

## Lab-1. Signali

Mehanizam signala na razini operacijskog sustava omogućava obradu događaja koji se pojavljuju paralelno s normalnim radom programa, tj. procesa, tj. njegove dretve.

Po tome je signal sličan mehanizmu prekida na razini procesora: procesor izvodi neku dretvu koju prekida prekid neke naprave. Procesor tada privremeno prekida izvođenje dretve prema njen kontekst, te skače na obradu prekida naprave. Po završetku obrade prekida vraća se i nastavlja s dretvom (obnavlja njen kontekst). Slično signali prekidaju dretvu, poziva se funkcija za obradu signala (pretpostavljena ili zadana u programu) te se po njenom završetku vraća u dretvu i nastavlja s njenim radom.

Razmotrimo primjere signala `SIGINT` (*signal interrupt*) i `SIGTERM` (*terminate*). Procesu se `SIGINT` šalje kad ga se želi prekinuti u radu. Najčešće je to „nasilan“ prekid zbog neke greške u izvođenju procesa. S druge strane, `SIGTERM` također služi za prekid rada procesa, ali najčešće zbog drugih razloga, ne zbog grešaka programa, npr. pri gašenju sustava kad treba ugasiti sve procese, posebice one „u pozadini“ (service).

U terminalu, aktivnom procesu ćemo `SIGINT` poslati kombinacijom tipki `Ctrl+C` i proces će se prekinuti (uobičajeno ponašanje). Signal se može poslati i posebnim naredbama ljsuke ili drugim programima preko sučelja OS-a. Naredbom `kill` signal možemo poslati procesu ako znamo njegov identifikacijski broj (PID) sa:

```
$ kill -<id_signala> <PID>
```

Neka proces ima PID 2351 tada mu `SIGTERM` možemo poslati sa:

```
$ kill -SIGTERM 2351
```

Znak `$` predstavlja prefiks u naredbenom retku ljsuke, nije dio naredbi.

Za većinu signala program može sam definirati što da se napravi u slučaju primitka. Ukoliko se to ne napravi, koristiti će se „uobičajeno“ ponašanje. Za većinu signala će to značiti prekid izvođenja programa, kao što je to za `SIGINT` i `SIGTERM`.

Program definira ponašanje za signal tako da OS-u kaže koju funkciju definiranu programom treba pozvati za pojedini signal. Postoji nekoliko sučelja za definiranje ponašanja programa za neki signal. Starija `signal` i `sigset` se nastoje zamijeniti novijim `sigaction`, pa će se on koristiti na primjeru u nastavku.

U primjeru se maskiraju tri signala `SIGUSR1`, `SIGTERM` i `SIGINT`. Signal `SIGUSR1` je „korisnički“ signal, bez posebne namjene. Ovdje se on koristi kao simulacija pojava nekog događaja na koji treba nešto napraviti, što se ovdje simulira. `SIGTERM` i `SIGINT` će po ispisu poruke prekinuti izvođenje procesa, s time da se u ovom primjeru sa `SIGTERM` ne izlazi odmah već samo preko varijable `nije_kraj` označi da treba završiti s radom.

Opis linija koda je naveden unutar samog programa.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

/* funkcije za obradu signala, navedene ispod main-a */
void obradi_dogadjaj(int sig);
void obradi_sigterm(int sig);
void obradi_sigint(int sig);

int nije_kraj = 1;

int main()
{
    struct sigaction act;

    /* 1. maskiranje signala SIGUSR1 */
    act.sa_handler = obradi_dogadjaj; /* kojom se funkcijom signal obrađuje */
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, SIGTERM); /* blokirati i SIGTERM za vrijeme obrade */
    act.sa_flags = 0; /* naprednije mogućnosti preskočene */
    sigaction(SIGUSR1, &act, NULL); /* maskiranje signala preko sučelja OS-a */

    /* 2. maskiranje signala SIGTERM */
    act.sa_handler = obradi_sigterm;
    sigemptyset(&act.sa_mask);
    sigaction(SIGTERM, &act, NULL);

    /* 3. maskiranje signala SIGINT */
    act.sa_handler = obradi_sigint;
    sigaction(SIGINT, &act, NULL);

    printf("Program s PID=%ld krenuo s radom\n", (long) getpid());
    /* neki posao koji program radi; ovdje samo simulacija */
    int i = 1;
    while(nije_kraj) {
        printf("Program: iteracija %d\n", i++);
        sleep(1);
    }
    printf("Program s PID=%ld završio s radom\n", (long) getpid());

    return 0;
}

void obradi_dogadjaj(int sig)
{
    int i;
    printf("Pocetak obrade signala %d\n", sig);
    for (i = 1; i <= 5; i++) {
        printf("Obrada signala %d: %d/5\n", sig, i);
        sleep(1);
    }
    printf("Kraj obrade signala %d\n", sig);
}

void obradi_sigterm(int sig)
{
    printf("Primio signal SIGTERM, pospremam prije izlaska iz programa\n");
    nije_kraj = 0;
}

void obradi_sigint(int sig)
{
    printf("Primio signal SIGINT, prekidam rad\n");
    exit(1);
}

```

Za pokretanje i demonstraciju potrebno je otvoriti dvije konzole/terminala, u jednoj će se pokrenuti program, a u drugoj naredbom `kill` slati signale, osim signala `SIGINT` koji se može poslati izravno s `Ctrl+C`. Primjeri rada prikazani su u dva stupca: s lijeve strane je ispis programa, a s desne naredbe koje su pokretane u drugoj konzoli u datom trenutku.

#### Primjer 1. Slanje `SIGINT` sa `Ctrl+C`

Terminal 1	Terminal 2
<pre>\$ gcc sig-primjer-1.c -o sig1 \$ ./sig1 Program s PID=14284 krenuo s radom Program: iteracija 1 Program: iteracija 2 Program: iteracija 3 ^CPrimio signal SIGINT, prekidam rad \$</pre>	

U prvom primjeru prikazano je slanje signala `SIGINT` preko kratice `Ctrl+C` (drugi terminal se u ovom primjeru nije koristio). Pri primitku signala pozvala se zadana funkcija koja je dotično ispisala te završila s radom. Isti signal se može poslati i naredbom `kill`, samo prvo treba zapamtiti PID procesa.

#### Primjer 2. Slanje `SIGINT` sa `kill`

Terminal 1	Terminal 2
<pre>\$ ./sig1 Program s PID=14296 krenuo s radom Program: iteracija 1 Program: iteracija 2 Program: iteracija 3 Primio signal SIGINT, prekidam rad \$</pre>	<pre>\$ kill -SIGINT 14296</pre>

Slično je i sa slanjem drugih signala.

#### Primjer 3. Slanje `SIGTERM`

Terminal 1	Terminal 2
<pre>\$ ./sig1 Program s PID=14299 krenuo s radom Program: iteracija 1 Program: iteracija 2 Program: iteracija 3 Primio signal SIGTERM, pospremam prije izlaska iz programa Program s PID=14299 završio s radom \$</pre>	<pre>\$ kill -SIGTERM 14299</pre>

Primjer s očekivanom obradom za `SIGUSR1` je u nastavku.

#### Primjer 4. Slanje `SIGUSR1`

Terminal 1	Terminal 2
<pre>\$ ./sig1 Program s PID=14425 krenuo s radom Program: iteracija 1 Program: iteracija 2 Pocetak obrade signala 10 Obrada signala 10: 1/5 Obrada signala 10: 2/5 Obrada signala 10: 3/5</pre>	<pre>\$ kill -SIGUSR1 14425</pre>

Obrada signala 10: 4/5 Obrada signala 10: 5/5 Kraj obrade signala 10 Program: iteracija 4 Program: iteracija 5 ^CPrimio signal SIGINT, prekidam rad \$	
--	--

Po primitku signala skače se u funkciju za obradu, a po dovršetku obrade vraća tamo gdje se stalo (u petlju u `main`). Pri prijemu prekida automatski je pohranjen kontekst dretve u tom trenutku prije nego li ona krene s obradom signala. Taj se kontekst poslije obnavlja i dretva normalno nastavlja s radom.

U obradi signala `SIGUSR1` privremeno se ne prihvaćaju novi signali `SIGUSR1` – to je uobičajeno ponašanje za svaki signal, on se privremeno blokira dok se prethodni istog tipa obrađuje. Međutim, u programu je pri maskiranju tog signala definirano da i `SIGTERM` ne prekida obradu signala `SIGUSR1`. Ti se signali obrađuju naknadno. Navedeno ponašanje pokazano je u slijedećem primjeru.

#### Primjer 5. Slanje više signala

Terminal 1	Terminal 2
\$ ./sig1 Program s PID=14492 krenuo s radom Program: iteracija 1 Program: iteracija 2 Početak obrade signala 10 Obrada signala 10: 1/5 Obrada signala 10: 2/5 Obrada signala 10: 3/5 Obrada signala 10: 4/5 Obrada signala 10: 5/5 Kraj obrade signala 10 Početak obrade signala 10 Obrada signala 10: 1/5 Obrada signala 10: 2/5 Obrada signala 10: 3/5 Obrada signala 10: 4/5 Obrada signala 10: 5/5 Kraj obrade signala 10 Primio signal SIGTERM, pospremam prije izlaska iz programa Program s PID=14492 završio s radom \$	\$ kill -SIGUSR1 14492  \$ kill -SIGUSR1 14492  \$ kill -SIGTERM 14492

Drugi signal `SIGUSR1` je prihvaćen tek nakon što je prvi obrađen, tj. on je do tada „bio na čekanju“. Slično je i sa signalom `SIGTERM` koji je prihvaćen tek po završetku druge obrade.

Ponašanje procesa na signal može biti:

1. obrada na uobičajeni način (pretpostavljeni), kada programom nije drukčije definirano,
2. obrada zadanom funkcijom (npr. sa `sigaction`),
3. privremeno ne prihvaćaj signal – blokiraj signal te
4. ignoriraj signal.

Načini 1, 2 i 4 mogu se postaviti sučeljem `sigaction`, uz konstantu `SIG_DFL` za 1, adresu funkcije za 2 te `SIG_IGN` za ignoriranje signala. Ponašanje za 3 se automatski postavlja pri

prihvatu signala. Sučeljem `sigset` i konstantom `SIG_HOLD` može postaviti ponašanje 3 kao i pozivom funkcije `sighold`.

Ponekad želimo da se unutar obrade jednog signala ne prihvaćaju drugi signali, barem u dijelovima obrade. Isto tako, u drugim dijelovima obrade bi željeli ponovno prihvaćati signale. Ovo se može postići na nekoliko načina.

Jedan (jednostavniji, sa „starijim sučeljem“) je pozvati `sighold(sig)` za svaki signal koji želimo privremeno blokirati, odnosno, `sigrelse(sig)` za svaki signal koji želimo opet omogućiti.

Drugi (preporučeni) je koristiti `pthread_sigmask`, ili `sigprocmask` npr. kodom:

```
void blokiraj_odblokiraj_signale(int blokiraj)
{
    sigset_t signali;
    sigemptyset(&signali);
    sigaddset(&signali, SIGTERM);
    sigaddset(&signali, SIGINT);
    if (blokiraj)
        pthread_sigmask(SIG_BLOCK, &signali, NULL);
    else
        pthread_sigmask(SIG_UNBLOCK, &signali, NULL);
}
```

Signale procesu šalje operacijski sustav radi svojih razloga ili na zahtjev nekog procesa. U gornjim primjerima se signale slalo naredbom (programom) `kill` ili izravno preko `Ctrl+C` što je ljsuka interpretirala kao zahtjev za slanjem signala i poslala to procesu koji je pokrenula.

Mnogi mehanizmi operacijskog sustava (UNIX) zasnivaju se na korištenju signala. Periodičke operacije moguće je napraviti tako da se od OS-a traži periodičko slanje signala s kojim se onda povezuje funkcija (npr. `setitimer`). Obična operacija `sleep(x)` ostvaruje se preko signala: dretva najprije od OS-a traži slanje signala nakon `x` sekundi (`alarm(x)`), a potom pauzira svoje izvođenje (`pause()`). Stoga se može dogoditi da takva odgoda sa `sleep(x)` ne traje `x` sekundi već manje. U simulaciji je poželjno `sleep(x)` zamijeniti petljom s `x` iteracija u kojoj se koristi `sleep(1)` da se smanji greška u odgodi.