Chapter 5
Signals

Class Syllabus

5.1) Basic Knowledge

1. How processes react to signals
2. Signals List and behaviours

5.2) Sending a signal

5.3) Signals and Processes

5.4) Masks

1. Affecting signal masks
2. Examples

5.5) Blocking a signal

1. *sigprocmask*
2. Example

5.6) Functions pointers

1. Example

5.7) Signal Handler

1. *sigaction*
2. Example

Basic Knowledge

- Same commands you saw in shell (*kill, trap*)
- **Initial Goal:** to inform a process about a particular event occuring in the machine/for the process
- Some signals are sent by the kernel, but users can send them too
- We can only send signals to the processes we own (except **root** of course!)

How process react to signals

When *P* Process receives the *S* signal:

if *S is masked* **then**

S is added to the list of arrived and ignored signals

else

if *S is not intercepted by P* **then**

S default action

else

Normal execution of *P* is halted, executing position in the code is memorized

Execution of the function of interception installed in *P*

Execution resumes where *P* was interrupted

POSIX signals list (1/3)

- Possible actions for signals are:
  1. **TERM**: Terminates the process
  2. **CORE**: Terminates the process and produce a *dump core file*
  3. **IGN**: Ignores the signal
  4. **STOP**: Stop the process
  5. **CONT**: Continue the process if currently stopped

- Not all the signals can be trapped => their default action can't be modified (* in the charts)

POSIX signals list(2/3)

Signal Name	Associated event	Default action
SIGHUP	Death of controlling process	1
SIGINT	Interrupt from keyboard(^C)	1
SIGQUIT	Quit from keyboard(^\ or ^4)	2
SIGILL	Illegal instruction detected	2
SIGABRT	Illegal termination coming from the *abort()* command	2
SIGFPE	Floating-point exception (arithmetical)	2
SIGKILL	Termination signal (*kill()*)	1*
SIGSEGV	Invalid memory reference	2
SIGPIPE	Broken pipe : write to pipe with no readers (*pipe()*)	1
SIGALRM	Timer signal from *alarm()*	1
SIGTERM	Termination signal	1

POSIX signals list(3/3)

Signal Name	Associated event	Default action
SIGUSR1	User-defined signal 1	1
SIGUSR2	User-defined signal 2	1
SIGCHLD	Child stopped or terminated	3
SIGSTOP	Stop the process	4*
SIGSTP	Stop the process from keyboard(^Z)	4
SIGCONT	Continue the process if stopped	5
SIGTTIN	Terminal input for background process(reading)	4
SIGTTOU	Terminal output for background process(writing)	4
SIGIO	Characters to read are coming	1 or 3
SIGURG	Urgent condition on socket(read, write, etc.)	3
SIGTWINCH	Window resize signal	3

Sending a signal

- #include <sys/types.h> /* for pid_t */
  #include <signals.h>

- int kill(pid_t pid, sig sig);
- Send the *sig* signal to the *pid* process
- Destination process needs to have the same UID as the source (except if UID source is 0)

- **Shell:** *kill(1)*
  - kill [-signal | -s signal] pid …
  - kill -USR1 76431 76534

Signals and processes

- One signal managing table for each process
- One default behaviour (death in general)
- Behaviour of a process when a signal is received can be modified => interception and handling
- Some signals can't be intercepted (*SIGKILL*)

Signal number	Pending Bit (called bit)	Mask Bit	void(*handling)()
SIGQUIT	0	1	dies_in_peace
SIGALRM	1	0	say_hello_there
SIGKILL	0	1	default
…	…	…	…

Affecting signal masks

- Mask (*sigset*): to modify a signal behaviour
- **int** sigemptyset(**sigset_t** *ens); /* array = {} */
- **int** sigfillset(**sigset_t** *ens); /* array = {1, 2, …, NSIG} */
- **int** sigaddset(**sigset_t** *ens, **int** sig); /* array = array + {sig} */
- **int** sigdelset(**sigset_t** *ens , **int** sig); /* array = array – {sig} */
- All of those functions return 0 if no error is encountered, else -1

Usage examples

- **int** sigismember(**const sigset_t** *ens, **int** signum);
  - Returns 1 if *signum* is contained in *set*
  - Returns 0 if *signum* is not contained in *set*
  - Returns -1 if an error is encountered

- **int** sigsuspend(**const sigset_t** *mask);
  - Temporary replaces the mask with the *mask* value, and stop the process until of the non-ignored signal is sent to it
  - Returns -1 if an error is encountered

- **int** sigismember(**sigset_t** *set);
  - Writes in *set* the list of currently blocked signals
  - Returns 0 if OK
  - Returns -1 if an error is encountered

Blocking one (or more) signal

- **int** sigprocmask(**int** how, **const sigset_t** *set, **sigset_t** *oldset);
- *sigprocmask* enables to block (temporary or not) signals

- When the concealment (=blocking) is lifted, previously blocked signals can be delivered (if they didn't disappeared yet)

- **How to**
  - **SIG_SETMASK:** replace *oldset* by *set*
  - **SIG_BLOCK:** block *set* signals AND *oldset* signals
  - **SIG_UNBLOCK:** unblock *set* signals
  - **If** *oldset* is not null => takes the value of the previous mask

- Returns 0 if OK, -1 if an error is encountered

Blocking a signal : example


```
sigset_t signals;
int sig;
/* signals = { SIGQUIT , SIGUSR1 } */
sigemptyset(& signals);
sigaddset(&signals, SIGQUIT);
sigaddset(&signals, SIGUSR1);
/* Mask installation */
sigprocmask (SIG_SETMASK, &signals, NULL);
/* The target */
sleep(20);
printf ("Signals sent while asleep: ");
sigpending (& signaux );
for(sig =1; sig < NSIG ; sig ++)
                if(sigismember(&signals, sig))
                                printf("%d ", sig );
puts ("\n");
/* Deblocage des signaux */
sigemptyset(&signals);
return sigprocmask(SIG_SETMASK, &signals, NULL);
```

Function pointers in C Programming

- **Prototype**:

```
double (*myFunction)(double, int);
```

- **Affection(with typing check)**:

```
double strength(double d, int n)
{ /* code of the strength function */ }

myFunction = strength;
```

- **Usage:**

```
resultat = (*myFunction)(1.5, 2);
```

Example of function pointer

- #include <stdlib.h>
  int atexit(void (* function)(void));
- *atexit(3)*
  - define a function to call just before the program's ending
  - *function:* "argument-less return-less function" argument type, pointing to the function to call

- **Example**

```
void the_end_is_coming() {
        puts("Goodbye Unforgiving World!");
}
int main() {
        atexit(the_end_is_coming);
        puts("Im Alive!");
        return 0;
}
```

struct sigaction (1/2)

- **struct** sigaction(

```
        void (*sa_handler)(int);
        void (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t sa_mask;
        sigset_t sa_mask;
}
```

- ***sa_handler*** or ***sa_sigaction***: function to run(or *SIG_DFL* or *SIG_IGN*)
- ***sa_mask***: list of blocked signals while the handler is executed
- ***sa_flags*** (examples):
  - **SA_NOCLDSTOP**: ignore **SIGCHLD** when the child dies
  - **SA_RESETHAND**: reactivate default handler (**NON POSIX**)

struct sigaction (2/2)

- **int** sigaction(**int** signum, **const struct** sigaction *act, **struct** sigaction oldact);

- ***signum:*** signal number to intercept

- if ***act*** is not null then
  act becomes the new sigaction

- if ***oldact*** is not null then
  old sigaction is written in ***oldact***

- Returns 0 if OK
- Returns -1 if an error is encountered

Example of signal handling

```
void hello(int sig) {
        printf("Hello there! , sig %d\n", sig );
}
int main() {
        struct sigaction action, save;
        sigemptyse ( action. sa_mask);
        action.sa_handler = hello;
        action.sa_flags = 0;
        if (sigaction(SIGQUIT, &action, &save )) {
                perror("Hello installation"); exit (1);
        }
        sleep(20);
        printf("End of break\n");
        if(sigaction(SIGQUIT, &save, NULL)) {
                perror("Signals restored"); exit (1);
        }
        return 0;
}
```