Operacijski sustavi za ugrađena računala

15. 6. 2023. Završni ispit iz predmeta

ime i prezime

Pisati ČITKO, nečitka rješenja se ne ocjenjuju.

1. (1) Napisati makro koji će ovisno o predznaku prvog argumenta (≥0, <0) vratiti vrijednost drugog argumenta povećanog ili smanjenog za jedan. Primjer poziva: `a = AZURIRAJ(x+12*y, -z/w)`

```
#define AZURIRAJ(A, B)    ((A) >= 0 ? (B) + 1 : (B) - 1 )
```

može se shvatiti i ovako:

```
#define AZURIRAJ(A, B)    ((A) >= 0 ? (B)++ : (B)-- )
```

Neki su studenti napisali:

```
    do if((A) >= 0) return (B) + 1 else return (B) - 1; while(0)
```

što je KRIVO! Makro nije funkcija!

2. (1) Napisati makro koji će zamijeniti vrijednost varijablama poslanim kao argumentima. Argumenti su nekog osnovnog tipa (char, int, float, double). Primjer poziva: `ZAMIJENI(x[3], y)`

```
#define ZAMIJENI(A, B)    do {                          \
                              typeof(A) _a_ = (A);      \
                              (A) = (B);                \
                              (B) = _a_;                \
                          } while(0)
```

3. (1) Ažurirati zadani `Makefile` da se u njega uključi dodatna datoteka s izvornim kodom `dodatak.c` koja se nalazi u početnom direktoriju (uz `main.c` i `osnovno.c`) te da se uključi i proširenje iz direktorija `prosirenje` koje sadrži zasebni `Makefile` koji će generirati objekt `prosirenje/prosirenje.o`.

```
PROGRAM = program
OBJEKTI = main.o osnovno.o dodatak.o
MODULI = op1/op1.o op2/op2.o prosirenje/prosirenje.o
$(PROGRAM): $(OBJEKTI) $(MODULI)
    $(CC) $(LDFLAGS) $(OBJEKTI) $(MODULI) $(LDLIBS) -o $(PROGRAM)
.PHONY: $(MODULI)
$(MODULI):
    $(MAKE) -C $(dir $@)
```

4. (4) Izvorni kod za neki sustav podijeljen je u direktorije: `bootloader` (program pokretač), `kernel` (jezgra), `p1`, `p2` i `p3` (programi). U svakom direktoriju je `Makefile` koji kaže kako prevesti izvorne kodove u zasebne slike. Izvođenje koda pokretača i jezgre obavlja se korištenjem fizičkih (apsolutnih adresa), dok se za programe koristi straničenje (i logičke adrese). Kod se priprema za sustav koji ima ROM na adresi 0x100000 te RAM na adresi 0x200000. Radi uštede prostora u ROM-u, slike jezgre (`kernel.bin`) i programa (`p1.bin`, `p2.bin`, `p3.bin`) se komprimiraju (`kernel+progs.zip`) i kao takve učitavaju u ROM. Zadatak pokretača je da otpakira i jezgru i programe u RAM te potom pokrene jezgru. Stoga mu je preko skripte za povezivanje potrebno poslati potrebne podatke, tj. gdje će se naći ta slika jezgre i programa u ROM-u, te kamo ju otpakirati (RAM). Također, jezgri su potrebne informacije gdje će se u RAM-u nalaziti otpakirane slike programa i koliko su velike. Stoga te informacije dodati u skriptu jezgre (u neke varijable). U svakom direktoriju nalazi se zasebna datoteka `prva.c` čiji kod treba postaviti na početak slika jer se prvi izvodi (u programu pokretaču će tu biti kod za otpakiranje, u jezgri za inicijalizaciju jezgre i pokretanje programa, u programima za rad korisna posla – ne pisati kod tih datoteka).

   Napisati skripte za povezivanje za sve tri komponente (pokretača, jezgru i programe). Nazivi/broj odjeljaka u izlaznim datotekama odaberite sami (može biti i samo jedan). Opisati postupak prevođenja i upisivanja slika u ROM.

za programe: px.ld

```
SECTIONS {
   .sve 0 : {
      prva.o (*)
      * (*)
   }
}
```

za jezgru: kernel.ld

```
RAM = 0x200000
SECTIONS {
   .sve RAM : {
      prva.o (*)
      * (*)
   }
   svevel = SIZEOF(.sve);
   p1adr = RAM + svevel;
   p1vel = P1VEL;
   p2adr = p1adr + p1vel;
   p2vel = P2VEL;
   p3adr = p2adr + p2vel;
   p3vel = P3VEL;
}
```

za pokretač: bootloader.ld

```
ROM = 0x100000
RAM = 0x200000
SECTIONS {
   .sve ROM : {
      prva.o (*)
      * (*)
   }
   svevel = SIZEOF(.sve);
   slika_ROM = ROM + svevel;
   otpakiraj_u = RAM;
}
```

Vrijednosti **P1VEL**, **P2VEL** i **P3VEL** dobivaju se nakon što se programi prevedu – pogleda se veličina tih datoteka. S tim podacima unesenim u `kernel.ld` prevede se i jezgra. Nakon toga se komprimiraju sve četiri datoteke: `kernel.bin`, `p1.bin`, `p2.bin`, `p3.bin` u `kernel+progs.zip`. Pokretač se prevede u `bootloader.bin` te se nakon toga u ROM upisuje prvo pokretač, a iza njega jezgra i programi.

5. (3) Neki sustav ima dva 24-bitovna brojila. Prvi na adresi `B1` odbrojava frekvencijom 10 MHz, a drugi na adresi `B2` odbrojava s 10 kHz. Kad brojila dođu do nule izazivaju prekid te učitavaju zadnju im upisanu vrijednost i ponovno odbrojavaju prema nuli. Brži iskoristiti za ostvarenje sata kojeg zapisati u sekundama i mikrosekundama (struktura `vrijeme_t`), a drugi za ostvarenje jednog alarma u granulaciji milisekunde (pretpostaviti da će tražena odgoda u milisekundama biti manja od 2^24/10). Potrebna sučelja (funkcije koje treba implementirati) su:

```
void inicijaliziraj(vrijeme_t početni_sat);
vrijeme_t dohvati_vrijeme();
vrijeme_t postavi_vrijeme(vrijeme_t novo_vrijeme); //vrati prijašnje vrijeme
void postavi_alarm(void (*obrada)(), int za_koliko_ms);
void prekid_brzog_brojila();
void prekid_sporog_brojila();
```

```
2^24 - 1 = 0xFFFFFF = 16777215
MAX1 = 16777210 //zadnji je 0 a ne 5, da se ne izgubi 0,5 mikrosekundi
MAX2 = 0xFFFFFF

typedef stuct _vrijeme_t_
{
    long sek;
    long usek;
 } vrijeme_t;

vrijeme_t sat;
void (*alarm)();

void inicijaliziraj(
   vrijeme_t početni_sat)
{
   sat = početni_sat;
   alarm = NULL;
   B1 = MAX1;
   B2 = MAX2;
}
vrijeme_t dohvati_vrijeme()
{
   vrijeme_t t = sat;
   long br = (MAX1 - B1) / 10;
   if (br >= 1000000) {
      t.sek++;
      br -= 1000000;
   }
   t.usek += br;
   if (t.usek >= 1000000) {
      t.sek++;
      t.usek -= 1000000;
   }
   vrati t;
}
vrijeme_t postavi_vrijeme(
   vrijeme_t novo_vrijeme)
{
   vrijeme_t t = dohvati_vrijeme();
   sat = novo_vrijeme;
   B1 = MAX1;
   vrati t;
}
void postavi_alarm(
   void (*obrada)(), int za_koliko_ms)
{
   alarm = obrada;
   B2 = za_koliko_ms * 10;
}
void prekid_brzog_brojila()
{
   sat.sek++;
   sat.usek += MAX1 / 10 - 1000000;
   if (sat.usek >= 1000000) {
      sat.sek++;
      sat.usek -= 1000000;
   }
}
void prekid_sporog_brojila()
{
   if (alarm != NULL) {
      void (*x)() = alarm;
      alarm = NULL;
      B2 = MAX2;
      x();
   }
}
```

6. (4) Ostvariti upravljački program za upravljanje serijskom vezom. Upravljački program mora implementirati sučelje:

```
struct naprava {
    char ums[128], ims[128]; //dodatni međuspremnici u memoriji
    int u_prvi, u_velicina, i_prvi, i_velicina;
    void inicijaliziraj(struct naprava *n);
    void prekid_novi_podaci(struct naprava *n); //kad se ulazni međuspremnik sklopa popuni
    void prekid_mogu_slati(struct naprava *n); //kad se izlazni međuspremnik isprazni
    int posalji(struct naprava *n, size_t velicina, void *ms); // broj poslanih bajtova
    int procitaj(struct naprava *n, size_t velicina_ms, void *ms); //broj primljenih najtova
}
```

Sklop za komunikaciju koristi registre (koristiti ta imena kao varijable u kodu):

`PR` – podatkovni registar – preko njega se šalju i čitaju podaci (pisanje=slanje, čitanje=primanje)

`UR` – upravljački registar s bitovima (samo se piše):

0. bit: upravljanje slanjem: 0 - znak po znak ili 1- koristi se izlazni međuspremnik sklopa (prekid nakon svakog znaka ili tek kad se taj međuspremnik isprazni; postaviti ovo drugo)

1. bit: upravljanje primanjem: znak po znak ili se koristi ulazni međuspremnik sklopa (prekid nakon svakog znaka ili tek kad se taj međuspremnik popuni na >80%; postaviti ovo drugo)

`SR` – statusni registar s dijelovima (samo se čita):

bitovi 0-3: broj još neposlanih bajtova u izlaznom međuspremniku sklopa

bitovi 4-7: broj novih (nepročitanih) bajtova u ulaznom međuspremniku sklopa

Međuspremnike upravljačkog programa (`ums`, `ims`) koristiti kružno. Međuspremnici sklopa mogu prihvatiti po 16 bajtova svaki.

```
void inicijaliziraj(struct naprava *n) {
   n->u_prvi = n->u_velicina = 0;
   n->i_prvi = n->i_velicina = 0;
   UR = 0b11;
}
void prekid_mogu_slati(struct naprava *n){
   while ((SR & 0x0f) < 16 && n->i_velicina > 0) {
      PR = n->ims[n->i_prvi];
      n->i_prvi = (n->i_prvi + 1) % 128;
      n->i_velicina--;
   }
}
void prekid_novi_podaci(struct naprava *n)
{
   int zadnji = (n->u_prvi + n->u_velicina) % 128;
   while (((SR & 0xf0) >> 4) > 0 && n->u_velicina < 128) {
      n->ums[zadnji] = PR;
      zadnji = (zadnji + 1) % 128;
      n->u_velicina++;
   }
}
int posalji(struct naprava *n, size_t velicina, void *ms)
{
   int i, zadnji = (n->i_prvi + n->i_velicina) % 128;
   for (i = 0; i < velicina && n->i_velicina < 128; i++) {
      n->ims[zadnji] = ms[i];
      zadnji = (zadnji + 1) % 128;
      n->i_velicina++;
   }
   prekid_mogu_slati(n);

   return i;
}
```

```
int procitaj(struct naprava *n, size_t velicina_ms, void *ms)
{
   int i;
   for (i = 0; i < velicina && n->u_velicina > 0; i++) {
      ms[i] = n->ums[n->u_prvi];
      n->u_prvi = (n->u_prvi + 1) % 128;
      n->u_velicina--;
   }
   prekid_novi_podaci(n);
   for (; i < velicina && n->u_velicina > 0; i++) {
      ms[i] = n->ums[n->u_prvi];
      n->u_prvi = (n->u_prvi + 1) % 128;
      n->u_velicina--;
   }
   return i;
}
```

7. (3) Ostvariti raspoređivanje dretvi, tj. osmisliti potrebnu strukturu podataka i ostvariti funkcije:

   `void dodaj_u_pripravne(struct jdretva *dretva);` //dodaje dretvu u red pripravnih

   `struct jdretva *uzmi_prvu_iz_pripravnih();` //iz pripravnih uzima dretvu najveća prioriteta

   `void odaberi_aktivnu();` //postavi novu aktivnu, ako je potrebno

   Pretpostaviti da se koristi prioritetno raspoređivanje (kao SCHED_FIFO) s 16 prioriteta (0-15). Funkcija `odaberi_aktivnu()` treba odabrati novu (ili staru) aktivnu dretvu i u globalnu varijablu `struct jdretva *aktivna` postaviti kazaljku na aktivnu dretvu. U opisnik dretve dodati varijablu `stanje` koja pokazuje je li dretva aktivna, pripravna ili blokirana (ta je informacija potrebna u `odaberi_aktivnu()`).

   Pretpostaviti da postoje funkcije/makroi za rad s redovima (`struct red`): `dodaj_na_pocetak_reda(red, objekt)`, `dodaj_na_kraj_reda(red, objekt)`, `uzmi_prvi_iz_reda(red)` gdje je `red` složen po redu prispijeća. Operacija „uzmi“ vraća `NULL` ako je red prazan.

```
struct jdretva {
   int stanje;
   int prio;
   ...
}
struct red pripr[16];

void dodaj_u_pripravne(struct jdretva *dretva)
{
   dodaj_na_kraj_reda(pripr[dretva->prio], dretva);
   dretva->stanje = PRIPRAVNA;
}
struct jdretva *uzmi_prvu_iz_pripravnih() {
   int i;
   for (i = 15; i >= 0; i++) {
      struct jdretva *dretva = uzmi_prvi_iz_reda(pripr[i]);
      if (dretva != NULL)
         return dretva;
   }
   return NULL;
}
void odaberi_aktivnu()
{
   struct jdretva *dretva = uzmi_prvu_iz_pripravnih();
   if (aktivna->stanje != AKTIVNA || aktivna->prio < dretva->prio) {
      if (aktivna->stanje == AKTIVNA)
         dodaj_u_pripravne(aktivna);
      aktivna = dretva;
   }
   else {
      dodaj_na_pocetak_reda(pripr[dretva->prio], dretva)
   }
}
```

8. (3) U sustavu koji koristi procese (npr. metodom straničenja) ostvariti komunikacijsku metodu cjevovoda (neka može postojati samo jedna cijev). Pretpostaviti da već postoji podrška jezgre za pripremu internih poziva, tj. ostvariti samo interne jezgrine funkcije:

```
void j_cjevovod_stvori(size_t velicina_medjuspremnika);
size_t j_cjevovod_posalji(jproces_t *proces, size_t velicina, void *podaci);
size_t j_cjevovod_procitaj(jproces_t *proces, size_t velicina, void *podaci);
```

Varijabla `podaci` je u logičkim adresama procesa proces. Pretvorbu logičke u fizičku adresu i obratno rade funkcije `j_fizicka(jproces_t *pr, void *logicka)` i `j_logicka(jproces_t *pr, void *fizicka)`.

Kada se ne može sve staviti u cijev, staviti koliko stane i taj broj vratiti. Kada se čita iz cijevi pročitati koliko je traženo ili koliko ima (ako nema dovoljno). Ne blokirati dretve u tim pozivima. Koristiti `kmalloc` za alokaciju memorije za cijev. Međuspremnik za cijev koristiti kružno.

```
globalne varijable:
char *cijev = NULL;
size_t kapacitet = 0;
size_t u_cijevi = 0;
size_t prvi = 0;

void j_cjevovod_stvori(size_t velicina_medjuspremnika)
{
   if (cijev != NULL) return; //već stvoren
   cijev = kmalloc(velicina_medjuspremnika);
   kapacitet = velicina_medjuspremnika;
   u_cijevi = 0;
   prvi = 0;
}

size_t j_cjevovod_posalji(jproces_t *proces, size_t velicina, void *podaci)
{
   size_t kraj = (prvi + u_cijevi) % kapacitet;
   int i;

   for (i = 0; i < velicina && u_cijevi < kapacitet; i++) {
      //ovdje treba raditi pretvorbu adrese, ne prije; susjedne adrese mogu biti u različitim stranicama
      cijev[kraj] = *((char *) j_fizicka(proces, podaci + i));
      kraj = (kraj + 1) % kapacitet;
      u_cijevi++;
   }
   return i;
}

size_t j_cjevovod_procitaj(jproces_t *proces, size_t velicina, void *podaci)
{
   int i;
   for (i = 0; i < velicina && u_cijevi > 0; i++) {
      *((char *) j_fizicka(proces, podaci + i)) = cijev[prvi];
      prvi = (prvi + 1) % kapacitet;
      u_cijevi--;
   }
   return i;
}
```