Laboratorio II

Corso A

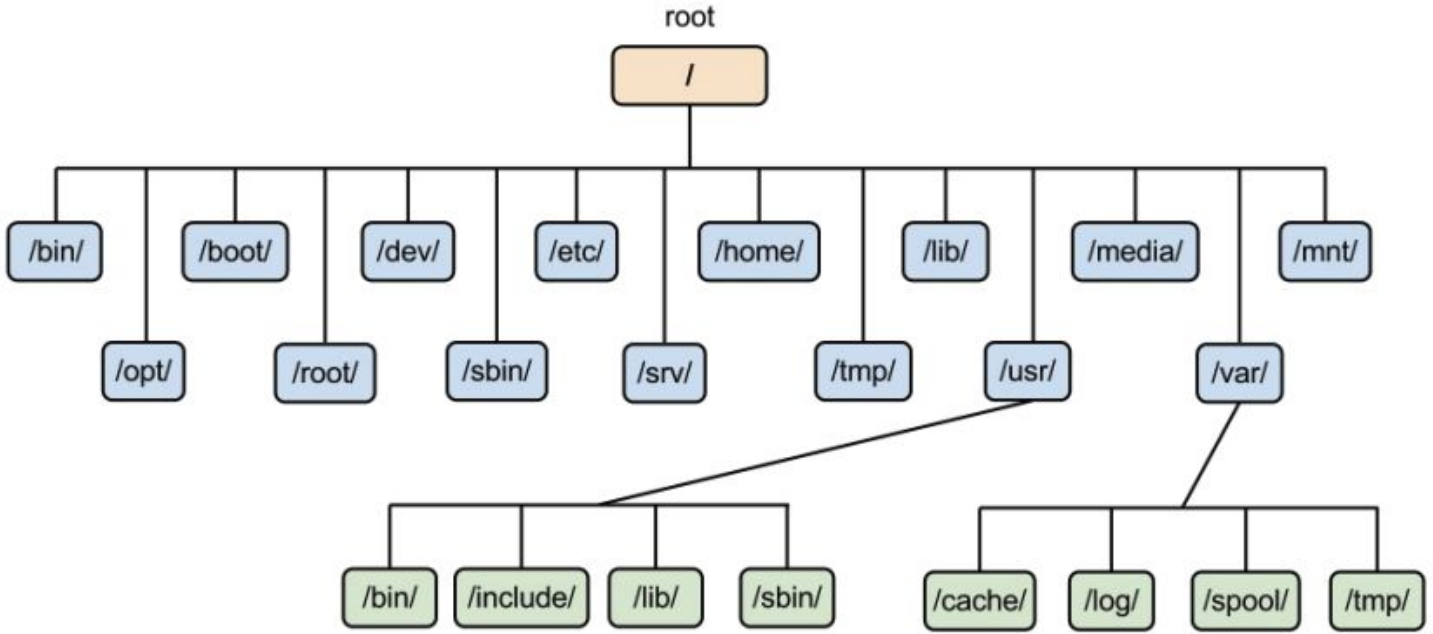
Lezione 10

Gerarchia di directory e file Unix

File I/O in C

Directory e file UNIX

I file UNIX sono strutturati in una gerarchia standard ad albero (FHS, https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard)



cartella	Cosa contiene
/	Cartella radice dell'albero di directory del FS
/home	Contiene le home directory degli utenti (es. */home/riccardo /home/marta /home/sara)*
/etc	Contiene i file (e directory) di configurazione del sistema (es. /etc/passwd, /etc/group, /etc/services, /etc/X11/, ….)
/usr	Contiene eseguibi e librerie associate a software installati dall'amministratore (tipicamente in /usr/local) o dalla distribuzione software (cioè Ubuntu, CentOS, Debian, RedHat, …)
/var	Contiene file variabili come i log di sistema, spool della posta e delle stampanti, cache, etc ...
/dev	Contiene i file associati alle periferiche (devices) e pseudo-devices (file speciali)
/bin	Contiene gli eseguibili di uso comune per gli utenti (es. cat, cut, grep, find, wc, ls, cp, mv, rm, mkdir, chmod, chown, ...) . Non c'e' una distinzione netta tra /bin, /usr/bin e /sbin
/sbin	Contiene gli eseguibili utilizzati prevalentemente dall'amministratore del sistema (es. chroot, ifconfig, route, ip, adduser, deluser, cron, ...)
/opt	Cartella opzionale utilizzata per installare pacchetti e software di terze parti (es. /opt/intel)
*/lib**	Contiene le librerie di sistema (a volte si distingue tra /lib, /lib32 e /lib64)
/proc	Contiene una gerarchia di file speciali che rappresentano lo stato attuale del sistema. Contiene informazioni sui dispositivi e sull'HW della macchina (cpu, memoria, cores, interrupts, ….)
/mnt	Directory usata per il mount point di FS temporanei (es. quando si monta una chiavetta USB)
/tmp	Directory contenente file temporanei del sistema e degli utenti

File associati alle periferiche /dev

file	Cos'è
/dev/zero	File virtuale che fornisce uno stream di byte NULL (0x00). La scrittura su tale file non ha effetto.
/dev/null	"buco nero" del sistema, qualsiasi stream di dati scritto nel file virtuale /dev/null viene buttato via. La lettura non restituisce nessun valore (es. echo "ciao mondo" > /dev/null)
/dev/full	"device sempre pieno", la scrittura in questo file speciale fallisce sempre con errore ENOSPC ("No space left on device")
/dev/random */dev/urandom*	File speciale che fornisce una interfaccia verso il generatore di numeri (pseudo-)casuali del kernel
/dev/sdX	Dispositivi a blocchi associati a dischi o partizioni di dischi (es. /dev/sda, /dev/sda1)
/dev/ttyX	Dispositivi a caratteri associati a terminali (es. /dev/ttyS0, /dev/tty1, ...) e pseudo-terminali (/dev/ptmx)
/dev/stdout	Descrittore di file associato allo standard output
/dev/stderr	Descrittore di file associato allo standard error
/dev/stdin	Descrittore di file associato allo standard input

Diritti e permessi

- Gli utenti di un sistema UNIX accede ai file in base ai permessi
- Permessi:
  - Proprietario, Gruppo, Altri
    - Read, Write, Execute (rwx)
- Visualizzare permessi : comando `ls`
- Cambiare permessi: comando `chmod` - permessi descritti usando rappresentazione binaria per ogni gruppo.
  - 7 : 111 : rwx
  - 5 : 101 : r-x
  - 4 : 100 : r--

DEMO

I/O in C

- In C i file sono visti come *stream* - flussi di dati - da una fonte a una destinazione
- Libreria `stdio.h` - funzioni della libreria standard per lavorare con file
  - Struct `FILE` - tipo di dato per gestire i file - stream, indicatori di posizione, di fine e di errore.
  - `EOF` - costante che indica la fine di un file
  - `stdin, stdout, stderr` - stream predefiniti per comunicare con tastiera e terminale
  - funzioni per apertura, lettura, scrittura, etc.

Apertura e chiusura del file

```
FILE * fopen ( const char * filename, const char * mode );
```

- apre il file `filename` e ne restituisce lo *stream* (o `NULL` se errore)
- possiamo avere un numero limitato di file aperti (`FOPEN_MAX`)
- `mode` - definisce le operazioni che si possono fare
  - `'r', 'rb'` -lettura, lettura binaria
  - `'w', 'wb'` - scrittura, scrittura binaria
  - `'a'` - append (aggiungere)
  - `'r+','w+','a+','r+b','w+b', 'a+b'` - aperti sia per lettura che scrittura

```
int fclose ( FILE * stream );
```

- chiude il file e disassocia lo stream
- restituisce 0, o `EOF` in caso di errore

```
FILE * freopen ( const char * filename, const char * mode, FILE *
stream );
```

- riutilizza uno *stream* per aprire un altro file o cambiare la modalità di accesso.
- utile per redirezionare gli *stream* standard

Gli *stream* `stdin, stdout, stderr` sono aperti di default all'inizio del programma.

```c
#include <stdio.h>

int main(){
  FILE* fin = fopen("in.txt","r");
  if (fin){
    printf("File aperto\n");
    fclose(fin);
  }
}
```

Tipi di file

- Testuali - rappresentano i dati utilizzando testo e encoding di testo
    - e.g. Numero intero 1 -> ‘1’ -> 49 -> 0x31
- Binari - non utilizzano i caratteri ma salvano direttamente la rappresentazione binaria del tipo di dato
    - e.g. Numero intero 1 -> 0x0001
- Devo leggere i dati utilizzando la stessa rappresentazione (tipo di dato, encoding) con cui sono stati scritti

Lettura file testuale

- `stdin` è uno *stream* - funzioni di lettura
  - Funzioni per leggere testo : `getchar, gets`
  - Funzioni per testo formattato: `scanf`
- Per file e altri stream generici:
  - `int fgetc ( FILE * stream );`
    - legge un carattere e lo restituisce
  - `int fscanf ( FILE * stream, const char * format, ... );`
    - simile a `scanf`
  - `char * fgets ( char * str, int num, FILE * stream );`
    - legge al massimo `num-1` caratteri e li salva in `str`.
    - si ferma a `'\n', EOF` o `num-1` caratteri.
    - restituisce `str` se ha letto qualcosa, o `NULL` se EOF o errore.

Lettura file testuale

- `FILE` - Indicatori EOF e di errore settati se si arriva a EOF o si avverte un errore nell'operazione corrente
- `errno` indica il tipo di errore (`errno.h`)
- Possiamo controllare lo stato:
  - `int feof ( FILE * stream );`
    - non 0 se lo *stream* è arrivato a EOF
  - `int ferror ( FILE * stream );`
    - non 0 se c'è stato un errore
  - `void perror ( const char * str );`
    - stampa un messaggio specifico all'errore codificato in `errno`, eventualmente preceduto da `str`.

```c
#include <stdio.h>
#include <string.h>

#define BUFF 1000

int main(){
  FILE* fin = fopen("inn.txt","r");
  char testo[BUFF];
  int len=0;
  printf("File aperto\n");
  //leggo riga per riga
  while(!feof(fin) && len<BUFF-1 && fgets(&testo[len], BUFF-len, fin)){
      len=strlen(testo);
  }
  if(!feof(fin)){
      printf("Buffer pieno.\n");
  }
  printf("Testo letto:\n%s",testo);
  fclose(fin);
}
```

in.txt

```
1  Ciao a tutti!
2  Ciao Ciao
3
```

```
> ./main
File aperto
testo letto:
Ciao a tutti!
Ciao Ciao
>
```

```c
#include <stdio.h>
#include <string.h>

#define BUFF 1000

int main(){
  FILE* fin = fopen("inn.txt","r");
  char testo[BUFF];
  int len=0;
  printf("File aperto\n");
  //leggo riga per riga
  while(!feof(fin) && len<BUFF-1 && fgets(&testo[len], BUFF-len, fin)){
      len=strlen(testo);
  }
  if(!feof(fin)){
      printf("Buffer pieno.\n");
  }
  printf("Testo letto:\n%s",testo);
  fclose(fin);
}
```

in.txt

```
1  Ciao a tutti!
2  Ciao Ciao
3
```

```
> ./main
File aperto
signal: segmentation fault (core dumped)
```

```c
#include <stdio.h>
#include <string.h>

#define BUFF 1000

int main(){
  FILE* fin = fopen("inn.txt","r");
  char testo[BUFF];
  int len=0;
  if(fin){
    printf("File aperto\n");
    //leggo riga per riga
    while(!feof(fin) && len<BUFF-1 && fgets(&testo[len], BUFF-len, fin)){
      len=strlen(testo);
    }
    if(!feof(fin)){
       printf("Buffer pieno.\n");
    }
    printf("Testo letto:\n%s",testo);
    fclose(fin);
  } else {
    perror("Errore aprendo in.txt ");
  }
}
```

```
> ./main
Errore aprendo in.txt : No such file or directory
>
```

```c
#include <stdio.h>
#include <string.h>

#define BUFF 1000

int main(){
  FILE* fin = fopen("in.txt","r");
  char testo[BUFF];
  int len=0;
  if (fin){
    printf("File aperto\n");
    //leggo riga per riga
    while(!feof(fin) && len<BUFF-1 && fgets(&testo[len], BUFF-len, fin)){
      len=strlen(testo);
    }
    if(ferror(fin))
      perror("Errore durante la lettura");
    else{
      if(!feof(fin)){
        printf("Buffer pieno.\n");
      }
      printf("Testo letto:\n%s",testo);
    }
    fclose(fin);
  } else {
    perror("Errore apprendo in.txt");
  }
}
```

```c
#define BUFF 10
```

```
> ./main
File aperto
Buffer pieno.
Testo letto:
Ciao a tu>
```

```
> ./main
File aperto
testo letto:
Ciao a tutti!
Ciao Ciao
>
```

Scrittura file testuale

- `stdout` - *stream* standard di uscita - funzioni dedicate:
  - `printf, putc, puts`
- Per altri stream:
  - `int fputc ( int character, FILE * stream );`
    - scrive un carattere
    - restituisce il carattere o EOF
  - `int fprintf ( FILE * stream, const char * format, ... );`
    - simile a `printf`
  - `int fputs ( const char * str, FILE * stream );`
    - Scrive tutti i caratteri di `str` in stream.

Lettura file binario

- `size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );`
  - Legge dallo `stream` `count` valori di dimensione `size` (in byte) e li salva in `ptr`.
  - restituisce il numero di byte letti (<=`count`)
  - Se incontra EOF o errore si ferma e setta il corrispettivo indicatore
  - posso usare `fread` per leggere dati di qualsiasi tipo, incluso testo (attenti al ‘\0’).

Scrittura file binario

- `size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );`
  - scrive nello `stream` `count` elementi di dimensione `size`, che trova in memoria all'indirizzo `ptr`

Bufferizzazione

- Le operazioni di scrittura/lettura su disco (o su altri device) sono costose
- A volte vale la pena raggruppare operazioni di scrittura di pochi dati in un unica operazione
- Gli *stream* e operazioni della libreria standard sono bufferizzate
  - Completamente (*fully buffered*)
  - Per riga (*line buffered*) - se device interattivo, dipende dall'implementazione
- `int fflush ( FILE * stream );`
  - su stream aperti in modalità `'w'` forza la scrittura svuotando il buffer
  - chiamata da `fclose`
- `void setbuf ( FILE * stream, char * buffer );`
  - Assegna `buffer` allo stream che diventa *fully buffered*. Se buffer è `NULL` lo *stream* diventa *unbuffered*.

Posizionarsi nel file

- Ogni stream ha associato un indicatore di posizione - modificato dalle funzioni di lettura/scrittura
- Possiamo anche leggerlo e scriverlo ‘a mano’
- `long int ftell ( FILE * stream );`
  - Restituisce il valore dell’indicatore. Per file binari è il numero di byte dall’inizio del file. Per file testuali dipende, può comunque essere utilizzato.
- `int fseek ( FILE * stream, long int offset, int origin );`
  - Imposta l’indicatore di posizione a `offset` byte dopo `origin`
  - `origin` deve essere uno tra
    - `SEEK_SET` - inizio del file
    - `SEEK_CUR` - posizione corrente
    - `SEEK_END` - fine del file
  - su file di testo `offset` deve essere 0 o una posizione precedente (`ftell`), origin deve essere `SEEK_SET`
- `void rewind ( FILE * stream );`
  - Rimette l’indicatore all’inizio.

Editare dei file

- Aprendo in modalità `'a'` - append
  - Aggiungo contenuti alla fine del file
- Per editare campi ‘interni’
  - File di testo
    - fattibile se il numero di caratteri da cancellare è lo stesso al numero di caratteri da inserire (e.g. trasformare in maiuscolo)
    - difficile se i dati da inserire hanno una dimensione diversa - va riscritto tutto il resto del file
  - File binario
    - più facile rispetto al file testuale se il tipo di dato è lo stesso (e.g. devo rimpiazzare un intero)

Esercizi

1. Leggere da tastiera 10 interi e scriverli in un file. Provare sia con file testuale che con file binario.

2. Aprire il file dell'esercizio 1 e sovrascrivere il terzo numero con un nuovo intero letto da tastiera. Poi rileggere l'intero file e stampare a video i numeri.

Domande?