1. Signali (u UNIX-u)

1.1. Općenito o signalima u UNIX-u

- signale primaju procesi, šalje ih jezgra, iz svojih razloga ili kao posrednik
- signale upućuje (izvor signala):
  1. jezgra OSa (npr. prilikom greške u pristupu memoriji – `SIGSEGV`)
  2. proces sam sebi (npr. nakon isteka vremena `SIGALRM`)
  3. drugi proces (preko sučelja OS-a, npr. funkcijom `kill`)
  4. korisnik (`Ctrl+C`, preko sučelja OS-a, naredbom `kill` iz ljuske)
- analogno prekidnim signalima na razini procesora (sklopovlja), signali su mehanizam na razini OS-a (primaju ih procesi)
- kao i prekidi, signali služe za obradu asinkronih pojava – proces radi nešto drugo kad mu se signal uputi
- signali se mogu ignorirati (ne svi) ili su povod akciji
- proces može reagirati na signal na sljedeće načine:
  1. prihvatiti i obraditi signal pretpostavljenom funkcijom (npr. prekid izvođenja na SIGINT Ctrl+C)
  2. prihvatiti i obraditi signal zadanom funkcijom (zadana u programu, npr. sa `sigaction`)
  3. zadržati signal (za eventualnu kasniju obradu) (NE RADI u Pythonu)
  4. ignorirati signal
- izuzetak od gornjeg pravila je signal `SIGKILL` (broj 9) koji uništava proces
- signal ne sadrži nikakve dodatne informacije (osim u nekim proširenjima)
- pri prihvatu signala ponašanje procesa za taj signal se mijenja u "zadržati signal" dok se on ne obradi; kad obrada završi isti se signal ponovno može prihvatiti na isti način (slično kao kod prihvata prekida: prvo se zabranjuje daljnje prekidanje...)
- OS pamti samo po jedan signal pojedinog tipa za svaki proces – samo jedan signal istog tipa može biti na čekanju. Npr. ako se trenutno obrađuje `SIGINT` i za vrijeme njegove obrade dođe još 5 signala `SIGINT`, samo će se jedan zapamtiti i kasnije obraditi.
- paziti kako koristiti `sleep` kojega budi bilo koji signal, a vraća broj neprospavanih sekundi
  - `sleep(x)` ⇒ `for (i = 0; i < x; i++) sleep(1);`
- UNIX ima oko 30-tak signala (simbolička imena nalaze se u "signal.h"), neki od njih:
  - `SIGINT` – `Ctrl + C`; `SIGQUIT` – `Ctrl + \ (ž)`; `SIGTSTP` – `Ctrl + Z`
  - `SIGALARM`, `SIGTERM`, `SIGKILL`
  - `SIGUSR1`, `SIGUSR2`
- (info) MS Windowsi ne ponašaju se isto za signale (koriste nešto drugo)

1.2. Postavljanje ponašanja za signal (maskiranje signala)

Sučelje `sigaction`

- postoji više sučelja, u nastavku je prikazan samo `sigaction`
- za prihvat signala prvo treba drefinirati funkciju za obradu signala oblika:
  - `void obrada(int sig){ obradi signal; }`
- potom pozivom `sigaction` povezati signal s tom funkcijom
  - `sigaction(SIGXXXX, &act, NULL)`

Primjer rada sa signalima

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

void obrada ( int sig )
{
    obradi signal;
}

int main ()
{
    struct sigaction act;
    act.sa_handler = obrada; //SIG_DFL, SIG_IGN
    act.sa_flags = 0;
    sigemptyset ( &act.sa_mask );

    sigaction(SIGINT, &act, NULL); //maskiraj SIGINT - Ctrl+C

    radi nešto korisno; //to se prekida signalom

    return 0;
}
```

Privremeno blokiranje/dozvoljavanje prekidanja sa signalima (u funkciji obrade)

- dok je program u obradi signala novi isti takav signal neće prekinuti obradu
- ipak, ako se želi da se u dijelu koda obrade ponovno prihvaća isti signal (kao u algoritmu za programski prihvat prekida prema prioritetu) onda se to može omogućiti tako da se u kod funkcije obrade doda:

```
void obrada(int sig) { //funkcija registrirana za SIGINT
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    ...
    sigprocmask(SIG_UNBLOCK, &set, NULL); //dozvoli prekidanje sa SIGINT
        nekritični dio koda; //može se prekidati novim signalom SIGINT
    sigprocmask(SIG_BLOCK, &set, NULL); //ponovno blokiraj signal
    ...
}
```

- u `set` se mogu dodati i drugi signali ako je i njih potrebno blokirati/odblokirati
- slično je moguće postići i za drugi signal dok smo u obradi prvog, kombinacijom `SIG_BLOCK` i `SIG_UNBLOCK` i obratno.