# LabSO 2023

## Laboratorio Sistemi Operativi - A.A. 2022-2023

Michele Grisafi - michele.grisafi@unitn.it

# C - files

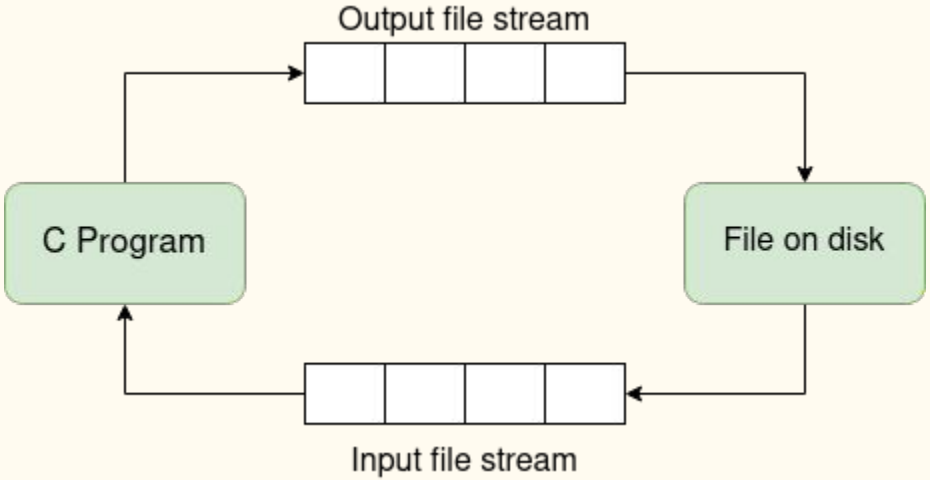
# Interazione con i file

In UNIX ci sono due modi per interagire con i file: **streams** e **file descriptors.**

- **Streams**: forniscono strumenti come la formattazione dei dati, bufferizzazione, ecc...
- **File descriptors**: interfaccia di basso livello costituita dalle system call messe a disposizione dal kernel.

# Interazione con i file - Streams

Utilizzando gli streams, un file è descritto da un puntatore a una struttura di tipo FILE (definita in stdio.h). I dati possono essere letti e scritti in vari modi (un carattere alla volta, una linea alla volta, ecc.) ed essere interpretati di conseguenza.



# Aprire e chiudere un file

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

Restituisce un FILE pointer (o NULL se errore) per gestire il `filename` nella modalità specificata da `mode.` Questa può essere:

- `r`: read
- `w`: write or overwrite (create)
- `r+`: read and write
- `w+`: read and write. Create or overwrite
- `a`: write at end (create)
- `a+`: read and write at end (create)

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

# Leggere un file

`int fgetc(FILE *stream)`
Restituisce un carattere dallo stream.

`char *fgets(char *str, int n, FILE *stream)`
Restituisce una stringa da `stream` e la salva in `str`. Si ferma quando `n-1` caratteri sono stati letti, una nuova linea (`\n`) è letta o la fine del file viene raggiunta. Inserisce anche il carattere di terminazione e, eventualmente, '`\n`'..

`int fscanf(FILE *stream, const char *format, ...)`
Legge da `stream` dei dati, salvando ogni dato nelle variabile fornite (simile a `printf`) seguendo la stringa `format`.

`int feof(FILE *stream)`
Restituisce 1 se lo `stream` ha raggiunto la fine del file, zero altrimenti.

# Scrivere su un file

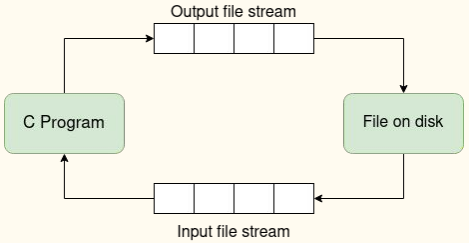
`int fputc(int char, FILE *stream)`
Scrive un singolo carattere `char` su `stream`.

`int fputs(const char *str, FILE *stream)`
Scrive una stringa `str` su `stream` senza includere il carattere null.

`int fprintf(FILE *stream, const char *format, ...)`
Scrive il contenuto di alcune variabile su `stream`, seguendo la stringa `format`.

E molte altre....

# Flush e rewind




Seguendo l'immagine, il contenuto di un file viene letto e scritto con degli streams (dei buffer) di dati. Come tali, è comprensibile come queste operazioni non siano immediate: i dati vengono scritti sul buffer e solo successivamente scritti sul file. Il **flush** è l'operazione che trascrive il file dallo stream. Questa operazione avviene quando:

- Il programma termina con un `return` dal main o con `exit()`.
- `fprintf()` inserisce una nuova riga.
- `int fflush(FILE *stream)` viene invocato.
- `void rewind(FILE *stream)` viene invocato.
- `fclose()` viene invocato

`rewind` consente inoltre di ripristinare la posizione della testina all'inizio del file.

# File Streams

filename.txt:

```
1 Nome1 Cognome1
2 Nome2 Cognome2
3 Nome3 Cognome3
```

```c
#include <stdio.h>

FILE *ptr;  //Declare stream file
ptr = fopen("filename.txt","r+"); //Open

int id;
char str1[10], str2[10];
while (!feof(ptr)){ //Check end of file
    //Read int, word and word
    fscanf(ptr,"%d %s %s", &id, str1, str2);
    printf("%d %s %s\n",id,str1,str2);
}

printf("End of file\n");
fclose(ptr); //Close file
```

# File Streams

```
#include <stdio.h>
#define N 10

FILE *ptr;
ptr = fopen("fileToWrite.txt","w+");
fprintf(ptr,"Content to write"); //Write content to file
rewind(ptr); // Reset pointer to begin of file
char chAr[N], inC;
fgets(chAr,N,ptr); // store the next N-1 chars from ptr in chAr
printf("'%d' '%s'",chAr[N-1], chAr);
do{
    inC = fgetc(ptr); // return next available char or EOF
    printf("%c",inC);
}while(inC != EOF); printf("\n");
fclose(ptr);
```

# File Descriptors

Un file è descritto da un semplice **intero** (file descriptor) che punta alla rispettiva entry nella file table del sistema operativo. I dati possono essere letti e scritti soltanto un buffer alla volta di cui spetta al programmatore stabilire la dimensione.

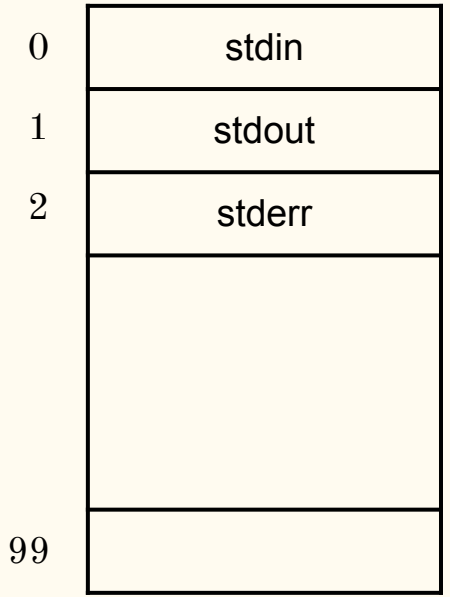
Un insieme di system call permette di effettuare le operazioni di input e output mantenendo un controllo maggiore su quanto sta accadendo a prezzo di un'interfaccia meno amichevole.

# File Descriptors

Per accedere al contenuto di un file bisogna creare un canale di comunicazione con il kernel, aprendo il file con la system call open la quale localizza l'i-node del file e aggiorna la *file table* del processo.

A ogni processo è associata una tabella dei file aperti di dimensione limitata, dove ogni elemento della tabella rappresenta un file aperto dal processo ed è individuato da un indice intero (il "file descriptor")

I file descriptor 0, 1 e 2 individuano normalmente standard input, output ed error (aperti automaticamente)

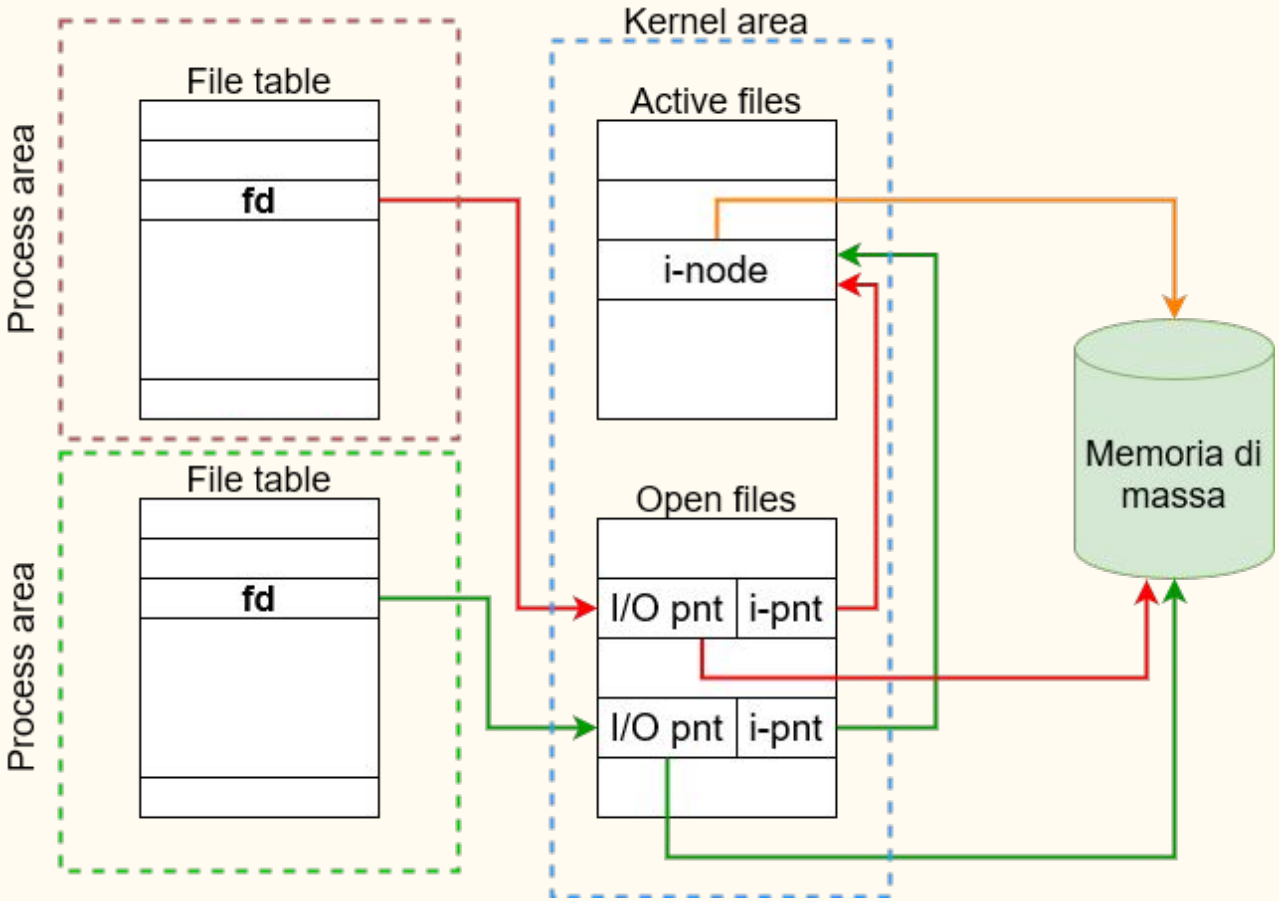


# File Descriptors

Il kernel gestisce l'accesso ai files attraverso due strutture dati: **la tabella dei files attivi e la tabella dei files aperti**. La prima contiene una copia dell'i-node di ogni file aperto (per efficienza), mentre la seconda contiene un elemento per ogni file aperto e non ancora chiuso. Questo elemento contiene:

- I/O pointer: posizione corrente nel file
- i-node pointer: Puntatore a inode corrispondente

La tabella dei file aperti può avere più elementi corrispondenti allo stesso file!


Kernel area
Process area
File table
fd
Active files
i-node
Memoria di massa
Process area
File table
fd
Open files
I/O pnt
i-pnt
I/O pnt
i-pnt

# Aprire e chiudere un file

```
int open(const char *pathname, int flags[, mode_t mode]);
```

`flags:` interi (ORed) che definiscono l'apertura del file. I più comuni:

- `O_RDONLY, O_WRONLY, O_RDWR`: almeno uno è obbligatorio.
- `O_CREAT`: crea il file se non esiste (con `O_EXCL` la chiamata fallisce se esiste)
- `O_APPEND`: apre il file in append-mode (auto lseek con ogni scrittura)
- `O_TRUNC`: cancella il contenuto del file (se usato con la modalità scrittura)

`mode`: interi (ORed) per i privilegi da assegnare al nuovo file: `S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXU, S_IRGRP, …, S_IROTH`

```
int close(int fd);
```

# Leggere e scrivere un file

`ssize_t` **`read`** `(int fd, void *buf, size_t count);`
Legge dal file e salva nel buffer `buf` fino a `count` bytes di dati dal file associato con il file descriptor `fd`.

`ssize_t` **`write`**`(int fd, const void *buf, size_t count);`
Scrive sul file associato al file descriptor `fd` fino a `count` bytes di dati dal buffer `buf`.

`off_t` **`lseek`**`(int fd, off_t offset, int whence);`
Riposiziona l'offset del file a seconda dell'argomento `offset` partendo da una certa posizione `whence`. `SEEK_SET:` inizio del file, `SEEK_CUR:` dalla posizione attuale, `SEEK_END:` dalla fine del file.

# Example

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

//Open new file in Read only
int openedFile = open(“filename.txt”, O_RDONLY);
char content[10]; int canRead;
do{
    bytesRead = read(openedFile,content,9); //Read 9B to content
    content[bytesRead]=0;
    printf(“%s”,content);
} while(bytesRead > 0);
close(openedFile);
```

# Example

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
//Open file (create it with user R and W permissions)
int openFile = open(“name.txt”, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR);
char toWrite[] = “Professor”;

write(openFile, "hello  world\n", strlen("hello world\n")); //Write to file
lseek(openFile, 6, SEEK_SET); // move I/O pointer
write(openFile, toWrite, strlen(toWrite)); //Write to file

close(openFile);
```

# C - files: canali standard I

- I canali standard (in/out/err che hanno indici 0/1/2 rispettivamente) sono rappresentati con strutture “stream” (`stdin`, `stdout`, `stderr`) e macro (`STDIN_FILENO`, `STDOUT_FILENO`, `STDERR_FILENO`).

- La funzione `fileno` restituisce l’indice di uno “stream”, per cui si ha:
  - `fileno(stdin)=STDIN_FILENO   // = 0`
  - `fileno(stdout)=STDOUT_FILENO // = 1`
  - `fileno(stderr)=STDERR_FILENO // = 2`
- `isatty(stdin) == 1` (se l’esecuzione è interattiva) OPPURE 0 (altrimenti)

`printf("ciao");` e `fprintf(stdout, "ciao");` sono equivalenti!

# C - files: canali standard II

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

void main() {
  printf("stdin:  stdin ->_flags = %hd, STDIN_FILENO  = %d\n",
    stdin->_flags,  STDIN_FILENO
  );
  printf("stdout: stdout->_flags = %hd, STDOUT_FILENO = %d\n",
    stdout->_flags, STDOUT_FILENO
  );
  printf("stderr: stderr->_flags = %hd, STDOUT_FILENO = %d\n",
    stderr->_flags, STDERR_FILENO
  );
}
```

# C - piping con bash

# Piping via bash

- Normalmente un'applicazione eseguita da bash ha accesso ai canali standard stdin, stdout e stderr (tastiera/video).
- Se le applicazioni sono usati in un'oprazione di piping (ee. `ls | wc -l`) allora l'output dell'applicazione sulla sinistra diventa l'input dell'applicazione sulla destra.
- Attenzione che l'esecuzione è parallela! Entrambi i comandi sono eseguiti allo stesso momento.

# Example

```
#define MAXBUF 10
#include <stdio.h>
#include <string.h>

int main() {
    char buf[MAXBUF];
    fgets(buf, sizeof(buf), stdin); // may truncate!
    printf("%s\n", buf);
    return 0;
}
```

```
$ gcc src.c -o pip.out
$ echo “hi how are you” | ./pip.out
```

# Example

```
$ gcc src.c -o inv.out
$ ls /tmp | ./inv.out
```

```c
#include <stdio.h>

int main() {
    int c, d;
    // loop into stdin until EOF (as CTRL+D)
    // read from stdin
    while ((c = getchar()) != EOF) {
        d = c;
        if (c >= 'a' && c <= 'z') d -= 32;
        if (c >= 'A' && c <= 'Z') d += 32;
        putchar(d); // write to stdout
    };
    return (0);
}
```

Esempio di una semplice applicazione che legge da stdin e stampa su stdout, invertendo i caratteri minuscoli con quelli maiuscoli.

# Esercizi

Crea un’applicazione che copia il contenuto di un file, leggendolo con i file streams e scrivendolo con i file descriptors. Crea poi un programma che fa il contrario. Prova a copiare carattere per carattere e linea per linea.

# CONCLUSIONI

Esistono diversi modi per interagire con i file: flussi e descrittori di file. Ogni approccio ha i suoi pro e i suoi contro. Entrambi ruotano attorno a una diversa rappresentazione di un file: gli stream operano su puntatori di file (cioè buffer), mentre i descrittori di file operano su interi (cioè voci di una struttura dati).