Operacijski sustavi za ugrađena računala Završni ispit, 27. 6. 2024.

Zadatke rješavati na ovom obrascu. Nečitki odgovori se neće ispravljati – krivi su.

1. (1) Treba li (i ako da kako) dodatno označiti/* varijable:
   - varijabla treba biti vidljiva samo unutar datoteke: **`static`** `int x`
   - sadrži adresu registra pristupnog sklopa naprave: **`volatile`** `int *a`
   - pri prevođenju treba ju smjestiti u odjeljak `BOOT`: `int b` **`__attribute__((section("BOOT")))`**
   - Varijablu definiranu u `a.c` sa `int var_iz_a=77;` želimo koristiti u `b.c`. Kako to napraviti?

     `u b.c treba prije korištenja napisati extern int var_iz_a;`
   - U zaglavlju `z.h` nalazi se i kratka funkcija `int kf()` koja se koristi i iz drugih datoteka. Treba li ju posebno označiti? Ako da kako?

     `static inline int kf()`

2. (1) Za navedene dijelove programa označiti u koje će ih odjeljke postaviti prevoditelj pri prevođenju (.c => .o).

   `int a, b=5, c[55];`

   `b=>data, a i c => bss`

   `char *d = "1234";`

   `d => data, "1234" u rodata`

   `static int inc(int *x, int *y) { (*x)++; (*y) += *x; return *y; }`

   `kod u .txt`

   `void ispisi() { printf("a=%d, b=%d, d=%s\n, a, b, d); }`

   `kod u .txt, "a=..." u rodata`

3. (1) Napisati makro `ZBROJI(R, A, B)` koji zbraja dva kompleksna broja (`A` i `B`) te rezultat sprema u `R`. Svi argumenti su tipa `kompleksni_t` koji ima polja `re`, i `im`. Obzirom da argumenti mogu biti složeni izrazi, u makrou se smiju pojaviti samo jednom.

```
#define ZBROJI(R, A, B)                \
do {                                   \
    kompleksni_t a = (A), b = (B), c;  \
    c.re = a.re + b.re;                \
    c.im = a.im + b.im;                \
    (R) = c;                           \
} while(0)
```

4. (2) Za neki ugradbeni sustav treba pripremiti sliku za učitavanje. Sustav ima ROM veličine 64 KB na adresi `ROM_ADR` te RAM veličine 32 KB na adresi `RAM_ADR`. Program za upisivanje slike u ROM u slici očekuje odjeljke: `.init`, `.loop`, `.const`, `.data`. Pretpostaviti da je izvorni kod već označen – funkcije su označene za koji odjeljak ih treba pripremiti. Podaci nisu označeni, ali konstante pripremiti za `const`, varijable za `data`, definirati varijablu `heap_start` koja će imat adresu iza svih varijabli te varijablu `stack_end` koja treba imati adresu zadnjeg bajta u RAM-u. Napisati skriptu za povezivanje tako da se instrukcije i konstante spremaju za ROM, a varijable za korištenja iz RAM-a (ali inicijalno u ROM-u).

```
SECTIONS {
    init_start = ROM_ADR;
    .init init_start : AT (init_start) {
          * (.init)
    }
    loop_start = init_start + SIZEOF(.init);
    .loop loop_start : AT (loop_start) {
          * (.loop)
    }
    const_start = loop_start + SIZEOF(.loop);
    .const const_start : AT (const_start) {
          * (.rodata)
    }
    data_start = const_start + SIZEOF(.const);
    .data RAM_ADR : AT (data_start) {
          * (.data .bss)
    }
    heap_start = data_start + SIZEOF(.data);
    stack_end = RAM_ADR + 32*1024 - 1;
}
```

5. Neki sustav nema sklop za prihvat prekida – svi zahtjevi za prekid izravno dolaze do zajedničkog prekidnog ulaza procesora. Svi pristupni sklopovi koji mogu slati zahtjev za prekid imaju četiri 8-bitovna registra (u uzastopnim adresama): `ULAZ`, `IZLAZ`, `SR`, `UPR`. Prva dva služe za prijenos podataka. Registar `SR` pokazuje stanje sklopa: bit 0 označava prisutnost nepročitanog podatka u `ULAZ`, bit 1 dostupnost registra `IZLAZ` za prihvat novog podatka za slanje. Registar `UPR` je upravljački: bit 0 označava treba li sklop generirati zahtjev za prekid na novi podatak te bit 1 treba li sklop generirati zahtjev za prekid kad se podatak pošalje i kad sklop od procesora može prihvatiti novi podatak za slanje. U sustavu ima ukupno osam naprava. Prva je na adresi `0xFF0000` (registar `ULAZ`), druga za četiri mjesta iza (`ULAZ` za drugu je na `0xFF0004`), itd.
   a) (2) Ostvariti prekidni podsustav s funkcijama `void init()`, `void registriraj_obradu(char id, void (*fja_obrade)(char))`, gdje su argumenti redni broj naprave i funkcija za obradu, te `prihvat_prekida()` koja se poziva na svaki prekid. Napravama za koje nije registrirana funkcija (`NULL`) zabraniti generiranje prekida.

```
void (*obrada[8])();
#define ADR(I)              (0xFF0000 + (I)*4)
#define POSTAVI_UPR(I, X)  do *((char *)(ADR(I) + 3)) = X; while(0)
#define DOHVATI_SR(I)      (*((char *)(ADR(I) + 2)))
void init() {
   int i;
   for (i = 0; i < 8; i++) {
      obrada[i] = NULL;
      POSTAVI_UPR(i, 0)
   }
}
void registriraj_obradu(int naprava, void *fja_obrade) {
   obrada[naprava] = fja_obrade;
   POSTAVI_UPR(naprava, 3)
}
void prihvat_prekida() {
   int i;
   for (i = 0; i < 8; i++) {
      if (DOHVATI_SR(i))
         obrada[i](i);
   }
}
```

   b) (1) Ostvariti (generički) upravljački program za upravljanje napravama funkcijama `void init()` (po potrebi), `char read(char id)` (čita samo jedan znak), `char write(char id, char data)`. Ako nema podataka pri čitanju, vratiti -1. Ako se podatak ne može poslati, vratiti -1. *Ovaj dio zadatka je neovisan od a) – ne treba koristiti prekide.*

```
#define ADR(I)               (0xFF0000 + (I)*4)
#define DOHVATI_SR(I)        (*((char *)(ADR(I) + 2)))
#define DOHVATI_ULAZ(I)      (*((char *)(ADR(I))))
#define POSTAVI_IZLAZ(I, X)  do *((char *)(ADR(I) + 1)) = X; while(0)

char read(char id) {
   if (DOHVATI_SR(id) & 1 == 0)
      return -1;
   return DOHVATI_ULAZ(id);
}
char write(char id, char data) {
   if (DOHVATI_SR(id) & 2 == 0)
      return -1;
   POSTAVI_IZLAZ(id, data);
   return 1;
}
```

6. (2) Neki sustav ima 10-bitovno brojilo na adresi `BROJILO` koje odbrojava frekvencijom f = 1 MHz. Pomoću njega treba ostvariti upravljanje koje:

   a) svake 1 ms poziva funkciju `obrada_1000()`

   b) svakih 347 μs poziva funkciju `obrada_347()`

   Kašnjenje neke obrade zbog završetka druge nije problem, ali ne zanemariti to vrijeme.

   Kada je iduća „aktivacija“ bliže od 5 μs koristiti radno čekanje. U protivnom koristiti prekid brojila (i dopustiti izvođenje drugih programa u međuvremenu). Napisati funkcije `init()` te `prekid_brojila()` (pretpostaviti da je već registrirana za taj prekid). Rješenje može uključivati „podsustav za upravljanje vremenom“, ali i ne mora, može biti samo rješenje za navedeni problem.

```
//s relativnim odgodama
t347  = 347 //za koliko idući put akt.
t1000 = 1000
učitano = 0
MAX = 0x3ff

init() {
   BROJILO = učitano = t347
}

ažuriraj(oduzmi) {
   t1000 -= oduzmi
   t347 -= oduzmi
   BROJILO = učitano = MAX
}

prekid_brojila() {
   ponavljaj {
      ažuriraj(učitano)
      if (t1000 <= 0) {
         t1000 += 1000
         obrada_1000()
         ažuriraj(učitano - BROJILO)
      }
      if (t347 <= 0) {
         t347 += 347
         obrada_347()
         ažuriraj(učitano - BROJILO)
      }
      učitano = min(t347, t1000)
      if (učitano > 5) {
         BROJILO = učitano;
         break;
      }
      while (MAX - BROJILO < učitano)
         ; //radno čekanje
   }
}
```

```
//s apsolutnim odgodama
t = 0 //"sat"
t347  = 347 //kada idući put aktivirati
t1000 = 1000
učitano = 0
MAX = 0x3ff

init() {
   BROJILO = učitano = t347
}

ažuriraj(dodaj) {
   t += dodaj
   BROJILO = učitano = MAX
}

prekid_brojila() {
   ponavljaj {
      ažuriraj(ucitano)
      ako je t347 <= t onda {
         t347 += 347
         obrada_347()
         ažuriraj(MAX - BROJILO)
      }
      ako je t1000 <= t onda {
         t1000 += 1000
         obrada_1000()
         ažuriraj(MAX - BROJILO)
      }
      učitano = min(t347, t1000) - t
      if (učitano > 5) {
         BROJILO = učitano;
         break;
      }
      while (MAX - BROJILO < učitano)
         ; //radno čekanje
   }
}
```

7. (2) U nekom jednostavnom višedretvenom operacijskom sustavu dodati mehanizam barijere kroz sučelja: `void binit(void *b, int ndretvi)`, `int bčekaj(void *b)`. Mehanizam barijere treba zaustaviti dretve na `bčekaj` dok `ndretvi` ne pozove tu funkciju, u kojem slučaju propustiti sve dretve te resetirati barijeru. Pretpostaviti da postoje sve potrebne interne funkcije za upravljanje dretvama (`dretva_t, red_t, dretva_t *aktivna(), void stavi_u_red(dretva_t *, red_t *)`, `void stavi_u_pripravne(dretva_t *)`, `void raspoređivač(), kmalloc, ...`). Ostvariti samo interne funkcije `k_binit` i `k_bčekaj` (i potrebne strukture podataka) – pretpostaviti da se tu očekuje samo glavni dio posla (prekidi su već zabranjeni, kontekst spremljen, ...). Pri inicijalizaciji, adresu interna objekta barijere vratiti preko `b` (bez maskiranja, ...). Ne treba provjeravati ispravnost argumenata i povratnih vrijednosti internih funkcija (sve rade dobro).

```
typedef kbarijera {
    int ndretvi;
    int brojac;
    red_t red;
}
kbarijera_t;

int k_binit(void *b, int ndretvi) {
    kbarijera_t *bar = kmalloc(sizeof(kbarijera_t));

    bar->ndretvi = ndretvi;
    bar->brojac = 0;
    k_inicijaliziraj_red(&bar->red);

    return 0;
}

int k_bčekaj(void *b) {
    kbarijera_t *bar = b;

    bar->brojac++;
    if (bar->brojac < bar->ndretvi) {
          stavi_u_red(&bar->red, aktivna());
    }
    else {
          bar->brojac = 0;
          oslobodi_sve_iz_reda(&bar->red);
    }
    raspoređivač();
    return 0;
}
```

8. (1) Navesti bar sedam različitih elemenata opisnika dretvi (npr. signalna maska i funkcije za obradu signala spadaju u jedan element, osmi).

id – identifikacijski broj dretve

stog, veličina stoga

način raspoređivanja i parametri

stanje dretve (aktivno, pripravno, pasivno, susp, ...)

red u kojem se nalazi

red za dretve koje čekaju njen kraj

opisnik konteksta (gdje je spremljeno kontekst dretve)

izlazni status (broj i errno)

privatni parametar

lista funkcija koje pozvati pri završetku

funkcija koju pozvati ako se odblokira „preuranjeno“

elementi za smještaj opisnika u razne liste

9. (1) Za ostvarenje procesa potrebno je koristiti a) privilegirani i neprivilegirani način rada procesora, b) programski prekid za poziv jezgrine funkcije iz programa, c) zasebno pripremanje slike jezgre i procesa, d) sklop za upravljanje memorijom. Opisati što se postiže svakim od navedenih mehanizama.

   a) iz programa se ne mogu pokretati privilegirane instrukcije koje mogu narušiti sustav (npr. zabrana prekida)

   b) zbog a) jezgrine funkcije moraju se pozvati mehanizmom prekida da se uđe u privilegirani način rada procesora

   c) procesi se izvode u logičkom adresnom prostoru, koriste drukčije adrese od jezgre, moraju se zasebno pripremiti

   d) radi izolacije procesa od jezgre

10. (1) Ostvariti jezgrinu funkciju `int k_malloc_ex(size_t *req, int type, void *adr)` koja od jezgre traži zauzimanje dinamičke memorije za proces. Argument `req` sadrži adresu varijable u kojoj se nalazi potrebna veličina, ali i preko koje će se vratiti veličina zauzete memorije. Argument `type` definira poželjna svojstva memorije. Preko argumenta `adr` treba vratiti adresu zauzeta bloka. Pretpostaviti da postoji identična interna jezgrina funkcija `kmalloc_ex`, ali koja radi s apsolutnim adresama, dok su dobiveni argumenti (adrese samo) u logičkom adresnom prostoru procesa.

```
int k_malloc_ex(size_t *req, int type, void **adr)
{
    size_t *kreq = U2K(req, aktivna()); //kamo zapisati dobivenu veličinu
    void *tmp; //kamo zapisati dobivenu adresu u apsolutnim adresama

    int status = kmalloc_ex(kreq, type, &tmp);

    void **kadr = U2K(adr, aktivna()); //kamo zapisati dobivenu adresu u log. adr.
    *kadr = K2U(tmp, aktivna()); //pretvori dobivenu adresu u log. i zapiši u adr

    return status;
}
//U2K => logička u fizičku (user to kernel)
//K2U => fizička u logičku
```