Quarta Esercitazione

Gestione di processi e segnali in Unix

Stefano Monti
stefano.monti6@unibo.it

Esercizio 1

Si realizzi un programma C Unix che utilizzi le primitive per la gestione dei processi e dei segnali, con la seguente interfaccia di invocazione:

```
esercizio1 N
```

Il processo padre genera N figli fratelli

- I primi N/2 figli terminano l'esecuzione alla **ricezione** del **segnale SIGUSR1**
- I secondi N/2 figli **terminano dopo 5 secondi**
- Ciascun figlio, prima di terminare, stampa a video la ragione della terminazione

Punti di attenzione

- I figli non devono fare altro che attendere
  - i primi N/2 un segnale dal padre
  - i secondi N/2 un numero fissato di secondi
  - → attenzione alla scelta delle primitive

- Padre termina i figli mediante segnale SIGUSR1
  - È necessario gestire la memorizzazione dei pid per ottenere l'invio dei segnali ai figli?

- I figli devono terminare in maniera “gestita”
  - in questo caso, solo stampa a video
  - in generale, chiusura risorse (es. file aperti, ecc...)


```
int main(int argc, char *argv[]){
      int count;  int N = atoi (argv[1]);
      int pids [N];
      pids[0]=getpid();
      for (count=1; count <= N ; count ++){
            int pid = fork();
            pids[count]=pid;
            if (pid < 0){
                  perror("Errore nella fork"); exit(-1);
            }
            else if (pid == 0) {//Codice figlio parallelo
                  if (count <= N/2){
                        signal(SIGUSR1,handler_notifica_padre);
                        pause();
                  }
                  else{
                        sleep(5);
                        printf("Processo %d : terminazione dopo 5 secondi\n", getpid());
                  }
                  exit(0);
            }
      }
      for ( count = 1; count <= N/2; count++){
            kill(pids[count],SIGUSR1);
       }
      for ( count = 0; count < N; count++){
            wait_child();
       }
}
```

```
void handler_notifica_padre() {
      printf("Processo %d : ricevuto SIGUSR1 dal padre\n", getpid());
}
```


Esercizio 2

Si realizzi un programma C Unix che utilizzi le primitive per la gestione dei processi e dei segnali.

Il programma, a partire dal padre P0, deve generare una **gerarchia di 3 figli/nipoti**

Ciascun figlio/nipote invia al padre un segnale solo se il proprio pid è pari

Il processo padre deve stampare l'indicazione dei processi con pid pari.

Punti di attenzione

Un segnale sostanzialmente comunica un evento

- nessun contenuto informativo oltre al segnale
  - per passare contenuto informativo tra processi vedremo altre syscall
- al più, usando diversi segnali, è possibile distinguere il processo che ha originato il segnale

Per risolvere l'esercizio con i soli segnali

- ciascun figlio/nipote lancia un segnale diverso dagli altri
- il padre è in grado di sapere quale segnale sta ricevendo → capisce da quale processo è arrivato il segnale

PROBLEMA: quanti segnali sono a disposizione del programmatore?


```
int pid0;
int main(int argc, char *argv[]){
      pid0=getpid();
      generafiglio(2);
      signal (SIGTERM,handler_segnali);  signal (SIGUSR1,handler_segnali);
      signal (SIGUSR2,handler_segnali);
      wait_child();
}

void generafiglio(int current){
      sleep(2);  int pid;
      pid=fork();
      if (pid==0){
            if (current == 0){//terzo figlio
                  if ((getpid() % 2) == 0)
                        kill (pid0,SIGTERM);
            } else if (current == 1){//secondo figlio
                  if ((getpid() % 2) == 0)
                        kill (pid0,SIGUSR2);
                  generafiglio(current-1);
                  wait_child();
            } else if (current == 2){//primo figlio
                  if ((getpid() % 2) == 0)
                        kill (pid0,SIGUSR1);
                  generafiglio(current-1);
                  wait_child();
            }
            exit(0);
      } }
```

```
void handler_segnali(int signum){
      if (signum == SIGUSR1){
            printf("Padre: il primo figlio ha PID pari\n");
      }
      else if (signum == SIGUSR2){
            printf("Padre: il secondo figlio ha PID pari\n");
      }
      else if (signum == SIGTERM){
            printf("Padre: il terzo figlio ha PID pari\n");
      }

}
```


Esercizio 3

Si realizzi un programma, che, utilizzando le system call del sistema operativo UNIX, soddisfi le seguenti specifiche:

Sintassi di invocazione:

eseguiInSequenza **N COM1 COM2**

Significato degli argomenti:

- eseguiInSequenza è il nome del file eseguibile associato al programma.
- N e` un valore intero positivo.
- COM1 e COM2 sono stringhe che rappresentano il nome di un file (per semplicita`, si supponga che il direttorio di appartenenza del file COM sia nel PATH)

Specifiche

Il processo iniziale (P0) deve creare 2 processi figli (fratelli) P1 e P2.

I processi P1 e P2 devono eseguire rispettivamente i comandi COM1 e COM2, con il vincolo esplicito che **COM1 venga eseguito soltanto al termine della esecuzione di COM2**.

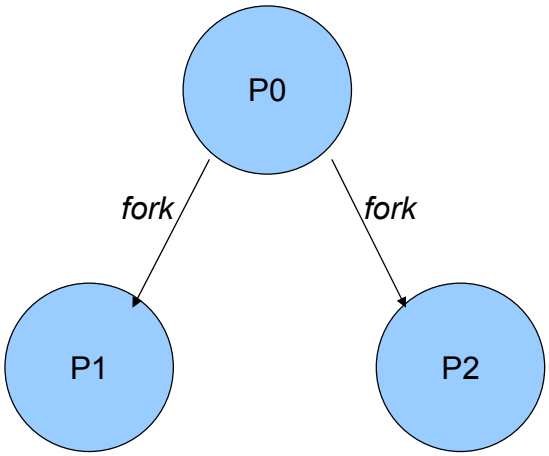
Considerando che i file COM1 e/o COM2 potrebbero non essere eseguibili, in caso di fallimento dell'esecuzione il processo figlio che ne è responsabile dovrà **notificare tale evento al padre P0** e successivamente terminare.

In caso di successo nell'esecuzione del comando di pertinenza, dopo N secondi dalla sua creazione, il processo figlio **P1** dovrà comunque **terminare la propria esecuzione**.

Note alla soluzione

- P0 crea P1 e P2: fork e **attende**
- P1 deve terminare dopo N secondi (qualunque sia il programma che esegue):
  - impostazione di timeout
  - gestione dei segnali & exec
- in caso di fallimento nell'esecuzione di COM(1,2) P(1,2) deve **notificare** l'evento a P0: invio di segnali e relativa gestione
- gestione della esecuzione "sequenzializzata" (COM2 prima di COM1):
  - P1 e P2 sono fratelli: quali strumenti a disposizione per sincronizzare P1 e P2?
  - può P2 eseguire COM2 e, in caso di successo, inviare un segnale a P1?

Schema di generazione (iniziale) dei processi



Variante

- Il padre P0, mentre il figlio esegue, continua la propria attivita` (cioe` non si pone in attesa di P1 e P2), per stampare il suo PID ripetutamente;
- tuttavia, al termine di P1 e P2, P0 dovra` tempestivamente raccogliere e stampare lo stato di terminazione dei figli (v. gestione del sigchld).