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 sprema 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“ (servise).

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 ljuske 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 ljuske, 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 napravit, 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.

```c
#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 zavrsio 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

`$ 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`
`$`	

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

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

Slično je i sa slanjem drugih signala.

Primjer 3. Slanje `SIGTERM`

Terminal 1	Terminal 2

`$ ./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 zavrsio s radom`
`$`
	`$ kill -SIGTERM 14299`

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

Primjer 4. Slanje `SIGUSR1`

Terminal 1	Terminal 2

`$ ./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`
	`$ kill -SIGUSR1 14425`

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 prihvatu 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
Pocetak 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
Pocetak 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 zavrsio 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`.

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 ljuska 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.

Zadatak

Simulirati rad prekidnog sustava sa sklopom za prihvat prekida s tri razine prekida (tri prioriteta). Prekide simulirati s proizvoljno odabranim signalima. Npr. signal `SIGINT` simulira prekid najviše razine (3), signal `SIGUSR1` simulira prekid srednje razine (2), a signal `SIGTERM` simulira prekid najniže razine (1). Prekide generirati ili preko tipkovnice (npr. Ctrl+C) ili nekim drugim programom (vlastitim ili naredbom `kill`) koji se pokreće u zasebnom terminalu. Osim simulacije registara sklopa za prihvat prekida, dodatnom strukturom podataka simulirati i što se nalazi na stogu.

Prilikom pokretanja program treba ispisati početno stanje sustava kada se još nije dogodio ni jedan prekid. Kada se dogodi prekid ispisuje se razina prekida. Obrada prekida se simulira tako da program ispiše da je obrada prekida počela, stanje sustava tijekom obrade prekida (kontrolne zastavice, tekući prioritet i stanje na stogu) i nakon spavanja od nekoliko sekundi ispisuje da je obrada završila.

Na početku svakog ispisa staviti trenutno vrijeme. Primjer kako dohvatiti vrijeme i spavati određeno vrijeme može se pogledati (i iskoristiti) iz primjera lab1-primjer_spavanja.c.

Primjer ispisa (ispis ne mora izgledati tako, ali mora sadržavati ove informacije!):

```
$ ./lab1
000.000:        Program s PID=14497 krenuo s radom
000.000:        K_Z=000, T_P=0, stog: -

002.041:        SKLOP: Dogodio se prekid razine 2 i prosljeđuje se procesoru
002.041:        K_Z=010, T_P=0, stog: -

002.041:        Počela obrada razine 2
002.041:        K_Z=000, T_P=2, stog: 0, reg[0]

003.716:        SKLOP: Dogodio se prekid razine 1 ali se on pamti i ne prosljeđuje procesoru
003.716:        K_Z=100, T_P=2, stog: 0, reg[0]

^C005.416:      SKLOP: Dogodio se prekid razine 3 i prosljeđuje se procesoru
005.416:        K_Z=101, T_P=2, stog: 0, reg[0]

005.416:        Počela obrada prekida razine 3
005.416:        K_Z=100, T_P=3, stog: 2, reg[2]; 0, reg[0]

010.416:        Završila obrada prekida razine 3
010.416:        Nastavlja se obrada prekida razine 2
010.416:        K_Z=100, T_P=2, stog: 0, reg[0]

012.041:        Završila obrada prekida razine 2
```

```
012.041:        Nastavlja se izvođenje glavnog programa
012.041:        K_Z=100, T_P=0, stog: -

012.041:        SKLOP: promijenio se T_P, prosljeđuje prekid razine 1 procesoru
012.041:        Počela obrada razine 1
012.041:        K_Z=000, T_P=1, stog: 0, reg[0]

017.041:        Završila obrada prekida razine 1
017.041:        K_Z=000, T_P=0, stog: -
...
```

Kombinacije:
123 ✓
132 ✓
213 ✓
231 ✓
312 ✓
321 ✓

Kako posloziti program?

-ako zabranim neke signale, necu moci registrirati da su se dogodili - NAIVAN PRISTUP
-na razini signala ne koristiti blokiranje drugih signala
-ako se dogodi prekid razine 2 dok se odvija obrada prekida signala iste vrste - ne prekidati i zapocinjati novu

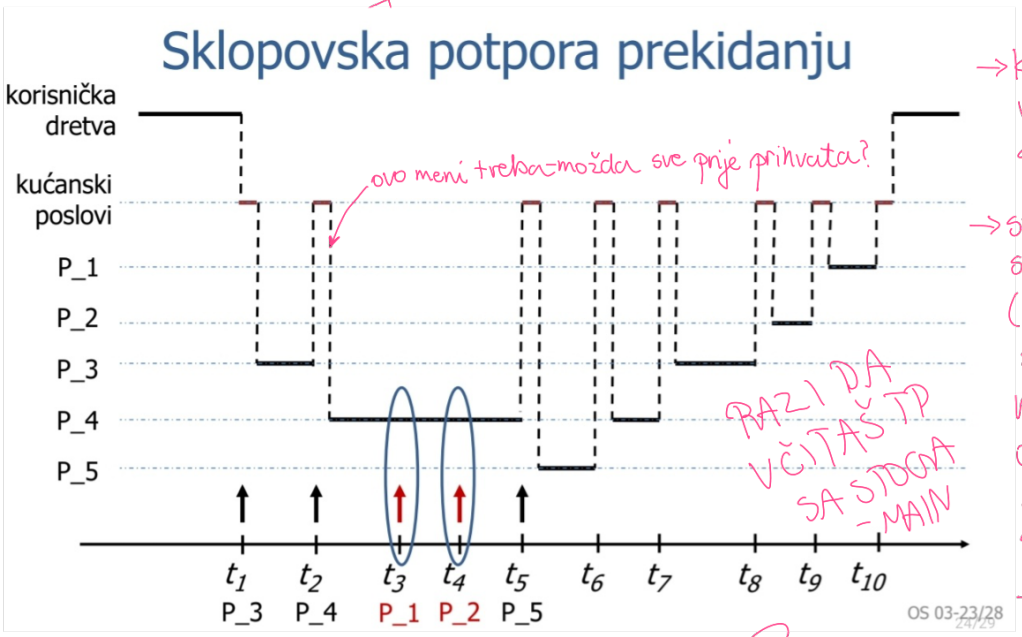
Koliko fja za obradu signala treba?
-samo jednu!
-registrirati pointer na istu fju - ona kao argument dobila redni broj signala
-ispisi zeljene promjene
-switch petlja s 3 potprograma: for petlja sa sleep po 1 sek za mjerenje obrade(moze i 1 potprogram - izgubi 10s zivota)

npr. kill sigint 74 – razina 3
kill sigterm 74 – razina 1

2
3
1

- prekid razine 2 : kill sigusr1 -10
- prekid razine 1 : kill sigterm -15
- prekid razine 3 : kill sigint -2

sigkill - nasilna smrt!



RAZLIKA VEZI TAŠTP SA STOGA - MAIN

→kad se dogodi prekid, vidi hoćeš li ga obraditi ili samo zabilježiti

→sve dok ima zahtjeva ili dok stog nije prazan (imam zastavice za obraditi ili sam nešto prekinula pa trebam nastaviti): ustanovi što ćeš obraditi: -ako je vrijednost sa stoga veća nego zastavica – nastavi obradu
-ako je veće g prioriteta zastavica, ne diraj stog već ispiši: sklop: tp mijenja stanje

→kad nemam zastavica više i stog je prazan – povratak u main

*dodatna provjera: stog nije prazan, nema zastavica: ako je 0 na stogu – povratak u main!