Jelzések és kezelésük

A jelzés egy nagyon egyszerű kommunikációs forma, az információ egy speciális egész szám, amelyet úgy is szoktak nevezni, hogy a jelzés típusa/értéke (signum). Jelzést általában a rendszer szokott küldeni valamilyen hiba vagy esemény bekövetkezésekor (pl. érvénytelen memóriahivatkozás, 0-val osztás, gyerek folyamat terminált stb.), de jelzést küldhetnek a folyamatok is egymásnak. Jelzések segítségével kommunikálhatnak egymással a folyamatok, vagy kommunikációjuk összehangolására, szinkronizációra is fel tudják használni a jelzéseket.

Ha egy folyamat jelzést kap, akkor a következőképpen reagálhat:

1. Elkapja a jelzést és végrehajt egy jelzéskezelő függvényt (kivéve SIGKILL és SIGSTOP)
2. Figyelmen kívül hagyja (ignore) a jelzést (kivéve SIGKILL és SIGSTOP).
3. A jelzéshez tartozó alapértelmezett (default) művelet hajtódik végre.

A jelzésértékekhez SIG kezdetű neveket definiáltak (makro), amely elnevezések utalnak a felhasználási módjukra (pl.: SIGTERM = terminálj). A jelzésértékekről, alapértelmezett működésükről a signal(7)-es man oldalon tájékozódhatunk, lássunk ebből egy részletet:

```
The  entries  in the  "Action" column of the table specify the default action
for the signal, as follows:

Term   Default action is to terminate the process.

Ign    Default action is to ignore the signal.

Core   Default action is to terminate the process and dump core.

Stop   Default action is to stop the process.

First the signals described in the original POSIX.1 standard.

Signal     Value     Action   Comment
 ----------------------------------------------------------------------
SIGHUP       1       Term    Hangup detected on controlling terminal
                                    or death of controlling process
SIGINT       2       Term    Interrupt from keyboard
SIGQUIT      3       Core    Quit from keyboard
SIGILL       4       Core    Illegal Instruction
SIGABRT      6       Core    Abort signal from abort(3)
SIGFPE       8       Core    Floating poin exception
SIGKILL      9       Term    Kill signal
SIGSEGV     11       Core    Invalid memory reference
SIGPIPE     13       Term    Broken pipe: write to pipe with no readers
SIGALRM     14       Term    Timer signal from alarm(2)
SIGTERM     15       Term    Termination signal
SIGUSR1  30,10,16    Term    User-defined signal 1
SIGUSR2  31,12,17    Term    User-defined signal 2
SIGCHLD  20,17,18    Ign     Child stopped or terminated
SIGCONT  19,18,25            Continue if stopped
SIGSTOP  17,19,23    Stop    Stop process
SIGTSTP  18,20,24    Stop    Stop typed at tty
SIGTTIN  21,21,26    Stop    tty input for background process
SIGTTOU  22,22,27    Stop    tty output for background process
```

A fenti táblázatból látszik, hogy ha a SIGTERM jelzést, nem kezeljük le, és nem hagyatjuk figyelmen kívül (ignore), akkor alapértelmezésként (Term) a folyamatot terminálja a rendszert, míg a SIGCHLD jelzés figyelmen kívül lesz hagyva (Ign) ugyanilyen esetben. A SIGKILL és SIGSTOP jelzések esetén, mindig az alapértelmezett működés fog életbe lépni, nem lehet kezelni vagy figyelmen kívül hagyni őket.

A jelzéskezelő függvényt a rendszer hívja meg a jelzés beérkezésekor, ezért meghatározott a függvény prototípusa. Egy paramétere van, amely megegyezik az elkapott jelzés értékével és nincs visszatérési értéke. Tehát a jelzéskezelő függvények típusa: *void f(int signum)*.
A *signal* nevű függvénnyel tudjuk beállítani, hogy adott jelzésérték esetén a folyamat miként reagáljon a jelzésre. A függvény meghívásakor a rendszer eltárolja, hogy mi a teendő, ha a folyamat a későbbiekben kap egy adott jelzést. Megadhatjuk a *signal* függvénynek a jelzéskezelő függvény címét, ha szeretnénk elkapni és kezelni a jelzést.

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

Paraméterek:

- signum: a jelzés értéke
- handler: a következők egyike lehet
  - a jelzéskezelő függvény címe (ha elkapni és kezelni akarjuk)
  - SIG_IGN: figyelmen kívül hagyás (ignore)
  - SIG_DFL: alapértelmezett műveletet (default)

Visszatérési érték:

- jelzéskezelő függvény címe vagy hiba esetén SIG_ERR

Hibák:

- EINVAL: érvénytelen jelzésérték

A következő példában megpróbálunk szegmens hibát előidézni, és elkapni a rendszer által küldött SIGSEGV jelzést.

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


void kezelo(int signum){
    switch(signum){
      case SIGSEGV: puts("szegmens hiba"); break;
      default: puts("egyéb");
    }
    exit(1);
}


int main(){
    int *p;
    signal(SIGSEGV,kezelo);
    p[3] = 3;
    return 0;
}
```

A jelzéskezelő függvény

A jelzéskezelő függvény bejegyzése. Első paraméter a jelzésérték, második a jelzéskezelő címe.

Itt probálunk szegmenshibát okozni.

A példában jelzéskezelő eljárás egy exit függvényt hívott, amely a folyamatot terminálta. Ha a jelzéskezelő eljárás nem kiugrással ér véget (*return*), akkor oda térünk vissza, ahonnan „kiugrottunk” (néhány függvénytől eltekintve majdnem oda, lásd később), tehát a szegmenshibát okozó memóriahivatkozáshoz. Ezt megismételve persze újra szegmens hibát kapunk.
Ha többször is meghívjuk a *signal* függvényt ugyan azzal a jelzés típussal, akkor a rendszer az utolsó hibátlan hívás eredményét tárolja el. Nézzünk erre egy példát, amelyben kipróbáljuk a SIGINT jelzés esetén

alkalmazható háromféle működési módot. A Ctrl+C billentyű leütés hatására a parancsértelmező folyamat küld egy SIGINT jelzést a folyamatnak.

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

void kezelo(int signum){
    puts("signal kezelő");
}

int main(){

    signal(SIGINT,kezelo);
    puts("SIGINT kezelés");
    sleep(5);

    signal(SIGINT,SIG_IGN);
    puts("SIGINT ignore");
    sleep(5);

    signal (SIGINT, SIG_DFL);
    puts("SIGINT default");
    sleep(5);

    return 0;
}
```

Az indulás után a folyamat bejegyezteti a jelzéskezelő függvényt. Ha 5 másodpercen belül kapunk SIGINT jelzést, akkor a kezelő lefut, majd a folyamat folytatja a működést ott, ahol abbahagyta (majdnem ott, mivel nem várja ki a sleep által beállított 5 másodpercet).

Ezután a SIG_IGN konstans segítségével tudatja a rendszerrel, hogy figyelmen kívül akarja hagyni a SIGINT jelzést. Az ezután leütött Ctrl+C billentyű valóban figyelmen kívül lesz hagyva.

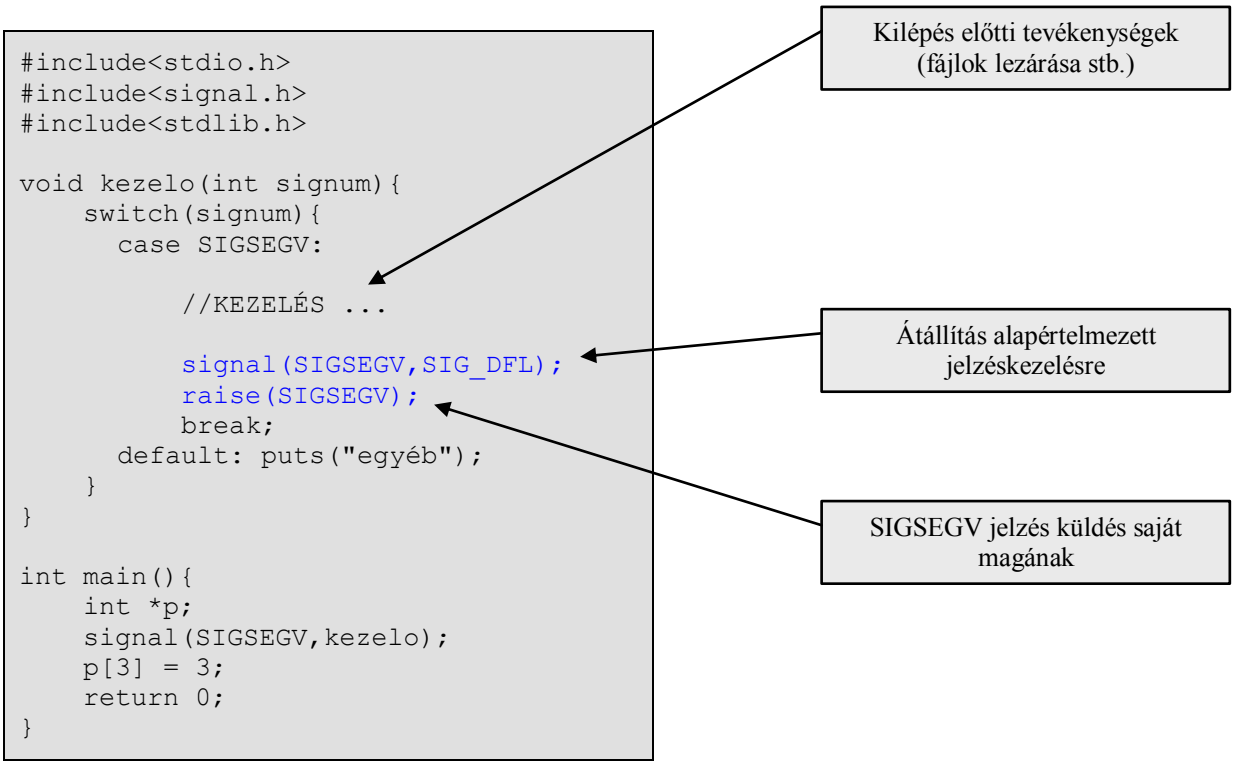
Végül a SIG_DFL konstans segítségével az alapértelmezett működésre váltunk. Ekkor leütött Ctrl+C billntyű hatására véget ér a folyamat.

Blokkolt folyamatok esetén, ha megszakíthatatlan a blokkolás, akkor rendszer visszatartja a jelzést, amíg a folyamat nem lesz aktív, megszakítható blokkolás esetén a rendszer „felébreszti” a folyamatot, hogy kezelhesse a jelzést (pl.: beolvasás terminálról).
Amikor egy jelzést kezelünk közben az illető jelzés blokkolva van, u.i. problémát okozhat, ha egy befejezetlen jelzéskezelés közben újra indulna a kezelő. Ez a blokkolás nem a figyelmen kívül hagyó művelet (SIG_IGN), hanem a rendszer a háttérben kezel egy blokkolt jelzések halmazát, amelyet a felhasználói folyamatok is elérhetnek, módosíthatnak (*sigprocmask*). A jelzéskezelő végrehajtása után a rendszer a törli az illető jelzés blokkolását. A blokkolás alatt beérkezett ugyanolyan jelzésekből „egy darabot kap meg a folyamat” a blokkolás feloldása után. A SIGKILL és SIGSTOP jelzéseket nem lehet blokkolni.

Jelzéskezelő eljárások készítése során, a jelzéskezelőből való visszatérésre többféle szokásos módszer terjedt el:

1. A jelzéskezelő eljárásban kilépünk a programból.
2. A jelzéskezelő eljárás végrehajtása után visszatérünk oda, ahonnan kiugrottunk.
3. A jelzéskezelő eljárásból a folyamat egy korábbi pontjába ugrunk vissza.

1. A jelzéskezelő eljárásban kilépünk a programból. Ha a jelzés alapértelmezett működése a terminálás, akkor nem az *exit* függvénnyel szokás kilépni, hanem a folyamat átállítja a jelzéskezelést alapértelmezett működésre és küld magának (*raise*) egy jelzést. Ekkor a szülő folyamat értesül róla (*return status*), hogy (milyen) jelzés szakította meg a folyamatot.



2. A jelzéskezelő eljárás végrehajtása után visszatérünk oda, ahonnan kiugrottunk. Ha a jelzéskezelő eljárás nem a programból való kilépéssel ér véget (*exit*), akkor általában oda térünk vissza, ahonnan „kiugrottunk”.
   Vannak olyan függvények, amelyek végrehajtása közben beérkezett jelzés esetén, a függvény már nem folytatható és a függvény EINTR hibával ér véget. Ilyenkor a jelzéskezelő eljárás lefutása után, a végrehajtás a függvény „után” fog folytatódni. Ezen függvények hibalistájában általában szerepel az EINTR hiba (pl.: read, write, vagy a korábbi példában látott *sleep* függvény, amely a beérkezett jelzés hatására EINTR hibával véget ér és nem várja ki a megadott időtartamig még fennmaradó időt.) Ezeknél a függvényeknél (*open, read, write* stb.) általában azt szeretnénk, hogy a függvény újra végrehajtódjon. Ezt megtehetjük, hogy addig ismételjük ciklusban, amíg a sikertelen a végrehajtás, és a hiba oka EINTR. Erre szolgál a TEMP_FAILURE_RETRY makro:

```
#ifndef TEMP_FAILURE_RETRY
#define TEMP_FAILURE_RETRY(expression) \
(__extension__ \
({ long int __result; \
do __result = (long int) (expression); \
while (__result == -1L && errno == EINTR); \
__result; }))
#endif


Int main(){

…


TEMP_FAILURE_RETRY(read(fd,buffer,sizeof(buffer)));

…

}
```

3. A jelzéskezelő eljárásból a folyamat egy korábbi pontjába ugrunk vissza. Ezt megtehetjük a *siglongjmp* függvénnyel, amelynek segítségével visszaugorhatunk a *sigsetjmp* függvény által elmentett környezetbe. Olyan függvénybe persze nem léphetünk vissza, amelyik már véget ért.

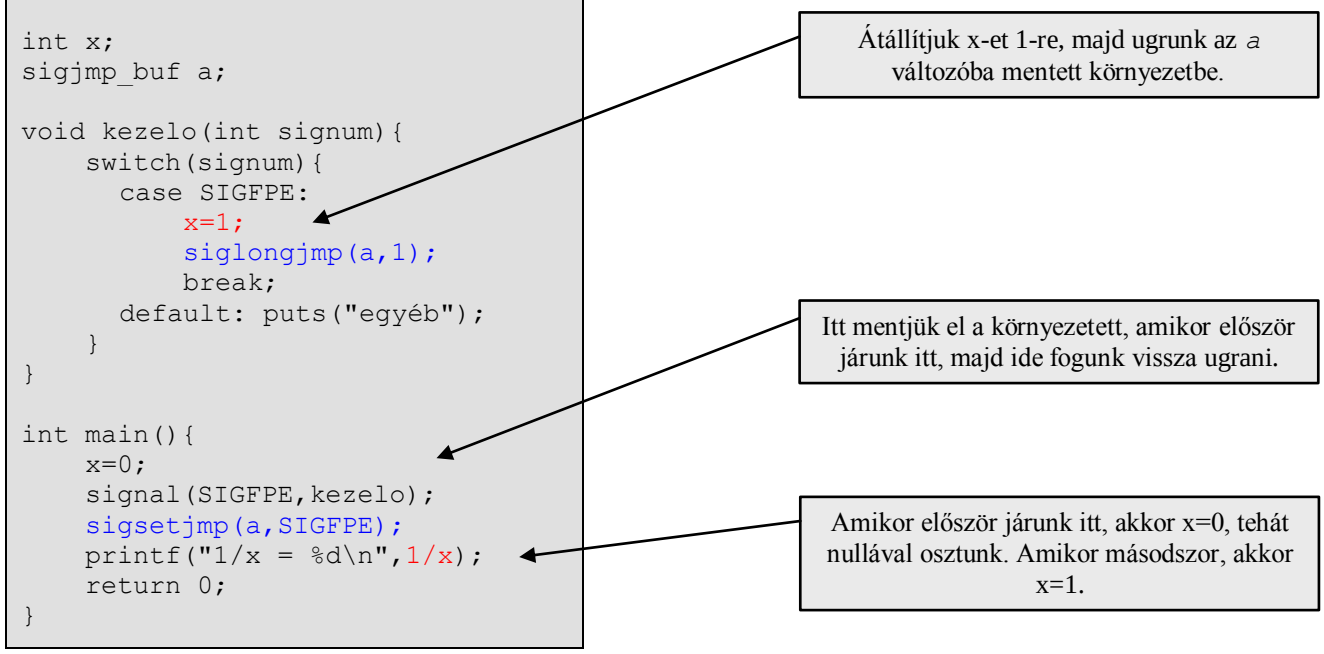
A következő példában 0-val osztunk, aminek következtében SIGFPE jelzést kapunk. A jelzéskezelőben átállítjuk a változó értékét 1-re, majd visszaugrunk az osztás elé és újra elvégezzük.

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

int x;
sigjmp_buf a;

void kezelo(int signum){
    switch(signum){
      case SIGFPE:
          x=1;
          siglongjmp(a,1);
          break;
      default: puts("egyéb");
    }
}

int main(){
    x=0;
    signal(SIGFPE,kezelo);
    sigsetjmp(a,SIGFPE);
    printf("1/x = %d\n",1/x);
    return 0;
}
```



(Ha *longjmp*-vel ugrunk ki a jelzéskezelőből, a jelzés blokkolását a rendszer nem oldja fel. Ezt nekünk kell megtenni a *sigprocmask* függvénnyel. A *siglongjmp* esetén erre nincs szükség.)

Jelzések küldése

Jelzést küldeni a *kill* függvénnyel tudunk. Nem minden folyamatnak küldhetünk jelzést, a küldőnek rendszergazdaként kell futnia, vagy a küldő valódi felhasználó azonosítójának meg kell egyeznie a címzett felhasználó azonosítójával.

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int signum);
```

Paraméterek:

- pid: címzett folyamat(ok) meghatározására szolgáló paraméter
  - ha pid>0: folyamat, amelynek azonosítója=pid
  - ha pid=0: hívó folyamattal azonos csoportban (process group) lévők
  - ha pid=-1: minden folyamat, akinek jogosult küldeni (kivéve init)
  - ha pid<-1: olyan folyamatok, amelyek csoportazonosítója=-pid
- signum: a jelzés értéke
  - ha signum=0, akkor nem küld szignált, hanem teszteli, hogy létezik-e a folyamat vagy a csoport (sikeres visszatérési érték esetén létezik)

Visszatérési érték:

- siker esetén 0
- hiba esetén -1

Hibák:

- EINVAL: érvénytelen szignál típus/érték
- EPERM: nincs jogosultságunk a szignált küldeni a folyamatnak
- ESRCH: a pid folyamat nem létezik

A következő példában a standard inputról beolvasott pid-del rendelkező folyamatnak próbálunk meg SIGTERM szignált küldeni, majd a *perror* függvény segítségével meggyőződünk a sikerességéről.

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

int main(){
    int pid;
    printf("pid: "); scanf("%d",&pid);
    kill(pid,SIGTERM);
    perror("kill");
    return 0;
}
```

Lássunk még néhány hasznos függvényt:

- `raise`: szignált küld a saját magának
- `pause`: szignál beérkezéséig várakozik
- `sigprocmask`: blokkolt szignálok kezelése
- `sigaction`: szignálok kezelése (többet tud, mint a *signal* függvény)

Javasolt feladatok:

1. Írjon programot, amely *fork*-kal létrehoz egy gyerek folyamatot. A gyerek folyamat másodpercenként írja ki a számokat 0-tól 10-ig, a szülő várjon 3 másodpercet, majd küldjön egy SIGTERM jelzést a gyereknek, majd újabb 3 másodperc múlva egy SIGKILL jelzést. A gyerek folyamat kezelje le a SIGTERM jelzést (írja ki, hogy "SIGTERM jelzést kaptam").
2. Írjon programot, amely az indítási paraméterként (*argv[1]*) megadott nevű folyamatoknak küld SIGTERM jelzést.

Összefoglalás:

- jelzéskezelő függvény: *void kezelo(int signum);*
- jelzéskezelő bejegyzése: *signal*
- jelzés küldése: *kill*
- Egy folyamat a jelzésre a következőképpen reagálhat:
  1. Elkapja a jelzést és végrehajt egy jelzéskezelő függvényt.
  2. Figyelmen kívül hagyja a jelzést (SIG_IGN).
  3. A jelzéshez tartozó alapértelmezett művelet hajtódik végre (SIG_DFL).
- Az alapértelmezett működés a jelzés értékétől függ: term, ignore, stop, continue.
- A SIGKILL és SIGSTOP jelzések esetén, mindig az alapértelmezett működés fog életbe lépni, nem lehet kezelni, figyelmen kívül hagyni, és nem lehet blokkolni.
- A jelzéskezelő futása során az adott jelzés blokkolva van, nehogy a kezelő újrainduljon (ezt a blokkolási listát a folyamat maga is tudja kezelni a *sigprocmask* függvénnyel).
- Jelzéskezelő eljárások készítése során, a jelzéskezelőből való visszatérésre többféle szokásos módszer terjedt el:
  1. A jelzéskezelő eljárásban kilépünk a programból. (exit helyett SIG_DFL-re visszaállunk, majd *raise*)
  2. A jelzéskezelő eljárás végrehajtása után visszatérünk oda, ahonnan kiugrottunk (néhány függvény esetén ne feledkezzünk meg TEMP_FAILURE_RETRY makróról).
  3. A jelzéskezelő eljárásból a folyamat egy korábbi pontjába ugrunk vissza (*sigsetjmp, soglongjmp*).