Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

# Signals

**Goal:** We are going to learn about signals. We can send a signal to a process from code (as the operating system send a signal in case of division by zero etc. or as we do when we press CTRL+C at the command line). The receiving process can act differently: it can stop, use a handler function, ignore the signal or block it. We can send some data through a signal.

**We learn about***:* SIGKILL, SIGTERM, SIGUSR1, etc. – signals; *signal, sigaction – connect a signal and a handler function (include signal.h, ); pause, sigsuspend – waits for a signal (include unistd.h, signal.h); kill, sigqueue – send a signal (include signal.h, sys/types.h); sigprocmask – blocks signals (include signal.h); alarm – timers (include unistd.h)*

## Tasks

1. Write a C program which contains only an infinite loop and execute it in the background! Send a SIGTERM signal from the command line! (`kill -SIGTERM PID number`, the PID number is given by `ps aux`) (*Please, check the result. What happens?* )

2. Modify the program with a handler function and connect the handler function to SIGTERM signal. Compile it and execute it in the background again. Send the SIGTERM signal from the command line – as before. (*Please, check it again! The program does not stop! Stop it with sending SIGKILL!)*

```
void handler(int signumber)
//handler function signature : one signal number

sighandler_t signal(int signum, sighandler_t handler);
// signum – the number of the signal, handler – the handler function binded to
the signal
```

3. Write a C program which has got a child process as well. The task of the child process is to send a SIGTERM signal from code. The parent should wait for it and write out the fact. Try it with and without signal handler function! (*Use kill function to send a signal in child and pause function in the parent. Rearly child sends the signal earlier than the parent executes pause, the pause will wait for the signal forever because it has been arrived already! The simpliest solution is to use a sleep, however it is not a really nice solution.)* Complete the program with an „answer" signal from the parent to the child!

```
int kill(pid_t pid, int sig);
//pid – the process to send the signal, sig – the signal to be sent

int pause(void);
//waits for an arriving signal
```

4. Modify the program, instead of pause function which may cause problems sometimes (described above), use a while cycle. The signal-handler should change the value of a variable and this value will be the condition of the while cycle…

5. Write a program with two child processes. Each of the child processes should create random numbers between 1 and 50. When the random number is less than 10 then the child process sends a signal to the parent and then stops. The parent waits both of them and at the end it stops too!

6. Write a program with two child processes. The first child should send a signal to the parent. The parent receiving it writes the fact to the screen and send another signal to the second child. The second child waits for it, writes out when it gets the signal and stops.

*Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government*

7. Modify the above written program and ignore SIGTERM signal. The child process should send a SIGTERM first and then send a SIGKILL too! *(Check that SIGTERM will not arrive at all!)*

8. Modify the program and block SIGUSR1 signal. The parent should wait for a signal and after the signal arriving unblock SIGUSR1. The child first should send a SIGUSR1 signal and after it a SIGTERM too. *(Check the result! The SIGUSR1 signal will not arrive, because it is blocked. After receiving the SIGTERM signal, the parent unblock SIGUSR1 and it will arrive also after it.)*

```
int sigemptyset(sigset_t *set);
//initialize the signal set
int sigfillset(sigset_t *set);
//initialize with full set of signals
int sigaddset(sigset_t *set, int signum);
//gives a signal (signum) to the set
int sigdelset(sigset_t *set, int signum);
//deletes a signal (signum) from the set
int sigismember(const sigset_t *set, int signum);
//is the signal (signum) member of th eset?
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
//block or unblock arriving signals with a sigset
//how: SIG_BLOCK - blocks, SIG_UNBLOCK - unblock
```

9. Write a C program which implements a digital clock by using a timer! (*Use alarm function and send a SIGALARM in each second. The handler function should write out the time! Do not use sleep and alarm function in the same program whilest the implementation of sleep is based on alarm function! Later we shall learn about other timer implementations as well! )*

```
unsigned int alarm(unsigned int seconds);
// second - the timer alerts after seconds sec
```

10. Write a C program in which child process sends a number through SIGTERM signal to the parent. (*You should use sigqueue instead of simple kill, sigaction function instead of signal function! Remember that you can get the PID number of the sending process too! )*

```
int sigqueue(pid_t pid, int sig, const union sigval value);
//send a signal (sig) to process (pid) plus a value to the process

int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
//binds the signal (sig) to sigaction struct (act) and remember the old value in
(oldact)

struct sigaction {
               void     (*sa_handler)(int);
               void     (*sa_sigaction)(int, siginfo_t *, void *);
               sigset_t   sa_mask;
               int        sa_flags;
               void     (*sa_restorer)(void);
           };
```