Seminar 1: PREKIDI I SIGNALI

SIGNALI

Jezgra može poslati procesu signal. Taj signal može biti proizveden od same jezgre, proces ga može poslati sam sebi ili drugom procesu, ili ga može poslati korisnik.

Primjer signala koji potiče od jezgre je signal koji je poslan kada proces pokuša pristupiti memoriji koja nije u njegovom adresnom prostoru. Primjer signala koji proces šalje sam sebi je alarm. Signal koji proces šalje drugom procesu je signal koji uništava procese, a šalje ga proces koji želi uništiti svoju "porodicu" procesa. Tipični korisnikov signal je prekidni signal. Standardno je postavljeno da se taj signal generira tipkom **`DEL`**, međutim, većina korisnika to mijenja u **`Ctrl-C`** (`stty intr ^C`).

Postoji tridesetak različitih signala (u različitim verzijama UNIX-a može ih biti manje ili više). Za većinu signala (izuzetak je `SIGKILL` i još neki) proces može kontrolirati što se dešava nakon što primi neki signal. Može prihvatiti ugrađenu akciju što (za većinu signala) rezultira uništavanjem procesa, može ga ignorirati ili može uhvatiti signal i izvesti određenu funkciju. Tip signala (cijeli broj) se prenosi u funkciju kao jedini argument i funkcija ne može otkriti izvor signala. Nakon povratka iz funkcije proces može nastaviti od mjesta prekida.

Značenje signala za procese je analogno značenje prekidnih signala na razini procesora. Signal može biti poslan u bilo kojem trenutku, ali ne mora biti primljen i ne mora nitko reagirati na signal. Ne sadrže nikakve informacije i može biti poslan samo određenom procesu ili procesima. Nikada se ne koriste za normalnu komunikaciju, nego samo za posebne događaje.

Simbolička imena signala nalaze se u `signal.h`.

Vrste signala

Moderne implementacije UNIX-a definiraju oko tridesetak signala. Ovdje je pregled samo najznačajnijih, dok potpun popis daje `man signal`:

`SIGINT (2)`	šalje se svakom procesu za koji je to kontrolni terminal u trenutku kada je pritisnuta tipka za prekid (standardno **`DEL`**, najčešće izmijenjeno u **`Ctrl-C`**).
`SIGQUIT (3)`	Slično kao `SIGINT`, ali se odnosi na tipku za kraj izvršavanja (standardno **`Ctrl-\`**, najčešće izmijenjeno u **`Ctrl-X`**).
`SIGKILL (9)`	Jedini siguran način uništavanja procesa je da mu se pošalje ovaj signal, jer ne može biti ignoriran niti uhvaćen.
`SIGALARM (14)`	čalje se kada istekne traženo vrijeme čekanja procesa.
`SIGTERM (15)`	Ovo je standardni signal za uništavanje procesa. Koristi se i kod isključivanja da ubije sve aktivne procese. Očekuje se da će proces koji ga primi uredno spremiti aktualno stanje prije završetka.

sigset i njemu bliski sistemski pozivi

```
void (* sigset(int sig, void (* func)())() ;
```

specificira šta se dešava prilikom primitka određenog signala. *sig* je broj signala. Drugi argument je kazaljka na funkciju i može biti:

- `SIG_DFL`. U ovom slučaju postavlja se automatska reakcija na signal. Za većinu signala to znači uništavanje procesa.
- `SIG_IGN`. Ovime se postavlja da signal bude ignoriran. Signal `SIGKILL` se ne može ignorirati.

- `SIG_HOLD`. Signal se prihvaća ali ne obrađuje i obradit će se onda kada se definicija ponašanja za taj signal promijeni. Može se čuvati samo po jedan signal svake vrste.
- `Kazaljka` na funkciju. Ovime se označava da po primitku signala treba pozvati funkciju. Ne može se upotrijebiti za `SIGKILL`. Funkciji će biti predan jedan cjelobrojni argument - vrsta signala.

Prototip je nešto teže razumijeti jer je jedan od argumenata kazaljka na funkciju, a i sama funkcija kao rezultat vraća kazaljku na funkciju - prethodnu definiciju ponašanja za zadani signal. To se može koristiti ako kasnije treba vratiti prijašnje stanje.

Nakon povratka iz funkcije za obradu signala, proces nastavlja s radom od mjesta na kojem je prekinut primitkom signala. Međutim, ako je signal prihvaćen za vrijeme čekanja na izvršenje poziva *read*, *write*, *open*, *ioctl*, *wait*, ili *pause* oni će biti prekinuti uz postavljanje greške `EINTR` u *errno* (ne odnosi se na pristup običnim datotekama).

Neposredno pred poziv funkcije za obradu signala *func*, ponašanje kod prijema dotičnog signala se mijenja u `SIG_HOLD`. Na povratku iz te funkcije restaurira se *func* kao funkcija za obradu signala i otpušta eventualno pristigli signal. To se može obaviti i prije završetka same funkcije, pozivom *sigrelse*, što onda omogućava prijem novog signala prije nego što je prethodni obrađen.

```
int sighold(int sig)  ;
int sigrelse(int sig) ;
```

Ovo su pozivi za manipulaciju sa signalima. *sig* je signal.

sighold i *sigrelse* se koriste za zaštitu kritičnih odsječaka u programu. *sighold*je analogan podizanju prioriteta i zadržavanju signala dok se prioritet ne spusti sa *sigrelse. sigrelse* obnavlja akciju procesa prethodno specificiranu u *sigset*, te otpušta ranije primljeni i zadržani signal.

Ostali pozivi za manipulaciju signalima

Iako su ranije opisani pozivi dovoljni za rješenje zadatka, slijedi opis još nekoliko važnijih funkcija i sistemskih poziva povezanih sa signalima:

```
int pause(void) ;
```

zaustavlja proces do dolaska nekog signala. To ne može biti neki od signala koji se ignoriraju. Također, ako signal izazove prekid procesa, *pause* ne može ništa vratiti. Tek ako funkcija za hvatanje signala vraća nešto, *pause* se vraća s rezultatom -1 i *errno*postavljenim na `EINTR`.

```
unsigned alarm(unsigned secs) ;
```

postavlja sat na vrijednost *secs* što predstavlja broj sekunda, a vraća vrijednost na koju je prethodno bio postavljen. Svaki proces ima sat spremljen u segmentu sistemskih podataka. Kada istekne postavljeno vrijeme šalje se signal `SIGALRM`.

```
unsigned sleep(unsigned secs) ;
```

alarm i *pause*u kombinaciji formiraju standardnu funkciju *sleep*. Prvo se postavlja alarm na traženo vrijeme u sekundama, a zatim poziva *pause* da čeka prijem signala. Međutim, *sleep* može trajati i kraće od zadanog vremena jer *pause* ne čeka samo `SIGALRM` nego bilo koji signal. Također, *sleep* se brine o ranije postavljenim alarmima pa može završiti ranije ili će po svom završetku restaurirati od prije postojeći alarm. U svakom slučaju, *sleep* kao rezultat vraća "neprospavano" vrijeme (koliko je proces ranije probuđen zbog drugih signala). To vrijeme se onda može nadoknaditi dodatnim spavanjem.

```
int kill(int pid, int sig) ;
```

šalje signal procesu. *pid* je ID broj procesa koji prima signal, a *sig* je broj signala. Ako je *pid* jednak 0, signal se šalje svim procesima koji su istoj grupi kao i proces koji šalje signal. To se obično upotrebljava kod komande *kill*, čime se uništavaju svi procesi koji se izvode u pozadini, bez obzira na njihov ID.

U većini slučajeva *kill* se koristi za uništavanje procesa (od tuda dolazi ime) ili za testiranje ponašanja kod greške simuliranjem signala. Najčešće se koristi kao komanda, a ne kao poziv u programu.

```
void (* signal(int sig, void (* func)())() ;
```

specificira šta se dešava prilikom primitka određenog signala, slično kao i*sigset*. *signal* je stariji i razlikuje se po tome što ne poznaje `SIG_HOLD`. Također, ponašanje na ulasku u funkciju za obradu signala čini ga manje upotrebljivim od *sigset*.

Neposredno pred poziv funkcije za obradu signala, ponašanje kod prijema dotičnog signala se mijenja u `SIG_DFL`. To znači da će slijedeći signal istog tipa ubiti proces. Ovo ponašanje se može promijeniti unutar same funkcije za obradu signala, ali uvijek postoji interval (koji kod većeg opterećenja može biti i prilično dugačak) unutar kojeg prijem signala može ubiti proces.

ZADATAK

Napisati program koji omogućava obradu prekida sa više razina prekida. Struktura prekidne rutine dana je sljedećim pseudokodom:

```
prekidna_rutina  /* pokreće se pojavom prekida uz zabranu daljih prekida */
   ispiši_vrijeme_ulaska_u_prekidnu_rutinu_s_točnošću_na_sekundu();
   odredi uzrok prekida, tj. indeks i
   postavi OZNAKA_ČEKANJA[i]
   dok j takav da je (OZNAKA_ČEKANJA[j]<>0  j>TEKUćI_PRIORITET) čini
      izaberi najveći j
      izbriši OZNAKA_ČEKANJA[j]
      spremi TEKUćI_PRIORITET u PRIORITET[j]
      proglasi TEKUćI_PRIORITET = j
      omogući prekidanje
      obrada_prekida(j)
      zabrani prekidanje
      vrati TEKUćI_PRIORITET iz PRIORITET[j]
   omogući prekidanje
kraj.
```

UNIX ne dozvoljava pojedinom korisniku izravno korištenje prekida procesora. Zato prekide treba simulirati koristeći signale koje jezgra operacijskog sustava šalje procesima.

UPUTA

Sklopovski prekid u jednoj razini obavlja se pritiskom na tipku **`Ctrl-C`**, čime se programu šalje signal `SIGINT`. Nakon toga će se prekinuti izvođenje programa, zabraniti dalje prekidanje, i pozvati funkcija za obradu signala koja simulira prekidnu rutinu.

Na početku prekidne rutine mora se nalaziti naredba koja će učitati indeks *i*, što je adekvatno određivanju uzroka prekida. Prekidna rutina mora biti funkcija s jednim cjelobrojnim parametrom (koji može ignorirati) i mora biti najavljena u glavnom programu naredbom:

```
sigset (SIGINT, prekidna_rutina);
```

Zabrana prekida simulira se naredbom: `sighold(SIGINT);`

Omogućavanje prekida simulira se naredbom: `sigrelse(SIGINT);`

Obrada prekida ne mora ništa korisno raditi, već samo treba trajati neko vrijeme. Umjesto dugih petlji, može poslužiti i *sleep*.

Za ispis vremena vidjeti npr. "man ctime".

Kostur rješenja dan je slijedećim kodom:

```
#include <stdio.h>
#include <signal.h>

#define N 10    /* broj razina prekida */

int OZNAKA_CEKANJA[N];
int PRIORITET[N];
int TEKUCI_PRIORITET;

void obrada_prekida ( int j )
{
   printf ("Poceo obradu prekida %d\n", j);
   /* obrada se simulira trošenjem vremena, sleep() ili slično - 10 s */
   printf ("Zavrsio obradu prekida %d\n", j);
}

void prekidna_rutina ( int sig )
{
   int i;

   printf ("Upisi razinu prekida ");
   scanf ("%d", &i);
                ...
}

int main ( void )
{
   sigset (SIGINT, prekidna_rutina);
   printf ("Poceo osnovni program\n");
   /* troši vrijeme da se ima šta prekinuti - 10 s */
   printf ("Zavrsio osnovni program\n");

   return 0;
}
```