustavu pri pozivu bilo koje jezgrine funkcije prvo se poziva `kthread_pause()`, a neposredno prije povratka iz jezgrine funkcije poziva se `kthread_resume()`. Na poseban alarm koji se poziva svakih `Tq` poziva se i funkcija `kscheduler_tick()`. Definirati potrebnu strukturu podataka (promjene u opisniku dretve) te ostvariti navedene tri funkcije tako da se ostvari raspoređivanje podjelom vremena s kvatnom vremena `Tq`. Dretva treba dobiti najmanje toliko vremena prije nego li će je funkcija `kscheduler_tick()` zamijeniti. Pretpostaviti da za upravljanje dretvama postoje funkcije:
`kthread_t *kthread_get_active_thread ();`
`void kthread_set_active_thread (kthread_t *thread);`
`void kthread_move_to_ready (kthread_t *thread);`
`kthread_t *kthread_get_first_ready ();`
Dohvat trenutnog sata u mikrosekundama obaviti s `long long kget_clock()`.

```
dodati u opisnik kthread_t: long long t_start, t_got;

void kthread_pause() {
    kthread_t *thread = kthread_get_active_thread ();
    thread->t_got += kget_clock() - thread->t_start;
}
void kthread_resume() {
    kthread_t *thread = kthread_get_active_thread ();
    thread->t_start = kget_clock();
}
void kscheduler_tick() {
```

```
    kthread_t *thread = kthread_get_active_thread ();
    if ( thread->t_got >= Tq ) {
        thread->t_got -= Tq;
        kthread_move_to_ready ( thread );
        thread = kthread_get_first_ready();
        kthread_set_active_thread (thread);
    }
}
```

13. (1) Neku jezgru koja je napravljena da radi u sustavu bez zaštite spremnika, mehanizmom procesa treba prilagoditi za takvu zaštitu. Stoga se sve jezgrine funkcije trebaju provjeriti i po potrebi ažurirati. Parametri koje jezgrine funkcije dobivaju su kopije parametara koje je dretva procesa stavila u poziv. U procesima se koriste logičke adrese (svaki proces kreće od adrese 0), a u jezgrinim funkcijama absolutne (fizičke). Početna adresa aktivna procesa (njegova absolutna adresa) u jezgrinim funkcijama se može dohvatiti preko funkcije `kget_process_start()`. Prilagoditi sljedeću jezgrinu funkciju da ispravno radi u tom promijenjenom okruženju.

```
void sys__kernel_funtion ( int uid, char *buffer, int size ) {
    int id = kfind_object ( uid );
    if ( id > 0 ) {
        char *obj = kobj_get_base (id);
        memcpy ( obj, buffer, size );
    }
}
```

```
void sys__kernel_funtion ( int uid, char *buffer, int size ) {
    int id = kfind_object ( uid );
    if ( id > 0 ) {
        char *obj = kobj_get_base (id);
        memcpy ( obj, buffer + kget_process_start(), size );
    }
}
```