## 10.1 Signali - Uvod

Signali se koriste za prekidanja programa. Signali predstavljaju način za obradu asinhronih događaja. Npr., korisnik može ukucati naredbu u terminalu kako bi poslao signal za ubijanje programa koji se izvršava.

## 10.2 Osnovni koncepti signala

Svaki signal ima ime i svako od ovih imena počinje sa SIG. Npr., SIGALRM je signal koji se generiše kada se tajmer koji postavlja funcija *alarm* završi. Sa svakim od postojećih signala povezan je jedan pozitivan ceo broj (*broj signala)*. Ovi brojevi su definisani u zaglavlju *signal.h*. U različitim slučajevima se mogu generisati signali:

- Za vreme izvršavanja programa korisnik šalje *signal prekida* (SIGINT) komandom Control-C. Na ovaj način se može zaustaviti proces koji se otrgao kontroli (npr. zauzima previše memorije)
- Za vreme izvršavanja programa korisnik šalje *signal privremenog prekidanja* (SIGSTOP) komandom Control-Z. Na ovaj način se može zaustaviti proces i kasnije se može nastaviti sa izvršavanjem tog procesa.
- Hardverski izuzeci generišu signale. Npr., deljenje sa nulom, pristup nedozvoljenoj memoriji. Ova stanja obično detektuje hardver, obaveštava kernel i kernel generiše signal za odgovarajući proces (signal SIGSEGV za pristup nedozvoljenoj memoriji, SIGFPE za deljenje nulom).
- Sistemski poziv *kill* nam omogućava da iz našeg programa pošaljemo bilo koji signal nekom drugom procesu uz ograničenje da moramo da budemo superuser ili vlasnik procesa kome šaljemo signal.
- Komanda *kill* omogućava nam da pošaljemo signal drugim procesima. Najčešće se koristi za prekid procesa pokrenutih u pozadini. Implementirana je da koristi sistemski poziv *kill*.
- Signali se mogu generisati kada se proces treba obavestiti o različitim događajima. Primer ovog tipa signala je SIGALRM.

Signali su asinhroni događaji koji se dešavaju u nepredvidivim vremenskim trenucima. Sa njima se radi tako što se obavesti kernel o *akciji* koja se preduzima kada program primi signal. Akcija može biti:

- Ignorisanje signala. Dva signala su izuzetak i ne mogu se ignorisati: SIGKILL i SIGSTOP. Ovi signali predstavljaju siguran način da superuser ili kernel ubiju ili *privremeno* stopiraju proces koji se otrgao kontroli. Proces koji je stopiran pomoću SIGSTOP se kasnije može nastaviti slanjem signala SIGCONT.
- Hvatanje signala. Kernel se obaveštava da treba pozvati našu funkciju kada dođe do signala. U ovoj funkciji možemo na primer obrisati privremene fajlove pre nego što prekinemo program. Signali SIGKILL i SIGSTOP ne mogu se uhvatiti.
- Podrazumevana akcija. Svaki signal ima podrazumevanu akciju (najčešće prekid programa).

Slika 10.1 u knjizi APUE prikazuje listu signala i podrazumevane akcije za svakog od njih. Za neke signale (u poslednjoj koloni imaju tekst *terminate*+*core*) se pri prekidu kreira fajl sa ekstenzijom *.core* u tekućem direktorijumu koji čuva informacije koje se mogu iskoristiti za debagovanje kako bi se videlo zašto je došlo do pucanja programa.

## 10.3 Sistemski poziv *signal*

Najlakši interfejs za rad sa signalima obezbeđuje funkcija *signal.* Njena deklaracija je

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

ili

```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

Arugment *signo* označava broj signala o kome se radi – kada dođe do tog signala treba pozvati funkciju *func*. Ona može biti SIG_IGN (ignorisanje signala), SIG_DFL (podrazumevana akcija) ili adresa funkcije koju smo napisali i koja treba da bude pozvana (ovu funkciju zovemo *signal-hendler* ili *funkcija za hvatanje signala*). Kada se završi signal handler funkcija, nastavlja se dalje izvršavanje koda. Funkcija *signal* vraća akciju koja je prethodno bila dodeljena tom signalu sa brojem signum.

**(PRIMER sig_usr_int.c)**
Jedna sesija pokretanja programa bi mogla da bude:

```
$ ./sigusr &

[5] 12082

$ kill -USR1 12082

SIGUSR1 signal caught.

$ kill -USR2 12082

SIGUSR2 signal caught.

$ kill -2 12082
SIGINT signal caught.

$ kill -9 12082

$
[5]+  Terminated              ./sigusr
```

Prethodni pozivi demonstriraju slanje signala iz komandne linije i obradu signala iz programa koji smo napisali. U man stranici *kill* komande mogu se videti opcije za slanje raznih signala programima (-2 za SIGINT, -9 za SIGKILL).

## 10.10 Funkcije *alarm* i *pause*

Funkcijom *alarm* postavljamo tajmer koji se završava posle naznačenog vremena

u budućnosti. Po završetku se generiše SIGALRM signal. Podrazumevana akcija za ovaj signal je prekid procesa. Ako se ova funkcija pozove a još uvek nije istekao prethodno postavljeni tajmer, onda se prethodni tajmer briše i formira novi. Ako želimo da uhvatimo SIGALRM, bitno je da postavimo hvatanje ovog signala pre nego pozovemo funkciju *alarm*.

**(PRIMER sleep.c)** Primer prikazuje našu verziju *sleep* funkcije.

**(PRIMER fibonaci_timer.c)** Primer prikazuje kako pokrenuti niz programa sa datim vremenskim ograničenjem za svaki program.

**(PRIMER segmentation_fault.c)** Primer prikazuje obrađivanje pristupa nedozvoljenoj memoriji umesto da se ostavi predefinisana akcija (predefinisana akcija je ispis *Segmentation fault* i prekid programa).

## Zadaci

Naredni zadatak se dosta razlikuje od prethodnog programa pa je savet krenuti iz početka a ne kopirati prethodni kod.

1. Napisati program koji pravi promenljivu *x* kojoj dodeljuje vrednost 0 i zatim vrši deljenje nekog celog broja ovom promenljivom (dakle, deljenje nulom). Obraditi ovaj signal (SIGFPE) i u tom slučaju ispisati odgovarajuću poruku o vrsti primljenog signala i prekinuti program.

2. Napisati program koji redom izračunava članove Fibonačijevog niza čiji su indeksi dati u datoteci čije se ime navodi kao prvi argument komandne linije i štampa ih na standardni izlaz. Koristiti tip *long double* za čuvanje i štampanje rezultata. Pre izračunavanja treba obezbediti da se u slučaju signala SIGALRM prekida program uz obaveštenje kod kog člana je došlo do prekida (iz drugog terminala se pokreće komanda *kill -14 process_id*). Na primer, jedna sesija programa bi mogla biti:

```
fib (1.000000) = 1.000000

fib (3.000000) = 3.000000

fib (4.000000) = 5.000000

fib (5.000000) = 8.000000

fib (9.000000) = 55.000000

fib (15.000000) = 987.000000

fib (17.000000) = 2584.000000
```

```
fib (20.000000) = 10946.000000

fib (25.000000) = 121393.000000

fib (30.000000) = 1346269.000000

fib (40.000000) = 165580141.000000

Not enough time to calculate fibonaci number with index of 50.000000
```

3. Modifikovati prethodni program tako da se za svaki izračunati član štampa i ukupno vreme utrošeno za njegovo izračunavanje.