

Esame di Calcolatori Elettronici II del 6/5/96 Parte 2

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

esame F1 F2 ..Fn Fris T C

dove:

- **F1, F2, .. Fn, Fris** rappresentano nomi assoluti di file.
- **T** rappresenta un intervallo di tempo (in sec.)
- **C** rappresenta un singolo carattere.

In particolare, il processo iniziale **P0** deve generare **n** figli (**P1,P2,.. Pn**).

Ogni processo figlio **Pi** deve contare le occorrenze del carattere **C** nel file **Fi** assegnatogli.

Il processo iniziale **P0**, trascorso un intervallo di tempo pari a **T** secondi, provoca la terminazione forzata di tutti i figli.

Ciascun figlio, prima di terminare, deve scrivere nel file **Fris** il risultato (parziale o definitivo) del conteggio eseguito.

A questo proposito, non ha interesse l'ordine con il quale i figli scrivono i risultati su **Fris**.

Si realizzi il programma avendo come obiettivo un alto grado di parallelismo.

Soluzione "Standard" (System V): contiene alcuni errori ed alcune inefficienze

```
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#define MAX_FIGLI 20

int cont=0, fd_ris; ⑥

void handler(segnale)
int segnale;
{ char s[10];
  sprintf(s,"%d",cont);
  write(fd_ris, s, 10);
  ①
  printf("termine processo %d",getpid());
  exit(0);
}

main(argc,argv)
int argc;
char **argv;
{
  int i, T, n, Nfigli, pid[MAX_FIGLI], fd_in,
  stato;
  char buf[1];

  if(argc<5){printf("Usage error\n");exit(1);}

  T=atoi(argv[argc-2]);
  Nfigli=argc-4;

  if ((fd_ris=creat(argv[argc-3],0777))<0) {
    perror("creat fallita\n");
    exit(1);
  }
```

```

sigset(SIGUSR1,handler); ❷

for(i=0;i<Nfigli;i++) {
    pid[i]=fork();
    if (pid[i]<0) {
        perror("Fork Fallita\n"); exit(1);
    }

    if (pid[i]==0){ /* Figlio*/
        fd_in=open(argv[i+1], O_RDONLY);
        while((n=read(fd_in,buf,1)==1)) ❸
            if(buf[0]==argv[argc-1][0])
                cont++;

        ❹
        handler();
    }
}

sleep(T); ❹

for(i=0;i<Nfigli;i++) kill(pid[i],SIGUSR1); ❺
for(i=0;i<Nfigli;i++) wait(&stato);

}

```

Possibili miglioramenti

- ❶ Errore: possibile corsa critica nell'handler, inserire una **signal**(SIGUSR1,SIG_IGN) in ❷
- ❸ Inefficienza: read di un carattere alla volta, sostituibile con qualcosa tipo


```

while((n=read(fd,buf,BUFSIZ))>0)
    ..... buf → str
    while((str=strchr(str,char))!=NULL)
        cont++;
      
```
- ❹ Inefficienza: il processo padre attende sempre T secondi, anche se i figli terminano prima, si potrebbe usare un alarm, tipo


```

alarm(T);
for(i=0;i<Nfigli;i++) wait(&stato);
      
```

 il gestore del SIGALRM deve mandare il SIGKILL ai processi non ancora terminati
- ❺ Si può mandare il SIGUSR1 al gruppo, utilizzando una sola kill(0,SIGUSR1), avendo cura di far ignorare il segnale al padre.
- ❻ Non dimenticarsi che queste variabili devono essere globali
- ❼ Attenzione alle differenze System V e BSD. Cosa sarebbe successo se invece di sigset() si fosse usato signal()?

VERSIONE CON PIPE
(per comunicare risultati tra figli e padre)

```
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#define MAX_FIGLI 20

int cont=0, piped[2];

void handler(segnale)
int segnale;
{ char s[10];
  sprintf(s,"%d\n",cont);
  write(piped[1], s, 10);
  exit(0);
}

main(argc,argv)
int argc;
char **argv;
{
  int i, T, n, Nfigli, pid, fd_in, stato;
  char buf[10];

  if(argc<5){printf("Usage error\n");exit(1);}

  T=atoi(argv[argc-2]);
  Nfigli=argc-4;

  pipe(piped);
```

```
  signal(SIGUSR1,handler);

  for(i=0;i<Nfigli;i++) {
    pid=fork();
    if (pid==-1) {
      perror("Fork Fallita\n"); exit(1);
    }

    if (pid==0){ /* Figlio*/
      close(piped[0]);
      fd_in=open(argv[i+1], O_RDONLY);
      while((n=read(fd_in,buf,1))!=1)
        if(buf[0]==argv[argc-1][0])
          cont++;
      signal(SIGUSR1,SIG_IGN);
      handler();
    }
  }

  close(piped[1]);
  signal(SIGUSR1,SIG_IGN);
  sleep(T);
  kill(0,SIGUSR1);
  for(i=0;i<Nfigli;i++) {
    read(piped[0],buf,10);
    printf("padre: %s\n",buf);
  }
  for(i=0;i<Nfigli;i++) wait(&stato);
  exit(0);
}
```