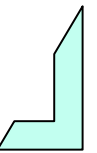




Esercitazione di Lab. di Sistemi Operativi

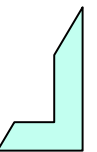
- I/O di basso livello -





Sommario

- @ System Call
- @ Fasi della compilazione
- @ Il compilatore GNU gcc
- @ Esercizi



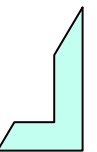
- System call -



Richiami sulle System Call

@ Unix permette ai processi di accedere a file, mediante un insieme di **System Call**:

- apertura/creazione:
 - **open, creat**
- chiusura:
 - **close**
- lettura:
 - **read**
- scrittura:
 - **write**
- modifica offset
 - **lseek**



- System call: open -

System call - open - Esempio d'uso

```
#include <sys/types.h>
#include <sys/stat.h>    aprire e/o creare file
#include <fcntl.h>
int open(char nomefile[], int flag, [int mode]);
```

- @ **nomefile**: è il nome del file (relativo o assoluto) da aprire e/o da creare
- @ **flag**: esprime il modo di accesso definito in <fcntl.h>:
 - @ O_RDONLY = 0, per accesso in lettura,
 - @ O_WRONLY = 1, per accesso in scrittura
 - @ A_APPEND = 2, per accesso in scrittura in append
 - @ è possibile abbinare ai tre modi precedenti, altri modi (mediante il connettore |); ad esempio: **O_CREAT**, per accesso in scrittura, se il file non esiste, viene creato.
- @ [**mode**] è un parametro richiesto soltanto se l'opzione di apertura determina la creazione del file (**O_CREAT**) e ne determina la protezione (**S_IRUSR** | **S_IWUSR**) def. in <fcntl.h> per il proprietario

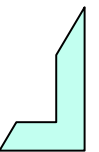


System call - open - Esempio d'uso

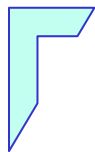
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(char nomefile[], int flag, [int mode]);

/* crea un file in sola scrittura */
fd = open("prova.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR );
```

- @ Il valore restituito dalla **open** è:
 - @ fd (**file descriptor**) associato al file
 - @ **-1** in caso di errore.
- @ Se la **open** ha successo, il file viene aperto nel modo richiesto, e l' I/O pointer è posizionato sul primo elemento (tranne nel caso di O_APPEND)



- System call: creat -



System call - creat - Esempio d'uso

- @ Un nuovo file può essere creato anche con **creat**

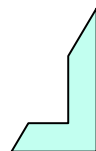
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(char nomefile[], int mode);
```

- **nomefile** è il nome del file (relativo o assoluto) da creare
- **mode** specifica i 12 bit di protezione per il nuovo file
(**S_IRUSR** | **S_IWUSR**) def. in <fcntl.h>

- @ Il valore restituito dalla **creat** è:

- @ fd (**file descriptor**) associato al file
- @ **-1** in caso di error

- @ Se la **creat** ha successo, il file viene aperto in scrittura, e l'I/O pointer posizionato sul primo elemento.



- System call: close -



System call - close - Esempio d'uso

- @ Per chiudere un file aperto si usa la funzione **close**:

```
#include <unistd.h>
int close(int fd);
```

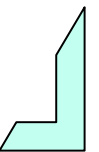
➤ **fd** è il **file descriptor** del file da chiudere

- @ Restituisce l' esito della operazione:

- @ 0 in caso di successo

- @ -1 in caso di errore

- @ Se la **close** ha successo si ha la memorizzazione del file su disco



- System call: write -



System call - write - Esempio d'uso

@ Per scrivere in un file si usa la funzione:

```
#include <unistd.h>
int write(int fd, char *buf, int n);
```

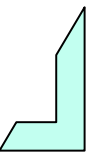
- **fd** è il file descriptor del file in cui scrivere
- **buf** è la struttura dati che contiene i caratteri da scrivere (array di char)
- **n** è il numero di caratteri da scrivere

@ Restituisce :

@ **nwrite** (numero di byte scritti) in caso di successo

@ **-1** in caso di errore

@ è previsto un carattere di **End-Of-File** che marca la fine del file



- System call: read -



System call - read - Esempio d'uso

- Ⓢ Per leggere in un file si usa la funzione:

```
#include <unistd.h>
int read(int fd, char *buf, int n);
```

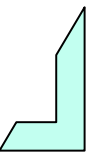
- **fd** è il file descriptor del file da leggere
- **buf** è la struttura dati in cui mettere i caratteri letti (array di char)
- **n** è il numero di caratteri da leggere

- Ⓢ Restituisce :

- Ⓢ **nread** (numero di byte letti) in caso di successo

- Ⓢ **-1** in caso di errore

- Ⓢ è previsto un carattere di **End-Of-File** che marca la fine del file

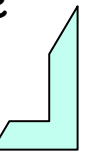


- Fasi della compilazione -



- Fasi della compilazione -

- @ Il compilatore GNU C in ambiente Unix, per generare il file eseguibile dal file sorgente, passa attraverso le seguenti fasi:
 - @ Fase 1) **Preprocessore**
 - @ Il preprocessore legge un sorgente C e produce in output un altro sorgente C, dopo avere **espanso in linea le macro**, e **incluso i file**. (#define MAX 100) (#include <nomefile>)
 - @ Fase 2) **Compilazione**
 - @ Traduzione del codice sorgente ricevuto dal preprocessore in codice assembly (codice molto vicino al linguaggio macchina)
 - @ Fase 3) **Assembler** (creazione del file oggetto)
 - @ Crea il codice oggetto (file con il suffisso .o)
 - @ Fase 4) **Linker** (creazione del file eseguibile)
 - @ Unisce le funzioni definite in altri file sorgenti o definite in librerie, con la funzione main() per creare il file eseguibile.



- Il compilatore GNU gcc in ambiente Unix -

- Compilazione -

- ② Per compilare in ambiente Unix, un programma C scritto nel file **source.c** si utilizza un compilatore GNU per sorgenti C, che viene invocato con il comando **gcc**:

Compilazione:

\$ **gcc** source.c ➡ **a.out** (eseguibile prodotto dal compilatore)

Opzioni comando gcc:

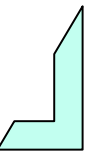
- flag **"-o"**: \$ gcc **-o** eseguibile.out sorgente.c
- flag **"-v"**: Per sapere la versione del compilatore
- flag **"-E"**: Serve per invocare solo il preprocessore
- flag **"-c"**: Per creare il codice oggetto senza chiamare il linker
- flag **"-Wall"**: per aumentare il livello dei messaggi prodotti in fase di compilazione
- \$ **man** gcc (manuale in linea)

- Il messaggi del compilatore -



- I messaggi del compilatore -

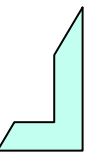
- @ Il compilatore invia dei messaggi all' operatore:
 - @ **Messaggi di avvertimento** (warning messages)
 - @ I messaggi di avvertimento indicano la presenza di parti di codice presumibilmente mal scritte o di problemi che potrebbero avvenire in seguito, durante l'esecuzione del programma. **I messaggi di avvertimento non interrompono comunque la compilazione.**
 - @ **Messaggi d'errore** (error messages)
 - @ I messaggi di errore invece indicano qualcosa che deve essere necessariamente corretto e causano l'interruzione della compilazione.
- @ Per **inibire** tutti i messaggi di **warning** usare l'opzione -w
 - @ \$ gcc -w -o eseguibile.out source.c
- @ Per avere il **massimo livello** di **warning** usare l'opzione -Wall
 - @ \$ gcc -Wall -o eseguibile.out source.c





- Compilare con opzione debug -

- Ⓢ Per effettuare il debug di un programma, utilizzate sempre l'opzione `-g`:
- Ⓢ `$ gcc -Wall -g -o eseguibile.out source.c`
- Ⓢ L'opzione `-g` fa sì che il programma eseguibile contenga informazioni supplementari che permettono al debugger di collegare le istruzioni in linguaggio macchina che si trovano nell'eseguibile alle righe del codice corrispondenti nei sorgenti C





Esercizio n° 1 (write)

- @ Scrivere un programma C "**scrivi.c**" che utilizzando la funzione primitiva "**write**", scriva in un file "**alfabeto.txt**" la seguente stringa di caratteri: "**ABCDEFGHILMNOPQRSTUVWXYZ**"

Compilazione:

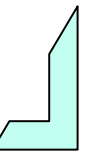
```
$ gcc -o scrivi.out scrivi.c
```

oppure

```
$ cc -o scrivi.out scrivi.c
```

Esempio di esecuzione:

```
$ ./scrivi.out
```



Soluzione Esercizio n° 1 (write)

```
#include <stdlib.h> /*exit*/
#include <errno.h> /*perror*/
#include <stdio.h>
#include <unistd.h> /*write, close*/
#include <sys/types.h> /*open*/
#include <sys/stat.h> /*open*/
#include <fcntl.h> /*open*/
#include <string.h> /*strlen*/
int main(void)
{   int fd, nwrite;
    char data[] = "ABCDEFGHILMNOPQRSTUVWXYZ"; /*array di char*/
    /* crea un file in sola scrittura */
    fd = open( "alfabeto.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR );
    if( fd == - 1 )
    {perror("errore apertura" ); return 1; }
    /* memorizza la stringa sul file */
    nwrite = write( fd, data, strlen( data ) );
    if( nwrite == -1 )
    { perror( "errore scrittura" ) ; return 2; }
    close( fd );
    exit( 0 );
}
```

Calcola la lunghezza della stringa da scrivere nel file



Esercizio n° 2 (read)

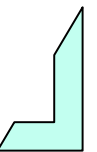
- ④ Scrivere un programma C "**leggi.c**" che utilizzando la funzione primitiva "**read**", legga il contenuto del file "**alfabeto.txt**" e lo stampi sullo standard output.

Esempio di esecuzione:

```
$ ./leggi.out
```

Output:

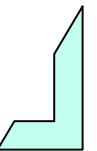
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"





Soluzione Esercizio n° 2 (read)

```
#include <stdlib.h> /* exit */
#include <errno.h> /* perror */
#include <stdio.h>
#include <unistd.h> /* write, read, close*/
#include <sys/types.h> /*open*/
#include <sys/stat.h> /*open */
#include <fcntl.h> /*open*/
#define BUFDIM 1000
int main(void)
{
    int fd, nread;
    char buffer[ BUFDIM ]; /*array di 1000 char*/
    fd = open( "alfabeto.txt", O_RDONLY);
    if( fd == - 1 )
    { perror( "errore apertura" );
      return 1 ; }
}
```



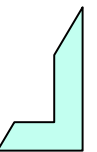


Soluzione Esercizio n° 2 (read)

```
/*cicla finchè c'è da leggere*/
while( (nread=read(fd, buffer, BUFDIM) ) > 0 )
{
    if( write(STDOUT_FILENO, buffer, nread) == -1 )
    {
        close( fd );
        return 2;
    }
} /*fine ciclo while*/

close( fd );
return 0;

} /*Fine programma */
```

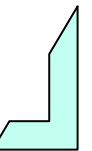


- offset -



- L' offset -

- Ⓢ Ad ogni file aperto è associato un intero, detto **offset**, che rappresenta la posizione (espressa in numero di byte dall'inizio del file) in cui verrà effettuata la prossima operazione di I/O (write, read)
- Ⓢ L' offset è inizializzato a zero da **open**
 - a meno che non sia specificato **O_APPEND**
- Ⓢ Le operazioni di **read** e **write** incrementano il valore dell' offset di un numero di byte pari al numero di byte letti/scritti



- System call: lseek -

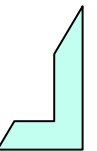


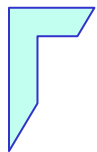
System call - lseek - Esempio d'uso

- @ Per modificare l' offset di un file:

```
#include <sys/types.h>
#include <unistd.h>
int lseek (int fd, int offset, int whence);
```

- @ **fd** è il file descriptor del file di cui si vuole mod. l'offset
- @ **offset** dipende dal valore del parametro **whence**
- @ **whence**:
 - @ **SEEK_SET**: setta l'offset al valore **offset** specificato
 - @ **SEEK_CUR**: fornisce l'offset corrente, partendo da **offset**
 - @ **SEEK_END**: fornisce l'offset di fine file, partendo da **offset**
- @ Restituisce:
 - @ il **nuovo** valore dell' offset
 - @ o **-1** in caso di errore





System call - lseek - Esempio d'uso

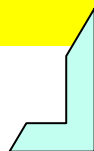
ESEMPIO

```
#include <sys/types.h>
#include <unistd.h>

/*Setta il valore dell'offset al valore specificato in
offset*/
lseek (fd, offset, SEEK_SET);

/*Restituisce l'offset corrente*/
lseek (fd, 0, SEEK_CUR);

/*Si posiziona alla fine del file*/
lseek (fd, 0, SEEK_END);
```





System call - lseek - Esempio d'uso

ESEMPIO

```
#include <sys/types.h>
#include <unistd.h>

int main(void) {

    if (lseek(fd, 0, SEEK_CUR) == -1) {
        printf("seek KO\n");
        exit( 1 );
    }
    else {
        printf("seek OK\n");
        exit( 0 );
    }
}
```





Esercizio n° 3 (lseek)

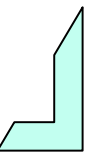
- Ⓢ Usando la primitiva **lseek**, modificare il programma C **"leggi.c"** in modo da leggere e stampare su standard output solo la stringa **"ABFGMNRSZ"** contenuta nel file **"alfabeto.txt"**

Input:

"ABCDEFGHIJKLMNOPQRSTUVWXYZ" caratteri

Output:

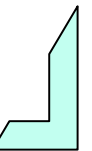
"ABFGMNRSZ"





Soluzione Esercizio n° 3 (lseek)

```
#include <stdlib.h> /* exit */
#include <errno.h> /* perror */
#include <unistd.h> /* write, read, lseek, close*/
#include <sys/types.h> /*open, lseek */
#include <sys/stat.h> /*open */
#include <fcntl.h> /* open*/
#define BUFDIM 1000
int main(void)
{
    int fd, nread;
    int i=0;
    char buffer[ BUFDIM ];
    /* apre il file in sola scrittura */
    fd = open( "alfabeto.txt", O_RDONLY);
    if( fd == - 1 )
    { perror( "errore apertura" );
      return 1 ); }
}
```

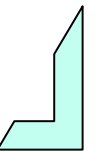




Soluzione Esercizio n° 3 (lseek)

```
/*legge due caratteri alla volta*/
while( ( nread=read(fd, buffer, 2) ) > 0 )
{
    if( write(STDOUT_FILENO, buffer, nread) == -1 )
    {
        close( fd );
        return 2 ;
    }
    i=i+5; /*valore del nuovo offset*/
    if (lseek(fd, i, SEEK_SET) == -1)
    {
        perror( "lseek errore" );
        close( fd );
        return 3;
    }

} /*fine ciclo while*/
close( fd );
return 0;
} /*Fine programma*/
```





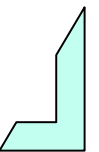
Esercizio n° 4

Scrivere un programma che accetta 4 argomenti:

extract <file1> <file2> <n> <m>

e scrive su file2 la parte di file1 che va dal byte n al byte m.

Il programma deve controllare che gli argomenti siano validi:
file1 deve essere un file esistente file2 non deve esistere n ed m devono essere interi non negativi, con $n \leq m$





Esercizio n° 5

- Si realizzi un programma C "**alternaByte.c**" che mostri il contenuto di un file a byte alterni (un carattere sì ed uno no). Si supponga che tale file venga passato come parametro. funzione main (int argc, char **argv)

Esempio di esecuzione:

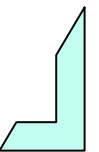
```
$ ./alternaByte.out <nome_file>
```

Il file alfabeto.txt:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Output:

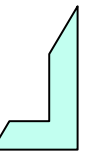
ACEGIMOQSUZ





Soluzione Esercizio n° 5

```
#include <stdlib.h> /* exec */
#include <errno.h> /* perror */
#include <unistd.h> /* write, read, lseek, close*/
#include <sys/types.h> /* open, lseek */
#include <sys/stat.h> /* open */
#include <fcntl.h> /*open*/
#define BUFDIM 1000
main (int argc, char **argv)
{
    int fd, nread;
    int i=0;
    char buffer[BUFDIM];
    if (argc != 2)
    {perror ("errore \n");
        exit (1);}
    if ((fd=open(argv[1], O_RDONLY)) <0)
    {perror("errore apertura file: ");
        return 1;}
}
```

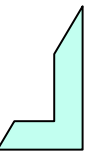




Soluzione Esercizio n° 5

```
while((nread=read(fd, buffer, 1)) >0 )
{
    if( write(STDOUT_FILENO, buffer, nread) == -1 )
    {
        close(fd);
        return 2;
    }
    i=i+2; /*valore per il nuovo offset*/
    if (lseek(fd, i, SEEK_SET) == -1)
    {
        perror( "lseek errore" );
        close(fd);
        return 3;
    }
}

/*fine ciclo while*/
close(fd);
return 0;
} /*fine programma*/
```





Esercizio n° 6

- ② Si realizzi un programma "**invertiByte.c**" che stampi sullo standard output il contenuto di un file (passato come argomento) alla rovescia cioè a partire dall'ultimo carattere fino ad arrivare al primo

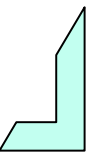
Sintassi di lancio del programma:

```
$ ./invertiByte.out <nome_file>
```

Come esempio si può utilizzare il file alfabeto.txt

Output:

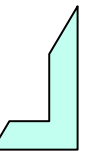
ZVUTSRQPONMLIHGFEDCBA





Soluzione Esercizio n° 6

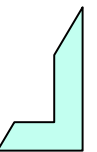
```
#include <stdlib.h> /* exec */
#include <errno.h> /*perror*/
#include <stdio.h>
#include <unistd.h> /*write, read, lseek, close*/
#include <sys/types.h> /*open, lseek */
#include <sys/stat.h> /*open */
#include <fcntl.h> /*open*/
#define BUFDIM 1000
main (int argc, char **argv)
{ int fd, nread;
  int i=1;
  int currpos;
  char buffer[BUFDIM];
  if (argc != 2)
  {perror("errore \n");
    exit (1); }
  if ( (fd=open(argv[1], O_RDONLY)) <0 )
  {perror("apertura sorgente: ");
    return 1;}
```





Soluzione Esercizio n° 6

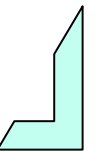
```
/*Si posiziona dalla fine del file*/
if (lseek(fd, 0, SEEK_END) == -1)
{
    perror( "lseek errore" );
    close(fd);
    return 2;
}
/*Restituisce l' offset corrente*/
currpos = lseek(fd, 0, SEEK_CUR);
currpos = currpos-i;
if (lseek(fd, currpos, SEEK_SET) == -1)
{
    perror( "lseek errore" );
    close(fd);
    return 3;
}
```





Soluzione Esercizio n° 6

```
while((nread=read(fd, buffer, 1)) >0 && currpos >= 0 )
{
    if( write(STDOUT_FILENO, buffer, nread) == -1)
    {
        close(fd);
        return 4;
    }
    currpos--; /* decrementa il valore dell'offset */
    if ( currpos >=0 ) {
        if (lseek(fd, currpos, SEEK_SET) == -1)
        {
            perror( "lseek errore" );
            close(fd);
            return 5;}
    }
    else { /*Se fine file*/
        close(fd);
        exit(0);}
}/*fine ciclo while*/
close(fd);
return 0;
} /*fine programma*/
```





Esercizio n° 7

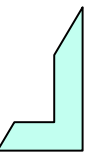
- Ⓜ Si realizzi un programma "**copia.c**" che effettui la copia tra due file passati come parametro

Sintassi di lancio del programma:

```
$ ./copia.out <file_dest> <file_sorg>
```

dove **file_dest** e **file_sorg** sono i file passati come parametro.

Il programma dovrà effettuare la copia del contenuto di **file_sorg** in **file_dest**



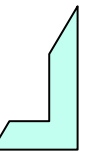


Soluzione Esercizio n° 7

```
#include <stdlib.h> /* exit */
#include <fcntl.h>
#include <stdio.h>
#define BUFDIM 1000
#define perm 0777
main (int argc, char **argv)
{ int infile, outfile, nread;
  char buffer[BUFDIM];
  if (argc != 3)
    { printf ("errore \n"); exit (1); }

  if ((infile=open(argv[2], O_RDONLY)) <0)
    { printf("apertura sorgente:"); exit(1); }

  if ((outfile=creat(argv[1], perm )) <0)
    { printf("apertura destinazione:");
      close (infile); return 1; }
```



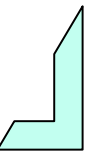


Soluzione Esercizio n° 7

```
while((nread=read(infile, buffer, BUFDIM)) >0 )
{
    if( write(outfile, buffer, nread) == -1)
    {
        close(infile);
        close(outfile);
        exit(1);
    }
    /*fine ciclo while*/

    close(infile);
    close(outfile);
    return 0;

} /*fine programma*/
```





- Fine Esercitazione -

