Soluzione di Compiti Unix

In particolare, vanno osservate le seguenti specifiche:

1. il processo **padre**, P0 legge da filein e comunica il contenuto di filein, **N** caratteri per volta, **agli altri due processi**;
2. degli altri **due processi figli**:
   - **P1** controlla che ciascun carattere ricevuto in ingresso sia diverso da Car. Nel caso riceva un carattere uguale a Car deve terminare l'esecuzione di tutti gli altri processi (P0 e P2), e successivamente terminare la propria esecuzione.
   - L'altro processo (**P2**) deve contare i caratteri ricevuti. Alla ricezione dell'M-esimo carattere, deve stampare un messaggio su standard output e quindi terminare la *propria* esecuzione.
3. In ogni caso, tutti i processi devono terminare la propria esecuzione una volta che non ci sono più dati da leggere (cioè, P0 quando ha finito di leggere da **filein**, P1 e P2 quando hanno finito di ricevere dati da P0).

Compito 2 luglio 2004 (A)

Si scriva un programma in C che, utilizzando le *system call* di Unix, preveda la seguente sintassi:

esame filein N M Car

dove:

· **filein e` il nome di un file esistente e leggibile.**

· **N e M sono due interi non negativi**

· **Car e` un carattere alfabetico**

Il processo iniziale (PO) deve generare due figli (P1 e P2).

Impostazione

- Trasmissione del contenuto di filein ad entrambi i figli -> P0 apre 2 pipe
- P1 puo` provocare la terminazione di P0 e P2 -> uso del segnale SIGPIPE, gestito da P0
- P2 puo` provocare la terminazione di se stesso: in questo caso P0 deve interrompere la comunicazione con P2 -> uso del segnale SIGPIPE , gestito da P0

Soluzione

```
/* appello del 2 Luglio 2004 tema A */

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

int p1[2], p2[2];
int OK1=1, OK2=1, scrittura;
int pid[2];
int fd;

void brokenP(int sig); /*gestore chiusura pipe (P0) */
```

```
main(int argc, char **argv)
{       int cont, M, N, k;
        char c, buff[80]; /* hp: N<= 80*/
        /* controllo sintassi */
        if (argc!=5)
        {       printf("sintassi !!\n");
                exit(-1);
        }
        N=atoi(argv[2]);
        M=atoi(argv[3]);
                signal(SIGPIPE, brokenP);
        pipe(p1);
        pipe(p2);
        pid[0]=fork();
```

```
  if (!pid[0]) /* processo P1: */
          {       close(p1[1]);
                  close(p2[0]);
                  close(p2[1]);
                  while(read(p1[0], &c, 1))
                   if(c==argv[4][0])
                   {   printf("P1: %c\n", c);
                       close(p1[0]);
                  /* ->invio di sigpipe a P0!!
                  ->terminazione di P2 e P0*/
                         exit(1);
                     }
                  close(p1[0]);
                  exit(0);
          }
```

```
else /* padre P0: */
{  pid[1]=fork();
   if (!pid[1])  /* processo P2: */
   {  close(p2[1]);
      close(p1[0]);
      close(p1[1]);
      cont=0;
      while(read(p2[0], &c, 1))
      {  cont++;
         if(cont==M)
         { printf("P2: %d-simo car= %c\n", M, c);
           close(p2[0]); /*sigpipe a P0!!*/
           exit(1);
         }
      }
      close(p2[0]);
      exit(0);
} /* fine P2*/
```

```
 /* padre P0: */
 close(p1[0]);
 close(p2[0]);
 fd=open(argv[1], O_RDONLY);
 OK1=1; OK2=1; /* indicano se le pipe sono aperte*/
 while(k=read(fd, buff, N))
 {       scrittura=1; /* fase di comunicazione con P1:*/
         if (OK1)        write(p1[1], buff, k);
         else printf("P0: la pipe p1 e` chiusa\n");
         scrittura=2; /* fase di comunicazione con P2:*/
         if (OK2)        write(p2[1], buff, k);
         else printf("P0: la pipe p2 e` chiusa... \n");
  }
  close(fd);
  close(p1[1]);
  close(p2[1]);
  exit(0);
}}/* fine main */
```

Compito 17 Giugno 2004 (E)

Si scriva un programma in C che, utilizzando le s*ystem call* di unix, preveda la seguente sintassi:

prova C F0 F2 N

dove:

- **prova** è il nome dell'eseguibile da generare;
- **C** e` un singolo carattere;
- **F0 e F2** sono nomi di file leggibili da tutti gli utenti;
- **N** e` un intero positivo.

Il programma dovrà funzionare nel modo seguente:

- Il processo 'padre' (**P0**) deve creare un processo figlio (**P1**) che a sua volta creera` un processo nipote (**P2**).

```
 void brokenP(int sig) /*gestore chiusura pipe*/
 {  printf("PIPE CHIUSA!!!!\n");
    signal(SIGPIPE, brokenP);
    if (scrittura==1)  /* P1 ha chiuso il lato
                          di lettura della   pipe */
    {  OK1=0;
       close(fd);
       close(p1[1]);
       close(p2[1]);
       kill(pid[1], SIGKILL);
       exit(1);
    }
    if (scrittura==2) /* P2 ha chiuso il lato
                         di lettura della pipe*/
    {       OK2=0;
            close(p2[1]);
    }
 }
```

I processi **padre** e **nipote** svolgeranno compiti **simili**:

- **P0** dovra` leggere il contenuto del file **F0**: ad ogni occorrenza del carattere **C**, dovra` comunicarne la posizione (rispetto all'inizio del file) al processo P1;
- **P2** dovra` leggere il contenuto del file **F2**: ad ogni occorrenza del carattere **C**, dovra` comunicarne la posizione (rispetto all'inizio del file) al processo P1;

- Il processo figlio **P1** dovra` comportarsi come segue:
  - man mano che riceve valori da P0, li somma; se tale somma raggiunge o supera il valore N, P1 dovra` interrompere **istantaneamente** la lettura del file nel processo P0 .
  - man mano che riceve valori da P2, li somma; se tale somma raggiunge o supera il valore N, P1 dovra` interrompere **istantaneamente** la lettura del file nel processo P2.
- Il processo **P0**, in caso di interruzione della lettura provocata da P1, attende la terminazione del figlio P1: in caso di terminazione volontaria, stampa sullo standard output lo stato di terminazione del figlio terminato.
- Il processo **P2**, in caso di interruzione della lettura provocata da P1, stampa il proprio PID sullo standard output e poi termina.

Impostazione

- P0 trasmette informazioni a P1
- P2 trasmette informazioni a P1

  -> apertura di due pipe
- P1 puo` provocare la terminazione di P0 e/o P2: uso del segnale SIGUSR1, gestito diversamente dai due processi -> gestori h0 e h2

```
N=atoi(argv[4]);
C=argv[1][0];
strcpy(F2, argv[3]);
pipe(p01);
signal(SIGUSR1, h0);
pid1=fork();
if (!pid1)
       figlio(N); /* chiamata senza ritorno*/
/* padre P0: */
close(p01[0]);
fd=open(argv[2], O_RDONLY);
while (read(fd, &letto, 1))
{      if (letto==C)
            write(p01[1], &k, sizeof(int));
       k++;
}
close(fd);
close(p01[1]);
wait(&status);
exit(0);
} /* fine main */
```

Soluzione

```
/* SINTASSI: esame  C F0 F2 N*/
#include <signal.h>
#include <fcntl.h>

void h0(int sig);
void h2(int sig);
void figlio(int n);
void nipote(char *F, char C);

int p01[2], p12[2], fd, status;
char C, F2[20];

main(int argc, char ** argv)
{  char letto;
   int N, pid1, k=0;
   if (argc!=5)
   {       printf("sintassi!\n");
           exit(-1);
    }
```

```
void figlio(int n) /* codice del figlio */
{   int pid2, pid0,k, sum0=0, sum2=0, r0=1, r2=1;
    close(p01[1]);
    pipe(p12);
    pid0=getppid();
    pid2=fork();
    if (!pid2)
          nipote(F2, C);/* chiamata senza ritorno*/
    close(p12[1]);
    while (r0 || r2)
    {   if(r0=read(p01[0],&k, sizeof(int)))
        {    sum0+=k;
             if (sum0>=n) kill(pid0, SIGUSR1);
         }
        if (r2=read(p12[0], &k, sizeof(int)))
        {    sum2+=k;
             if (sum2>=n) kill(pid2, SIGUSR1);
        }
    }
    close(p01[0]); close(p12[0]); wait(&status); exit(0);
} /* fine figlio*/
```

```
void nipote(char f[], char c)
{   int k=0;
    char letto;
    close(p01[0]);
    signal(SIGUSR1, h2);
    close(p12[0]);
    fd=open(f, O_RDONLY);
    while (read(fd, &letto, 1))
    {       if(letto==c)
                    write(p12[1], &k, sizeof(int));
            k++;
    }
    close(fd);
    close(p12[1]);
    exit(0);
} /* fine nipote */
```

```
void h0(int sig) /* gestore SIGUSR1 per P0 */
{    printf("P0: ho ricevuto il segnale di interruzione
       lettura...\n"); /* non necessaria */
     close(fd);
     close(p01[1]);
     wait(&status);
     if ((char)status==0)
        printf("Term. volontaria con stato %d\n", status>>8);
     exit(0);
}

void h2(int sig) /* gestore SIGUSR1 per P2 */
{    printf("P2: ho ricevuto il segnale di interruzione
      lettura...\n"); /* non necessaria */
     close(fd);
     close(p12[1]);
     printf("Nipote %d: termino per SIGUSR1!!\n", getpid());
     exit(0);
}
```