Lezione 5 - 12/4/21 (Segnali, Handler)

Segnali Unix

Ci sono vari **eventi** che possono avvenire in maniera **asincrona** al normale **flusso di un**
programma, alcuni dei quali in maniera **inaspettata** e non **predicibile**. Per esempio, durante l'esecuzione di un programma ci può essere una **richiesta di terminazione** o di **sospensione** da parte di un utente, la **terminazione** di un processo **figlio** o un **errore**
generico.

Unix prevede la gestione di questi eventi attraverso i **segnali**: quando il sistema operativo si accorge di un certo evento, genera un **segnale** da **mandare** al **processo** interessato il quale **potrà decidere** (nella maggior parte dei casi) **come comportarsi.**

- **SIGALRM** → alarm clock
- **SIGCHLD** → child terminated
- **SIGCOUNT** → continue, if stopped
- **SIGINT** → terminal interrupt, CTRL+C
- **SIGKILL** → kill process
- **SIGQUIT** → terminal quit
- **SIGSTOP** → stop
- **SIGTERM** → termination
- **SIGUSR1** → user signal
- **SIGUSR2** → user's signal (??????????????????????)

I segnali sono anche detti "software interrupts" perché sono a tutti gli effetti delle interruzioni del normale flusso del processo generato dal sistema operativo.

Come per gli interrupts, il programma può decidere come gestire l'arrivo di un segnale (presente nella lista pending):

- Eseguire l'**azione di default**
- **Ignorandolo** (non sempre possibile) → il programma **prosegue** normalmente
- Eseguendo un **handler personalizzato** → programma si **interrompe**

Liste segnali in Unix

- **Pending signals:** segnali emessi che il processo dovrà gestire
- **Blocked signals:** segnali non comunicati al processo

Default Handler

L'handler di default può:

- Ignorare il segnale
- Terminare il processo
- Continuare l'esecuzione (se il processo era in stop)
- Stoppare il processo

Ogni processo può sostituire

Ogni processo può **sostituire** il **gestore di default** con una **funzione "custom"** (**a parte**
per **SIGKILL** e **SIGSTOP**) e comportarsi di conseguenza. La **sostituzione** avviene **tramite** la system call **signal()** (definita in "**signal.h**")

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

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
void myHandler(int sigNum){
    printf("CTRL+Z\n");
}
void main(){
```

```
    signal(SIGINT,SIG_IGN); //Ignore signal
    signal(SIGCHLD,SIG_DFL); //Use default handler
    signal(SIGTSTP,myHandler); //Use myHandler
}
```

Esempio:

```
//sigCST.c
#include <signal.h>
#include <stdio.h>
void myHandler(int sigNum){
    printf("CTRL+Z\n");
    exit(2);
}
int main(){
    signal(SIGTSTP,myHandler);
    while(1);
}
```

```
//sigDFL.c
#include <signal.h>

int main(){
    signal(SIGTSTP,SIG_DFL);
    while(1);
}
```

```
//sigIGN.c
#include <signal.h>

int main(){
    signal(SIGTSTP,SIG_IGN);
    while(1);
}
```

Custom handler

Un **handler personalizzato** deve essere una **funzione di tipo void** che **accetta come argomento un intero**, il quale rappresenta il **segnale catturato**. Questo **consente** l'utilizzo di uno **stessa handler** per **segnali differenti.**

```
//param.c
#include <signal.h>
#include <stdio.h>
void myHandler(int sigNum){
    if(sigNum == SIGINT){
        printf("CTRL+C\n");
    }
    else if(sigNum == SIGTSTP){
        printf("CTRL+Z\n");
    }
    signal(SIGINT,myHandler);
    signal(SIGTSTP,myHandler);
}
```

Alcuni segnali:

Alcuni segnali:

Aa SIGXXX	description	default
SIGALARM	Alarm clock	quit
SIGCHLD	Child terminated	ignore
SIGCONT	Continue, if stopped	ignore
SIGINT	Terminal interrupt, CTRL+C	quit
SIGKILL	Kill process	quit
SIGSYS	Bad argument to syscall	quit with dump
SIGTERM	Software termination	quit
SIGUSR1/2	User signal 1/2	quit
SIGSTOP	Stopped	quit
SIGTSTP	Terminal stop, CTRL+Z	quit

Esempio:

```
//child.c
#include <signal.h>
#include <stdio.h>
```

```
#include <unistd.h>
#include <sys/wait.h>
void myHandler(int sigNum){
    printf("Child terminated!\n");
}
int main(){
    signal(SIGCHLD,myHandler);
    int child = fork();
    if(!child){
        return 0; //terminate child
    }
    while(wait(NULL)>0);
}
```

Inviare i segnali kill()

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

Invia un segnale ad uno o più processi a secondo dell'argomento pid:

- pid>0 → segnale al processo con PID=pid
- pid=0 → segnale ad ogni processo dello stesso gruppo
- pid=-1 → segnale ad ogni processo possibile (stesso UID/RUID)
- pid<-1 → segnale ad ogni processo del gruppo |pid|

Restituisce 0 se il segnale è stato inviato e -1 nel caso di errore.

```
//kill.c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

void myHandler(int sigNum){
  printf("[%d]: ALARM!\n",getpid());
}

int main(int argc, char **argv){
  signal(SIGALRM, myHandler);
  int child = fork();
  if(!child)  while(1)
  printf("[%d]: sending alarm to %d in 1 s\n",getpid(),child);
  sleep(1);
```

```
  kill(child,SIGALRM);
  printf("[%d]: sending SIGTERM to %d in 1s\n",getpid(),child);
  sleep(1);
  kill(child, SIGTERM);
  while(wait(NULL)>0);
}
```

Kill da bash

kill è anche un programma in bash che accetta come primo argomento il tipo di segnale (kill -l per la lista) e come secondo argomento il PID del processo.

```
//bash.c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void myHandler(int sigNum){
    printf("[%d]ALARM!\n",getpid());
    exit(0);
}
int main(){
    signal(SIGALRM,myHandler);
    printf("I am %d\n",getpid());
    while(1);
}
/*
./bash.out
#On new window/terminal
kill -14 <PID>
kill -l #per vedere la lista di segnali associata al numero
*/
```

Programmare un alarm: alarm()

```
unsigned int alarm(unsigned int seconds);
```

```
//alarm.c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
short cnt = 0;
```

```
void myHandler(int sigNum){
    printf("ALARM!\n");
    cnt++;
}
int main(){
    signal(SIGALRM,myHandler);
    alarm(5); //Set alarm in 5 seconds
    //Set new alarm (cancelling previous one)
    printf("Seconds remaining to previous alarm %d\n",alarm(2));
    while(cnt<1);
}
```

alarm permette di avviare un segnale **SIGALARM** in un numero specificato di secondi, se c'è già un alarm viene restituito dalla funzione il numero di secondi che mancavano per l'alarm prima di quello dichiarato.

Mettere in pausa: pause()

```
//pause.c
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

void myHandler(int sigNum){
  printf("Continue!\n");
}
int main(int argc, char **argv){
  printf("%d\n",getpid());
  signal(SIGCONT,myHandler);
  signal(SIGUSR1,myHandler);
  pause();
}
/*
./pause.out
kill -18/-10 <PID>
*/
```

Un processo messo in pausa riprende appena riceve il primo segnale, è stato creato un segnale a posta **SIGCONT** per lasciar il processo procedere ignorando il segnale.