Lezione 18

- Scrivere un programma che riporta le modifiche di dimensione di un file (specificato come primo argomento sulla riga di comando) nell'arco di un intervallo di tempo (specificato in secondi come secondo argomento sulla linea di comando). Alla fine il programma deve produrre sullo schermo del terminale un istogramma delle modifiche (si utilizzi ad esempio il carattere *).

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

main(int argc,char **argv) {
  int sec,i,mod=0;
  struct stat buf;
  time_t last_mtime;

  if(argc!=3) {
    fprintf(stderr,"Uso: mod <file> <secondi>\n");
    exit(1);
  }

  if(stat(argv[1],&buf)==-1) {
    perror("Errore nella chiamata a stat");
    exit(2);
  }

  last_mtime=buf.st_mtime;
  sec=atoi(argv[2]);

  for(i=0;i<sec;i++) {

    if(stat(argv[1],&buf)==-1) {
      perror("Errore nella chiamata a stat");
      exit(2);
    }

    if(buf.st_mtime!=last_mtime) {
      mod++;
      last_mtime=buf.st_mtime;
    }

    sleep(1);
  }

  if(mod==0)
    printf("\nNegli ultimi %s secondi il file '%s' "
           "non e' stato modificato.\n",argv[2],argv[1]);
  else
```

```
    printf("\nModifiche effettuate al file '%s' "
           "negli ultimi %s secondi:\n",argv[1],argv[2]);

  for(i=0;i<mod;i++)
    putchar('*');
    putchar('\n');

  exit(0);
}
```

- Scrivere un programma che copia il primo file passato come argomento sulla linea di comando nel file passato come secondo argomento, utilizzando le chiamate di sistema `open`, `read` e `write`. Si confrontino le prestazioni di questo programma con quelle di `copyfile.c` che sfrutta il memory mapped I/O.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFSIZE 512

main(int argc,char **argv) {
  int fd1,fd2;
  ssize_t n;
  char buf[BUFSIZE];

  if(argc!=3) {
    fprintf(stderr,"Uso: copy <file1> <file2>\n");
    exit(1);
  }

  if((fd1=open(argv[1],O_RDONLY))==-1) {
    perror("Impossibile aprire in lettura il primo file");
    exit(2);
  }

  if((fd2=open(argv[2],O_WRONLY | O_CREAT | O_TRUNC,0644))==-1) {
    perror("Impossibile aprire in scrittura il secondo file");
    close(fd1);
    exit(3);
  }

  while((n=read(fd1,buf,BUFSIZE))>0) {

    if(write(fd2,buf,n)<n) {
      perror("Errore di scrittura");
      close(fd1);
      close(fd2);
      exit(4);
```

```
    }

  }

  close(fd1);
  close(fd2);

  if(n==-1) {
    perror("Errore di lettura");
    exit(5);
  }

  exit(0);
}
```

Per confrontare in modo oggettivo le prestazioni del programma precedente e quello che fa uso del memory mapped I/O illustrato a lezione si può far uso del comando `time`. In generale, le prestazioni del programma precedentemente riportato dipendono pesantemente dal valore `BUFSIZE`. Infatti, impostandolo a 1 si ottengono le prestazioni peggiori in assoluto. Man mano che il valore aumenta le prestazioni migliorano. Queste ultime diventano ottimali quando `BUFSIZE` rappresenta un valore che sia un multiplo della dimensione di un blocco fisico del disco (tipicamente 512 byte). La ragione principale per cui il programma visto a lezione offre prestazioni migliori sta nel fatto che usa un numero considerevolmente minore di chiamate a system call. Infatti l'overhead dovuto allo switching fra user mode e kernel mode (durante l'esecuzione di una system call) è alquanto pesante e si fa sentire man mano che aumenta il numero di chiamate.