

Gestione di Processi e Accesso a File

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

ESEMPIO Dati Car Frazione Risultati

Il programma ESEMPIO, quindi, accetta 4 argomenti:

- Dati, che rappresenta il nome assoluto di un file di testo;
- Car, che rappresenta un carattere;
- Frazione, che rappresenta un intero;
- Risultati, che rappresenta il nome assoluto di un file.

Il programma ESEMPIO richiede la lettura del file specificato da parte di una gerarchia di processi *lettori*. In particolare, ciascun processo della gerarchia è incaricato di scandire solo una frazione di file della dimensione (espressa in numero di bytes) data dal parametro Frazione. Pertanto, la dimensione della gerarchia e' strettamente dipendente dalla dimensione del file dato.

Ogni processo *lettore* deve individuare nel file tutti i caratteri uguali al parametro Car; ad ogni occorrenza, ciascuno di essi ne memorizza la posizione nel file Risultati.

La soluzione deve essere mirata a fornire il massimo grado di parallelismo.

Soluzione:

Le specifiche del problema sono tali da suggerire una soluzione ricorsiva: ognuno dei processi lettori esegue una funzione ricorsiva (*cerca*) che, ad ogni invocazione, nel caso più generale crea un processo figlio che esegue la stessa funzione ricorsiva.

La struttura del programma principale risulta pertanto molto semplice:

```
main(argc, argv)
int argc;
char **argv;
{
    int pid, frazione, stato;

    frazione=atoi(argv[3]); /*dimensione frazione */

    pid=fork();
    if (pid==0) { /*codice primo lettore*/
        cerca(argv[1], 0, argv[2][0], frazione);
        exit(0);
    } else { /* codice padre*/
        wait(&stato);
        exit(0);
    }
}
```

Il massimo grado di parallelismo si ottiene facendo in modo che ciascun lettore acceda al file dato *indipendentemente* dagli altri, e cioè' con I/O pointer distinti.

Per questo ogni processo lettore apre individualmente il file *Dati*; successivamente esso stabilisce, in base alla dimensione del file e alla frazione assegnatagli, se deve creare o meno un ulteriore lettore (soluzione ricorsiva), e poi, dopo l'eventuale creazione, procede alla scansione della frazione.

Sia per la valutazione della dimensione del file, che per il posizionamento del puntatore, viene usata la primitiva lseek.

La struttura della funzione *cerca* e' quindi la seguente:

```
void cerca(nomefile, livello, carattere, frazione)
char *nomefile;
int livello, frazione;
char carattere;
{
    int pos, dim, figlio,i, fd, n, stato;
    char r;
    figlio=0;
    fd=open(nomefile, O_RDONLY);
    dim=lseek(fd,0,2); /*ricava la dimensione del file */
    pos=livello*frazione;
    if (pos+frazione<dim) { /* e' necessario creare un figlio*/
        if (fork()==0) {
            close(fd);
            cerca(nomefile, ++livello, carattere, frazione);
            exit(0);
        }
    }
    lseek(fd,pos,0); /* posiziona l'I/O pointer */
    n=1;
    for (i=0; (i<frazione)&&(n==1); i++) { /*ciclo di lettura */
        n=read(fd,&r,1);
        if (r==carattere)
            printf("proc. %d, posizione %d\n",getpid(),pos+i);
    }
    close(fd);
    wait(&stato); /* se non ha creato figlio, wait non blocca*/
}
```

```

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

main(argc, argv)
int argc;
char **argv;
{
    int pid, frazione, fileout, stato;

    frazione=atoi(argv[3]); /*dimensione frazione */
    close(1);
    fileout=creat(argv[4],0777);/*redirezione std. output*/
    pid=fork();
    if (pid==0) { /*codice primo lettore*/
cerca(argv[1], 0, argv[2][0], frazione);
        exit(0);
    } else { /* codice genitore*/
        wait(&stato);
        exit(0);
    }
}

```

Si noti che, per semplificare la scrittura sul file *Risultati*, si è utilizzata la ridirezione dello standard output.

Si realizzi una soluzione concorrente non ricorsiva ma iterativa.