

Uso dei segnali e gestione di processi

Si progetti un programma C che, utilizzando le primitive del sistema Unix, abbia un'interfaccia del tipo:

```
ESEMPIO prog_1 prog_2 .. prog_N
```

Il programma ESEMPIO quindi, può accettare da 1 a N argomenti (prog_i): ogni argomento prog_i rappresenta il nome di un file eseguibile.

Il programma ESEMPIO deve mettere in esecuzione in parallelo tutti i file prog_i passati come parametri; inoltre, deve consentire all'utente di provocare la terminazione forzata dell'esecuzione di un particolare file prog_i specificato interattivamente. Commentare e discutere le scelte adottate.

Soluzione:

L'esecuzione parallela degli N file eseguibili nel sistema Unix può essere realizzata mediante l'uso delle primitive fork ed exec.

In particolare, il programma esame chiamerà tante fork quanti sono i programmi da eseguire, creando quindi N processi figli, ognuno dedicato alla esecuzione di un particolare prog_i. Ciascun figlio, una volta creato, chiamerà a sua volta una system call della famiglia exec (ad esempio execl), mediante la quale sostituisce al suo codice quello del programma dato:

```
char nome[10][20];
int pid[10];

main(argc, argv)
int argc;
char **argv;
{
    int  retval, figlio, i;

    for (i=1;i<argc;i++) {
        strcpy(nome[i], argv[i]);
        pid[i]=fork(); /* creazione del figlio i-esimo*/
        if (pid[i]==0) { /*codice figlio i-esimo*/
            /*esecuzione del codice prog_i*/
            execlp(argv[i], (char *)0);
            exit(1);
        }
    }
    < codice del padre >
}
```

Una volta creati i processi e messi in esecuzione i relativi programmi, il problema da risolvere è la realizzazione del meccanismo di terminazione selettivo. Questo può essere efficacemente realizzato attraverso l'uso dei segnali.

Ad esempio, si può mandare il segnale SIGINT (ctrl-C, da tastiera) al processo padre. Una volta ricevuto il segnale SIGINT, il padre deve ricevere dall'utente l'indicazione relativa al figlio da uccidere.

Questo si realizza associando al segnale utilizzato (SIGINT) una funzione *trap* che provoca la lettura dallo standard input dell'informazione relativa al figlio da eliminare. La funzione trap deve avere la visibilità delle strutture dati nome (dove sono memorizzati i nomi dei programmi in esecuzione) e pid (dove sono contenuti gli identificatori dei processi figli): a tale scopo queste variabili sono state dichiarate esternamente al main.

```
void trap()
{
    char buff[20];
    int i, trovato=0;

    printf("\n Nome programma da terminare ? ");
    scanf("%s", buff);
    for (i=1;i<10;i++)
        if (!strcmp(buff, nome[i]))
            trovato=i;
    if (trovato!=0)
        kill(pid[trovato], SIGKILL);
    else
        printf("Nome non valido.\n");
}
```

Pertanto è necessario che il padre esegua una signal per associare la trap a SIGINT.

Inoltre, poiché i destinatari del segnale SIGINT sono tutti i processi associati al terminale utilizzato, è necessario che ogni figlio mascheri il segnale SIGINT: anche questa operazione si realizza con la system call signal (o la sigset), associando SIG_IGN al segnale dato.

Introducendo anche una fase di controllo dei parametri, la versione finale del programma è quindi la seguente:

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

char nome[10][20];
int pid[10];

void trap(signo)
int signo;
{
    char buff[20];
    int i, trovato=0;

    printf("\n Nome programma da terminare ? ");
    scanf("%s", buff);
    for (i=1;i<10;i++)
        if (!strcmp(buff, nome[i]))
            trovato=i;

    if (trovato!=0)
        kill(pid[trovato], SIGKILL);
    else
        printf("Nome non valido.\n");
}

main(argc, argv)
int argc;
char **argv;
{
    int  retval, figlio, i, stato;

    if (argc<2) {
        printf("Almeno 1 parametro\n");
        exit(1);
    }

    sigset(SIGINT, trap); /*associo trap a SIGINT */
    for (i=1;i<argc;i++) {
        strcpy(nome[i], argv[i]);
        pid[i]=fork();
        if (pid[i]==0) { /*codice figlio i-esimo*/
            sigset(SIGINT, SIG_IGN); /* ignoro SIGINT*/
            execlp(argv[i], (char *)0);
            perror("dopo execl");
            exit(1);
        }
    }
    for (i=1;i<argc;i++)
        wait(&stato); /* attesa dei figli */
}
```