SVEUČILIŠTE U MOSTARU
FAKULTET STROJARSTVA, RAČUNARSTVA I ELEKTROTEHNIKE
PREDDIPLOMSKI STUDIJ RAČUNARSTVA

OPERACIJSKI SUSTAVI

VJEŽBE

Nastavnik: prof.dr.sc. Sven Gotovac
gotovac@fesb.hr

Asistent: Željko Šeremet
zeljko.seremet@fsre.sum.ba

MOSTAR, SVIBANJ 2024.

Signali i prekidi

Sadržaj

- Pojam signali
- Vrste signala
- *sigset* i njemu bliski sustavski pozivi
- Ostali pozivi za manipulaciju signalima
- Win32

Pojam signal (1)

- 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 (SIGSEGV). Primjer signala koji proces šalje sam sebi je alarm (SIGALRM). Signal koji proces šalje drugom procesu je najčešće 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).

Pojam signal (2)

- 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.

Pojam signal (3)

- Značenje signala za procese je analogno značenju 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. Najčešće se ne koriste za normalnu komunikaciju, nego samo za posebne događaje.
- Simbolička imena signala nalaze se u signal.h.

Vrste signala (1)

- Imena signala su pozitivne cjelobrojne konstante.
- ANSI C standard specificira sljedećih 6 signala: SIGINT, SIGILL, SIGFPE, SIGSEGV, SIGTERM, SIGABRT..

Vrste signala (2)

- Moderne implementacije UNIX-a definiraju oko tridesetak signala. Ovdje je pregled samo najznačajnijih, dok potpun popis daje man -s 3HEAD 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**).

Vrste signala (3)

- SIGKILL (9) - Jedini siguran način uništavanja procesa je da mu se pošalje ovaj signal, jer ne može biti ignoriran niti uhvaćen.
- SIGALARM (14) - šalje 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 SUSTAVSKI POZIVI (1)

- void (* sigset(int sig, void (* func)())() ;
- specificira što će se zbiti prilikom primitka određenog signala (maskira signal). *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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (2)

- 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).

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (3)

- Neposredno pred poziv funkcije za obradu signala *func*, ponašanje prilikom 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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (4)

- 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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (5)

- Iako su ranije opisani pozivi dovoljni za rješenje zadatka, slijedi opis još nekoliko važnijih funkcija i sustavskih 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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (6)

- 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 sustavskih podataka. Kada istekne postavljeno vrijeme šalje se signal SIGALRM.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (7)

- 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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (8)

- int usleep(useconds_t useconds);
- Funkcija usleep je slična funkciji sleep, samo što može raditi s kraćim vremenskim intervalima od jedne sekunde. Odnosno, vrijeme 'spavanja' izražava se u mikrosekundama.
- 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 ljuske *kill*, čime se uništavaju svi procesi koji se izvode u pozadini, bez obzira na njihov ID.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (9)

- 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.

SIGSET I NJEMU BLISKI SUSTAVSKI POZIVI (10)

- Umjesto funkcije sigset može se koristiti i naprednija funkcija sigaction. Opis te funkcije dan je i kratkim primjerom (sigaction.c).
- Osim direktno preko tipkovnice, proces može dobiti signal od drugog procesa (funkcija kill(pid,sig)). Također, iz komandne linije se naredbom kill može poslati signal nekom od procesa (npr. kill -SIGINT 12345).

Primjer

- Kostur programa za obradu prekida dan je sljedećim kodom:

```
#include <stdio.h>
#include <signal.h>
void prekidna_rutina(int sig)
{ /* obrada prekida */}
int main (void)
{
    sigset (SIGTSTP, prekidna_rutina);
    printf("Poceo osnovni program PID=%d\n", getpid());
    /* troši vrijeme da se ima što prekinuti - 10 s */
    printf ("Zavrsio osnovni program\n");
    return 0;}
```

Win32

- Ukoliko se želi raditi u Win32 okruženju stvari se malo kompliciraju, jer Win32 arhitektura drukčije obrađuje prekide. Naime, za obradu prekida stvara se nova dretva te program odjednom postane višedretveni!
- Više o usklađenosti te prenošenju UNIX programa na Win32.

Zadaća 1.

- Neka program simulira neki dugotrajni posao (slično servisima) koji koristi dvije datoteke: u jednu dodaje do sada izračunate vrijednosti (npr. kvadrati slijednih brojeva), a u drugu podatak do kuda je stigao. Npr. u **obrada.txt** zapisuje 1 4 9 … (svaki broj u novi red) a u **status.txt** informaciju o tome gdje je stao ili kako nastaviti. Npr. ako je zadnji broj u **obrada.txt** 100 u **status.txt** treba zapisati 10 tako da u idućem pokretanju može nastaviti raditi i dodavati brojeve.
- Prije pokretanja te je datoteke potrebno ručno napraviti i inicijalizirati. Početne vrijednosti mogu biti iste – broj 1 u obje datoteke.
- Pri pokretanju programa on bi trebao otvoriti obje datoteke, iz **status.txt**, pročitati tamo zapisanu vrijednost. Ako je ona veća od nule program nastavlja s proračunom s prvom idućom vrijednošću i izračunate vrijednosti nadodaje u **obrada.txt**. Prije nastavka rada u **status.txt** upisuje 0 umjesto prijašnjeg broja, što treba označavati da je obrada u tijeku, da program radi.
- Na signal (npr. SIGUSR1) program treba ispisati trenutni broj koji koristi u obradi. Na signal SIGTERM otvoriti **status.txt** i tamo zapisati zadnji broj (umjesto nule koja je tamo) te završiti s radom.
- Na SIGINT samo prekinuti s radom, čime će u **status.txt** ostati nula (čak se taj signal niti ne mora maskirati – prekid je pretpostavljeno ponašanje!). To će uzrokovati da iduće pokretanje detektira prekid – nula u **status.txt**, te će za nastavak rada, tj. Određivanje idućeg broja morati „analizirati“ datoteku **obrada.txt** i od tamo zaključiti kako nastaviti (pročitati zadnji broj i izračunati korijen). Operacije s datotekama, radi jednostavnosti, uvijek mogu biti u nizu open+fscanf/fprintf+close, tj. ne držati datoteke otvorenima da se izbjegnu neki drugi problemi. Ali ne mora se tako. U obradu dodati odgodu (npr. sleep(5)) da se uspori izvođenje.

Zadaća 1.

- Primjer pseudokoda za navedeni zadatak:

```
globalne varijable:
        broj
        imena datoteka
program
        maskiraj signale

        broj = pročitaj broj iz status.txt
        ako je broj == 0 onda
                čitaj brojeve iz obrada.txt dok ne dođeš do kraja datoteke
                broj = zadnji pročitani broj
        zapiši 0 u status.txt na početak datoteke (prepiši ono što je bilo!)
        ponavljaj //beskonačna petlja
                broj = broj + 1
                x = obrada(broj)
                dodaj x u obrada.txt
                odgodi(5)
na sigusr1:
        ispiši broj
na sigterm:
        zapiši broj u status.txt
        završi program
na sigint:
        završi program
```

Literatura

- Korisni linkovi:
- Kratke upute za rad u UNIX/Linux okruženju, prevođenje i pokretanje
- Auditorne vježbe - turbo uvod u C++
- Auditorne vježbe - nastavak C++
- Primjeri sa vježbi (primjeri.rar)

KRAJ