Soluzione del Compito del 14 Giugno 2002

Compito A – Esercizio 1

- Si scriva un programma in C che realizzi un comando che, utilizzando le system call di unix, preveda la seguente sintassi:

  `esame file_in car N1 N2`

- dove:
- esame è il nome dell'eseguibile da generare
- file_in è il nome di un file esistente, su cui si hanno i diritti di lettura
- car è un carattere
- N1 e N2 sono interi positivi

Il comando dovrà funzionare nel modo seguente:

- il processo 'padre' (P0) deve creare un processo figlio, P1.
- il processo 'figlio' (P1) deve creare un processo nipote, P2;
- il padre P0 deve leggere il contenuto di **`file_in`**: ogni volta che incontra il carattere **`car`** all'interno del file, ne deve comunicare al figlio e al nipote la posizione all'interno del file (numero intero positivo);
- il figlio P1, per ogni valore V ricevuto da P0, confronta il valore di V con N1; al raggiungimento (ed eventuale superamento) del valore N1, P1 deve avvisare il padre, provocare la terminazione del padre e di P2 e infine terminare.
- il nipote P2, per ogni valore V ricevuto da P0, confronta il valore di V con N2; al raggiungimento (ed eventuale superamento) del valore N2, P1 deve avvisare P0, provocare la terminazione di P0 e di P1 e infine terminare.

Impostazione

- Comunicazione dei processi figlio e nipote con P0: uso di due pipe:

  `pipe1`
  `pipe2`

- uso dei segnali.

Soluzione :

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

void handler1(int sig); /*gestore segnali figlio */
void handler2(int sig); /*gestore segnali nipote */
int ppid, fpid, npid;

main(int argc, char **argv)
{     int pos, pipe1[2], pipe2[2];
      char n, car;
      int N1, N2, val, fd;
  if (argc!=5)
        {   printf("sintassi!\n");
              exit(-1);
        }
ppid=getpid();
N1=atoi(argv[3]); N2=atoi(argv[4]);
if (pipe(pipe1)<0) exit(-2);
if (pipe(pipe2)<0) exit(-2);
```



```
if ((fpid=fork())<0)/*creaz. figlio*/
{  perror("fork"); exit(-3);}
else if (fpid==0) /* figlio*/
{   if((npid=fork())<0) /* creaz. nipote*/
    {  perror("fork"); exit(-3);}
    else if (npid==0) /* nipote*/
    {       close(pipe1[0]);
            close(pipe1[1]);
            close(pipe2[1]);
            sleep(1);
            while(n=read(pipe2[0], &pos,sizeof(int))>0)
            { printf("NIPOTE: ricevuto %d\n", pos);
              if(pos>=N2)
              {       kill(ppid,SIGUSR2);
                      close(pipe2[0]);
                      exit(0);
              }
             }/* fine while*/
             close(pipe2[0]); exit(0);
    } /* fine nipote */
```



```
else /*figlio*/
{       close(pipe2[0]);
        close(pipe2[1]);
        close(pipe1[1]);
        sleep(1);
        while(n=read(pipe1[0], &pos,sizeof(int))>0)
        {       printf("FIGLIO: ricevuto %d\n", pos);
                if(pos>=N1)
                {       kill(ppid,SIGUSR1);
                        kill(npid,SIGKILL);
                        close(pipe1[0]);
                        exit(0);
                }
         }
         close(pipe1[0]); exit(0);
} }/* fine figlio */
```



```
/* padre: */
signal(SIGUSR1, handler1);
signal(SIGUSR2, handler2);
close(pipe1[0]);
close(pipe2[0]);
if ((fd=open(argv[1],O_RDONLY))<0)
{perror("apertura file"; exit(-5);}
pos=0;
while((n=read(fd,&car, 1))>0)
{ if(car==argv[2][0])
  {   write(pipe1[1], &pos, sizeof(int));
      write(pipe2[1], &pos, sizeof(int));
  }
  pos++;
}
close(fd); close(pipe1[1]); close(pipe2[1]);
} /* fine main*/
```

```
void handler1(int sig) /*gest.segnali figlio */
{ printf("segnale %d dal figlio %d!\n", sig, fpid);
  exit(0);
}


void handler2(int sig) /*gestore segnali nipote */
{ printf("segnale %d dal nipote %d!\n", sig, npid);
  kill(fpid, SIGKILL);
  exit(0);
}
```