

1. (4) Programska komponenta priprema se za sustav koji ima ROM na adresi 0x10000 te RAM na adresi 0xA0000. Napisati skriptu za povezivanje koja treba generirati sliku sustava koja će se posebnim alatom učitati u ROM. Sve što može ostati u ROM-u neka tamo trajno ostane i od tuda koristi, a sve ostalo pripremiti za korištenje iz RAM-a. Napisati funkciju *premjesti()* koja se poziva odmah pri pokretanju sustava a koja kopira potrebne dijelove iz ROM-a u RAM. Po potrebi koristiti i postojeću funkciju `void memcpy(void *dest, void *src, size_t size)`.

Rješenje:

<pre>ROM = 0x10000 RAM = 0xA0000 SECTIONS { .text ROM : { * (.text) * (.rodata) } text_size = SIZEOF(.text); data_in_rom = .; .data RAM : AT(ROM + text_size) { * (.data) * (.bss) } data_size = SIZEOF(.data); }</pre>	<pre>void premjesti() { extern char RAM, data_in_rom, data_size; memcpy(&RAM, &data_in_rom, &data_size); }</pre>
---	--

2. (4) Neki sustav ima sklop za prihvrat prekida s 16 ulaza na koje se spajaju naprave kada traže prekid. Sklop ima sljedeće registre: 16-bitovni upravljački registar (UR) na adresi 0xFB00, 8-bitovni registar prioriteta (RP) na adresi 0xFB04 te 8-bitovni registar trenutna prioriteta (TP) na adresi 0xFB08. Svaki bit upravljačkog registra označava da li se razmatra ili ne ekvivalentni ulaz – kada je bit obrisan (nula) zahtjev za prekid dotične naprave se ignorira, u protivnom se prihvaća. Svaki ulaz ima dodijeljeni prioritet koji odgovara rednom broju ulaza: ulaz 0 ima prioritet 0 (najmanji prioritet), ulaz 15 prioritet 15 (najveći prioritet). U registru RP nalazi se prioritet najprioritetnijeg zahtjeva koji čeka na obradu ili broj 0xFF kada nema niti jednog zahtjeva. Ako je vrijednost u RP veća od one u TP, sklop šalje zahtjev za prekid prema procesoru. Pri prihvatu prekida procesor treba upisati prioritet trenutne obrade u TP. Sklop će tada obrisati taj zahtjev za prekid u svojim internim registrima i po potrebi ažurirati registar RP. Ostvariti podsustav za upravljanje prekidima (u C-u) sa sučeljima za inicijalizaciju podsustava, funkciju za registraciju funkcije za obradu prekida te funkciju koja se poziva na svaki prekid (a koja mora utvrditi prioritet prekida i pozvati odgovarajuću registriranu funkciju). Započetu obradu prekida ne prekidati novim zahtjevima, makar i većeg prioriteta (obradu obaviti sa zabranjenim prekidanjem). Onemogućiti ulaze za koje ne postoji funkcija za obradu (tako da se takvi zahtjevi ne prosljeđuju procesoru).

Rješenje:

<pre>short int *UR = (short int *) 0xFB00; unsigned char *RP = (unsigned char *) 0xFB04; unsigned char *TP = (unsigned char *) 0xFB08; void (*obrada[16])(); void inicijaliziraj_prekide() { int i; for (i = 0; i < 16; i++) obrada[i] = NULL; *TP = 0; *UR = 0; }</pre>	<pre>void registriraj(int prio, void *funkcija) { obrada[prio] = funkcija; if (funkcija != NULL) *UR = *UR (1<<prio); else *UR = *UR & ~(1<<prio); } void prekidna_rutina() //vidi komentar ispod { int prio = *RP; *TP = prio; obrada[prio](); *TP = 0; }</pre>
--	---

Samo zahtjevi onih naprava za koje su postavljene funkcije mogu se prosljediti dalje (ovdje), pa nije potrebno ispitivati da li je ta funkcija postavljena. Zabrana prekida i dozvola nije potrebna jer procesor pri prihvatu prekida automatski zabrani daljnje prekidanje, a pri povratku iz prekida dozvoli ponovno prekidanje.

3. (4) Neki sustav ima brojilo na adresi BROJILO koje odbrojava frekvencijom $f=800$ kHz od učitane vrijednosti do nule kada izaziva prekid te postavlja najveću vrijednost u brojilo (0xFFFF) i nastavlja s odbrojavanjem (ciklus ima najviše 0xFFFF otkucaja). Korištenjem navedena brojila ostvariti podsustav za upravljanje vremenom koji se sastoji od sata izraženog u mikrosekundama (tip long u C-u), te jednog alarma sa sučeljima:

```
void inicijaliziraj();
void dohvati_sat (long *t);
void postavi_sat (long *t);
void postavi_alarm (long *kada, void (*alarm)());
void prekid_sata();
```

Argument kada je apsolutno vrijeme izraženo u mikrosekundama. Pri promjeni sata funkcijom postavi_sat, obrisati alarm bez njegove aktivacije. Za pretvorbu broja otkucaja u vrijeme i obratno napraviti makroe CNT2USEC(CNT) i USEC2CNT(USEC). Pretpostaviti da vrijeme izraženo u mikrosekundama neće preći najveću vrijednost koja stane u tip long (tj. vrijednost MAXLONG).

Rješenje:

<pre>#define MAXCNT 0xFFFF #define FREQ 800000 //ne koristi se #define CNT2USEC(CNT) ((CNT)*5/4) //kom.isp. #define USEC2CNT(USEC) ((USEC)*4/5) static long sat; static void (*obrada_alarma)(); static long t_akt; static unsigned short int ucitano; static unsigned short int *brojilo = BROJILO; static void podesi_brojilo() { if (t_akt < MAXLONG) { long za_koliko = USEC2CNT(t_akt - sat); if (za_koliko > 0 && za_koliko < MAXCNT) ucitano = za_koliko; } else { ucitano = MAXCNT; } *brojilo = ucitano; } void inicijaliziraj() { sat = 0; obrada_alarma = NULL; t_akt = MAXLONG; podesi_brojilo(); }</pre>	<pre>void dohvati_sat(long *t) { *t = sat + CNT2USEC(ucitano - *brojilo); //Ovdje ne ažurirati sat! Jer na slijedeći prekid će se opet isto //vrijeme nadodati na sat. } void postavi_sat(long *t) { inicijaliziraj(); sat = *t; } void postavi_alarm (long *kada, void *alarm) { sat += CNT2USEC(ucitano - *brojilo); t_akt = kada; obrada_alarma = alarm; podesi_brojilo(); } void prekid_sata () { sat += CNT2USEC(ucitano); if (t_akt <= sat) { t_akt = MAXLONG; podesi_brojilo(); obrada_alarma(); } else { podesi_brojilo(); } }</pre>
--	--

Ostaviti „*1000000/800000“ nije dobro jer se prvim množenjem može izaći iz opsega broja tipa long (najčešće 32 bita). Obzirom da su brojke okrugle to se lako pokradi i ostane 5/4.

CNT je cijeli broj i množiti ovo realnim brojem 1,25 nije preporučeno, pogotovo što je očito ovo program za neki mikrokontroler, a oni gotovo uvijek nemaju operacije s realnim brojevima.

Nadalje, bitno je prvo pomnožiti s 5 a onda podijeliti s 4. Obrnutim redoslijedom se gubi na preciznosti.

4. (4) Računalo nekom udaljenom uređaju šalje podatke preko posebnog sklopa sljedećim protokolom: prvo se pošalje bajt 0xF0 koji označava početak poruke; slijedi bajt koji označava duljinu korisne informacije koja se želi poslati; potom se šalje bajt po bajt informacije te na kraju se još pošalje bajt 0x0F. Sklop koji se koristi za slanje ima dva registra: registar podatka RP preko kojeg se šalju podaci te registar stanja (RS). Prvi bit registra stanja (jedini bit koji se koristi u tom registru) označava je li sklop spreman prihvatiti sljedeći bajt za slanje. Pri pisanju u RP se taj bit automatski obriše dok sklop ne bude spreman za prihvrat novog bajta, kada ga sklop ponovno postavlja. Svaki puta kada se taj bit u postavi u 1 sklop izaziva zahtjev za prekid na koji se poziva funkcija obrada_prekida_x() (koju treba ostvariti). Ostvariti upravljanje navedenim sklopom uz poštivanje opisanog protokola kroz funkciju:

```
int pošalji(void *informacija, char veličina);
```

gdje informacija sadrži informaciju koju treba poslati (bez dodatnih bajtova protokola) te veličina njenu veličinu. Funkcija treba raditi asinkrono, tj. postaviti zadanu poruku za slanje, eventualno odraditi i početno slanje, ali se ostatak slanja treba ostvariti u funkciji za obradu prekida. Funkcija pošalji treba vratiti nulu kad je zadata slanje. Ako se pošalji pozove dok prethodna operacija nije gotova poziv treba vratiti -1 i taj se novi zahtjev ignorira. Ostvariti funkcije inicijaliziraj, pošalji i obrada_prekida_x (uz potrebnu dodatnu strukturu podataka za privremenu pohranu poruke).

Rješenje:

```
char duljina, sljedeći;
char bufer[127]; //može i 255; problem je u „char veličina“ {-128 do 127}
int status; // 0 - ništa, 1 - slanje u tijeku

void inicijaliziraj ()
{
    status = 0;
}

int pošalji (void *informacija, char veličina)
{
    if (status != 0)
        return -1;
    status = 1;
    duljina = veličina + 3;
    sljedeći = 0;
    bufer[0] = 0xF0;
    bufer[1] = veličina;
    memcpy(bufer + 2, informacija, veličina);
    bufer[duljina - 1] = 0x0F;
    obrada_prekida_x();
    return 0;
}

void obrada_prekida_x()
{
    if (status == 1) { //slanje
        while (*RS && sljedeći < duljina) //ili samo: if (*RS)
            *RP = bufer[sljedeći++]; // *RP = bufer[sljedeći++];
        if (sljedeći == duljina)
            status = 0; //slanje je gotovo
    }
}
```

5. Osmisliti i ostvariti raspoređivanje dretvi u jednostavnom sustavu koji koristi prioritarno raspoređivanje s 4 različita prioriteta (0-3). Definirati opisnik dretve sa `struct dretva` u koji treba dodati samo potrebne elemente za raspoređivanje te element `struct ostalo sve_ostalo`. Osmisliti strukturu podataka potrebnu za raspoređivanje te ostvariti funkcije:

```
void inicijaliziraj_raspoređivač();
void dodaj_u_pripravne(struct dretva *dretva);
void raspoređivanje();
```

Funkcija `dodaj_u_pripravne` se poziva kad se stvori nova dretva ili postojeća postaje pripravna (prethodno je bila blokirana), a `void raspoređivanje()` se poziva prije povratka u aktivnu dretvu, a čija je zadaća u globalnu varijablu `struct dretva *aktivna` postaviti kazaljku na aktivnu dretvu (odabrati najprioritetniju među pripravnim, uključujući i trenutno aktivnu) te po potrebi pozvati funkciju za zamjenu konteksta:

```
void zamijeni_kontekst(struct dretva *stara_aktivna, struct dretva *nova_aktivna);
```

koja postoji, dok argument `stara_aktivna` može biti i `NULL`. Pretpostaviti da će nakon inicijalizacije jezgra stvoriti bar jednu dretvu, dodati ju u red pripravnih i pozvati `raspoređivanje` te da će u nastavku rada u sustavu uvijek postojati barem jedna dretva spremna za izvođenje (npr. pored ostalih postoji i *idle* dretva koja se nikada ne blokira).

Rješenje:

```
struct dretva {
    int stanje; //0 nije pripravna, 1 je
    int prio;
    struct dretva *iduća;
    struct ostalo sve_ostalo;
}

struct dretva *aktivna;
#define BRPR 4
struct pripravne {
    struct dretva *prva, *zadnja;
}
pripravne[BRPR];
```

```
void inicijaliziraj_raspoređivač()
{
    aktivna = NULL;
    for (int i = 0; i < BRPR; i++)
        pripravne[i].prva = pripravne[i].zadnja = NULL;
}
```

```
void dodaj_u_pripravne(struct dretva *dretva)
{
    int i = dretva->prio;
    if (pripravne[i].prva)
        pripravne[i].zadnja->iduća = dretva;
    else
        pripravne[i].prva = dretva;
    pripravne[i].zadnja = dretva;
    dretva->iduća = NULL;
    dretva->stanje = 1;
}
```

```
void raspoređivanje()
{
    struct dretva *stara = aktivna, *prva = NULL;
    for (int i = BRPR - 1; i >= 0 && !prva; i--)
        if (pripravne[i].prva)
            prva = pripravne[i].prva;
    if (stara == NULL || stara->stanje == 0 || (prva != NULL && prva->prio > stara->prio)) {
        aktivna = prva;
        pripravne[prva->prio].prva = prva->iduća; //makni novu aktivnu iz reda pripravnih
        if (stara && stara->stanje)
            dodaj_u_pripravne(stara);

        zamijeni_kontekst(stara, aktivna);
    }
}
```

=> Moramo znati ovo za micanje aktivne dretve; da li je samo maknemo iz kazaljka aktivna, ili ju moramo i staviti u red pripravnih. Ako je prethodno ta dretva blokirana, kazaljka aktivna i dalje treba pokazivati na nju da bi u raspoređivanje mogli spremati kontekst te dretve.

Red za pripravne može biti i jednostavna lista, uređena prema prioritetima. Tada je ubacivanje u listu linearne složenosti, ali u zadatku se nije tražilo da operacije budu konstantne složenosti.

Ovo treba biti zadnja radnja u ovoj funkciji ako se mijenja aktivna dretva

6. U sustavu koji koristi upravljanje memorijom na način da se u procesima koriste logičke adrese (npr. straničenjem) treba ostvariti jezgrinu funkciju `int sys_povećaj_buffer(void *stari, size_t veličina_prije, void **novi, size_t potrebna_veličina)`. Argument `stari` pokazuje na već dodijeljeni blok, argument `novi` pokazuje na varijablu u koju treba spremiti adresu novog povećanog bloka. Obje adrese su iskazane u logičkoj adresi koja se koristi unutar procesa. Funkcije koje se mogu koristiti (postoje unutar jezgre) su:

```
kprocess_t *dohvati_aktivni_proces();  
void *pretvori_u_fizičku_adresu (kprocess_t *proces, void *logička_adresa)  
void *pretvori_u_logičku_adresu (kprocess_t * proces, void *fizička_adresa)  
void *zauzmi_blok_za_proces (kprocess_t *proces, size_t size); //vraća fizičku adresu  
void oslobodi_blok_u_procesu (kprocess_t *proces, void *mem); //mem je fizička adresa  
void memcpy (void *dest, void *src, size_t size); //obje adrese su fizička
```

Jezgrine funkcije pozivaju se mehanizmom prekida – funkciju `sys_povećaj_buffer` ne poziva dretva izravno već preko prekida. Unutar jezgrinih funkcija koriste se fizičke adrese.

Rješenje:

```
int sys_povećaj_buffer (void *stari, size_t veličina_prije, void **novi, size_t potrebna_veličina)  
{  
    void *fa_stari, **fa_novi, *novi_blok;  
  
    kproces *proces = dohvati_aktivni_proces();  
    fa_stari = pretvori_u_fizičku_adresu (proces, stari);  
  
    novi_blok = zauzmi_blok_za_proces(proces, potrebna_veličina);  
    memcpy(novi_blok, fa_stari, veličina_prije);  
    oslobodi_blok_u_procesu(proces, fa_stari);  
  
    fa_novi = pretvori_u_fizičku_adresu (proces, novi); //gdje staviti adresu novog bloka  
    *fa_novi = pretvori_u_logičku_adresu(proces, novi_blok); //procesu vraćamo logičku adresu bloka  
  
    return 0;  
}
```