

LabSO 2021

Laboratorio Sistemi Operativi - A.A. 2020-2021

dr. Andrea Naimoli	Informatica LT andrea.naimoli@unitn.it
dr. Michele Grisafi	Ingegneria informatica, delle comunicazioni ed elettronica (LT) michele.grisafi@unitn.it

C - struttura

—

main.c

- A parte casi particolari (es. sviluppo moduli per kernel) l'applicazione deve avere una funzione “main” che è utilizzata come punto di ingresso
- Il tipo della funzione è void (se non specifichiamo un valore di ritorno) o int (se è esplicitamente indicato un valore di ritorno)
- Il valore di ritorno è un intero che rappresenta il codice di uscita dell'applicazione (variabile \$? in bash) ed è 0 di default se omesso
- Quando la funzione è invocata riceve normalmente in input il numero di argomenti (incluso l'eseguibile) e la lista come “vettore di stringhe” (*)

(*) in C una stringa è in effetti un vettore di caratteri, quindi un vettore di stringhe è un vettore di vettori di caratteri, inoltre i vettori in C sono sostanzialmente puntatori (al primo elemento del vettore)

esempio

Compilazione:

```
gcc main.c -o main
```

Esecuzione:

```
./main arg1 arg2
```

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("%d\n", argc);
    printf("%s\n", argv[0]);
    return 0;
}
```

In output si ha “3” (numero argomenti incluso il file eseguito) e “./main” (primo degli argomenti).

In generale quindi `argc` è sempre maggiore di zero.

Direttive e istruzioni fondamentali

- `#include ... / #define ... /`
- `char / int / ... / enum (v. esempio seguente)`
- `for (initialization ; test; increment) { ... ; }`
- `break / continue`
- `switch (expression){ case val: ... [break;] [default: ...] }`
- `while (expression) { ... } / do { ... } while (expression)`
- `if (expression) { ... } [else { ... }]`
- `struct / union`

(consultare una documentazione standard ed esercitarsi)

Librerie standard

Librerie possono essere usate attraverso la direttiva `#include`.

Tra le più importanti vi sono:

- `stdio.h`: `FILE`, `EOF`, `stderr`, `stdin`, `stdout`, `fclose()`, etc...
- `stdlib.h`: `atof()`, `atoi()`, `malloc()`, `calloc()`, `free()`, `exit()`, `system()`, `rand()`, etc...
- `string.h`: `memset()`, `memcpy()`, `strncat()`, `strcmp()`, `strlen()`, etc...
- `math.h`: `sin()`, `cos()`, `sqrt()`, `floor()`, etc...
- `unistd.h`: `STDOUT_FILENO`, `read`, `write`, `fork`, `pipe`, etc...
- `fcntl.h`: `creat`, `open`, etc...

...e ce ne sono molte altre.

Direttive - esempi

```
#include <stdio.h>
#define ITER 5
#define POW(A) A*A

int main(int argc, char **argv) {
#ifdef DEBUG
    printf("%d\n", argc);
    printf("%s\n", argv[0]);
#endif
    int res = 1;
    for (int i = 0; i < ITER; i++){
        res *= POW(argc);
    }
    return res;
}
```

```
gcc main.c -o main.out -D DEBUG=0
```

```
gcc main.c -o main.out -D DEBUG=1
```

```
./main.out 1 2 3 4
```

Stesso risultato!

Structs e Unions

Structs permettono di aggregare diverse variabili, mentre le unions permettono di creare dei tipi generici che possono ospitare uno di vari tipi specificati.

```
struct Books{
    char author[50];
    char title[50];
    int bookID;
} book1, book2;

struct Books book3 =
{"Rowling", "Harry Potter", 2};
strcpy(book1.title, "Moby Dick");
book2.bookID = 3;
```

```
union Result{
    int intero;
    float decimale;
} result1, result2;

union Result result3;
result3.intero = 22;
result3.decimale = 11.5;
```


Typedef

Typedef consente la definizione di nuovi tipi di variabili o funzioni.

```
typedef unsigned int intero;  
  
typedef struct Books{  
    ...  
} bookType;  
  
intero var = 22;  
bookType book1;
```

C - esempio “enum”

```
#include <stdio.h>

enum State {Undef = 9, Working = 1, Failed = 0};
void main() {
    enum State state=Undef;
    printf("%d\n", state);
}
```

C - files

—

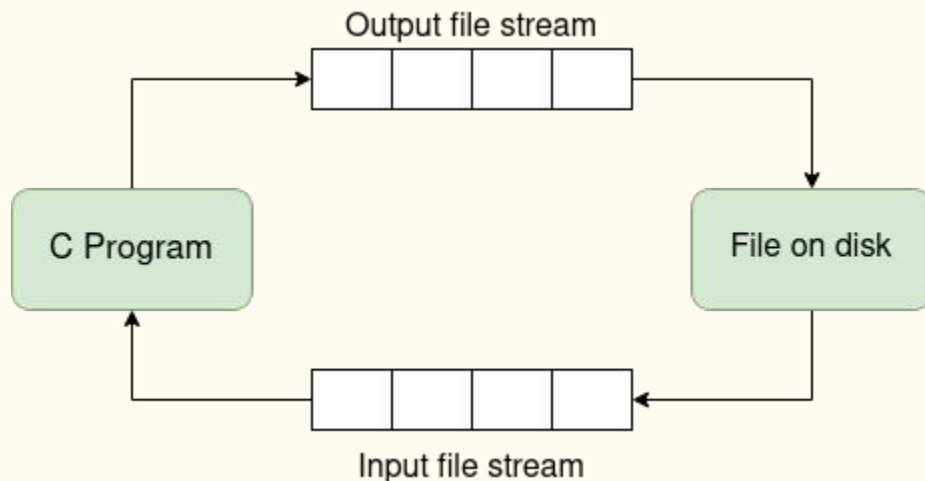
Interazione con i file

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

- Streams: fornisce 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 - Stream

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.



Interazione con i file - Stream

```
#include <stdio.h>

FILE *ptr;
ptr = fopen("filename.txt", "r+");

int id;
char str1[10], str2[10];
while (!feof(ptr)){
    fscanf(ptr, "%i %s %s", &id, str1, str2);
    printf("%i %s %s\n", id, str1, str2);
}

printf("End of file");

fclose(ptr);
```

filename.txt:

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

Interazione con i file - Stream

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

FILE *ptr;
ptr = fopen("filename.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); // next N chars stored in chAr
printf("%s",chAr);
do{
    inC = fgetc(ptr); // next available char or EOF
    printf("%c",inC);
}while(inC != EOF);
fclose(ptr);
```

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

```
// stdch.c

#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, STDERR_FILENO = %d\n",
        stderr->_flags, STDERR_FILENO
    );
}
```

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 (circa 100 elementi), 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)

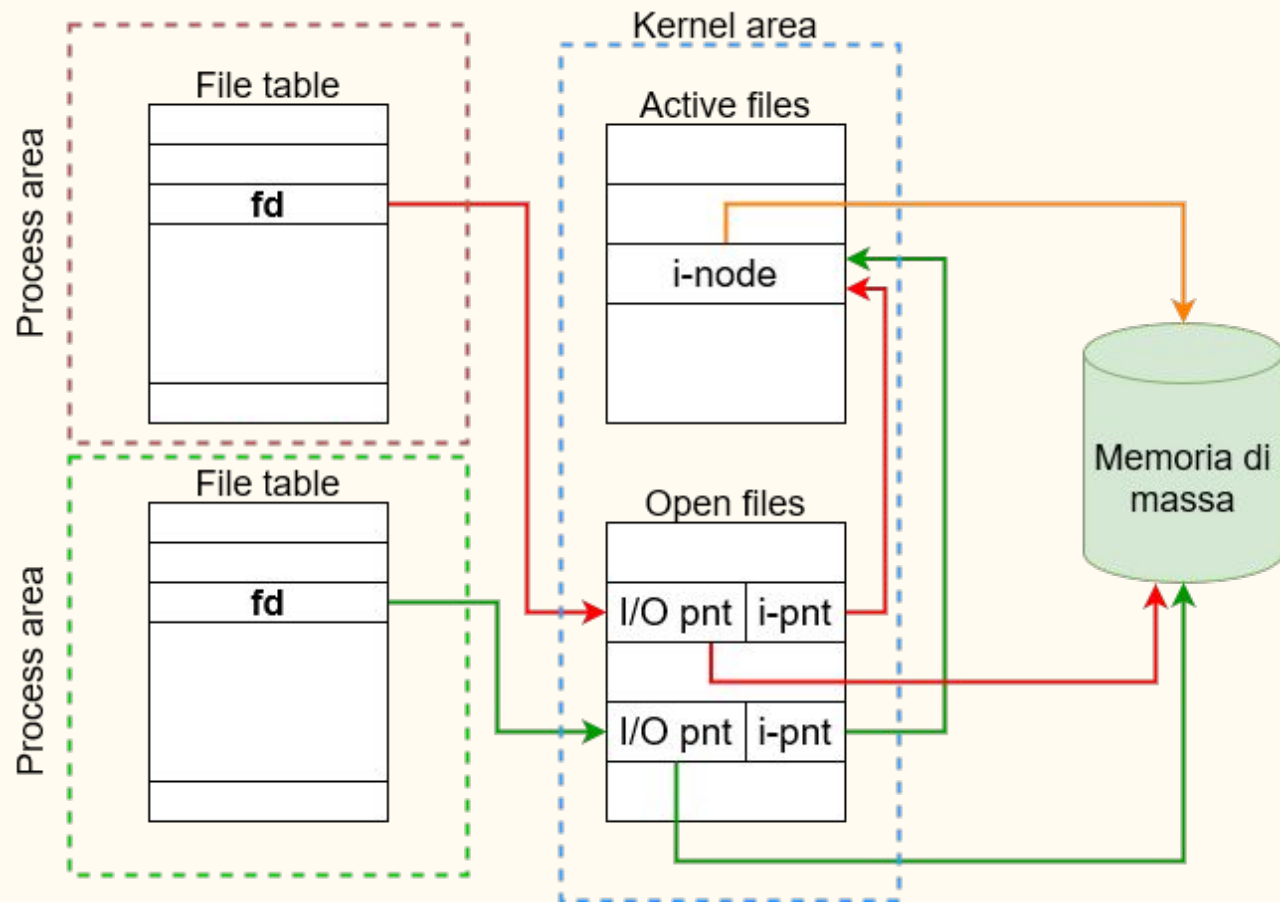
0	stdin
1	stdout
2	stderr
99	

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'inode 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!



Interazione files - File Descriptors

L'Input/Output Unix è basato essenzialmente su cinque funzioni: **open**, **read**, **write**, **lseek** e **close**.

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

int newFile = creat("name.txt",S_IRUSR|S_IWUSR); //Create a file with read/write permission
int newFile2 = open("name.txt", O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);

char content[10]; int canRead;
do{
    canRead = read(newFile2,content,10);
    printf("%s",content);
} while(canRead >0);

close(newFile); close(newFile2);
```

Interazione files - File Descriptors

L'Input/Output Unix è basato essenzialmente su cinque funzioni: **open**, **read**, **write**, **lseek** e **close**.

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

int openFile = open("name.txt", O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);
char toWrite[] = "Professor";

write(openFile, "hello world\n", strlen("hello world\n"));
lseek(openFile, 6, SEEK_SET); // riposiziona l'I/O pointer
write(openFile, toWrite, strlen(toWrite));

close(openFile);
```

C - funzioni/operatori

— generali e di uso comune

printf / fprintf

```
int printf(const char *format, ...)  
int fprintf(FILE *stream, const char *format, ...)
```

Inviano dati sul canale stdout (`printf`) o su quello specificato (`fprintf`) secondo il formato indicato.

Il formato è una stringa contenente contenuti stampabili (testo, a capo, ...) ed eventuali segnaposto identificabili dal formato generale:

`%[flags][width][.precision][length]specifier`

Ad esempio: `%d` (intero con segno), `%c` (carattere), `%s` (stringa), ...

Ad ogni segnaposto deve corrispondere un ulteriore argomento del tipo corretto.
(rivedere esempio precedente)

sizeof (*operatore*)

`sizeof (type) / sizeof expression`

Si tratta di un operatore che elabora il tipo passato come argomento (tra parentesi) o quello dell'espressione e restituisce il numero di bytes occupati in memoria.

```
#include <stdio.h>
void main() {
    int x = 10;
    printf("variable x      : %lu\n", sizeof x);
    printf("expression 1/2 : %lu\n", sizeof 1/2);
    printf("int type       : %lu\n", sizeof(int));
    printf("char type      : %lu\n", sizeof(char));
    printf("float type     : %lu\n", sizeof(float));
    printf("double type    : %lu\n", sizeof(double));
}
```

exit

```
void exit(int status)
```

Il processo è terminato restituendo il valore `status` come codice di uscita. Si ottiene lo stesso effetto se all'interno della funzione `main` si ha `return status`.

La funzione non ha un valore di ritorno proprio perché non sono eseguite ulteriori istruzioni dopo di essa.

Il processo chiamante è informato della terminazione (*)

(*) tale comunicazione avviene tramite un “segnale” apposito. I segnali sono trattati più avanti nel corso.

C - vettori e stringhe

—

C - vettori I

I vettori sono sequenze di elementi omogenei (tipicamente liste di dati dello stesso tipo, ad esempio liste di interi o di caratteri).

I vettori si realizzano con un puntatore al primo elemento della lista.

Ad esempio con `int arr[4] = {2, 0, 2, 1}` si dichiara un vettore di 4 interi inizializzandolo: sono riservate 4 aree di memoria consecutive di dimensione pari a quella richiesta per ogni singolo intero (tipicamente 2 bytes, quindi $4*2=8$ in tutto)

C - vettori II

`char str[5] = {'c', 'i', 'a', 'o', 0} : 5*1 = 5 bytes`

`str` è dunque un puntatore a `char` (al primo elemento) e si ha che:

`str[n]` corrisponde a `*(str+n)`

e in particolare `str[0]` corrisponde a `*(str+0)=*(str)=*str`

C - stringhe

Le stringhe in C sono vettori di caratteri, ossia puntatori a sequenze di bytes, la cui terminazione è definita dal valore convenzionale 0 (zero).

Un carattere tra apici singoli equivale all'intero del codice corrispondente.

In particolare un vettore di stringhe è un vettore di vettore di caratteri e dunque:

```
char c;           #carattere  
char *str;        #vettore di caratteri / stringa  
char **strarr;    #vettore di vettore di caratteri / vettore di stringhe
```

Si comprende quindi la segnatura della funzione main con ****argv**.

C - esempio carattere e argc/argv

```
#include <stdio.h>

int main(int argc, char **argv) {
    int code=0;
    if (argc<2) {
        printf("Usage: %s <carattere>\n", argv[0]);
        code=2;
    } else {
        printf("%c == %d\n", argv[1][0], argv[1][0]);
    };
    return code;
};
```


C - argomenti da CLI

- Per il parsing degli argomenti da CLI la libreria `getopt.h` mette a disposizione `getopt` e `getopt_long`. (v.
https://www.gnu.org/software/libc/manual/html_node/Getopt.html)
- Si può effettuare un parsing manuale scorrendo gli argomenti.

C - parsing manuale argomenti: esempio

```
#define MAXOPTL 64
#define MAXOPTS 10

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

char opt[MAXOPTS][MAXOPTL]; // array of options: keys
char val[MAXOPTS][MAXOPTL]; // array of options: values

void main(int argc, char **argv) {
    int a=0, o=0;
    // loop into arguments:
    while (++a<argc && o<MAXOPTS) {
        if (strcmp("-h", argv[a])==0) {
            strcpy(opt[o++], "help");
        }
    }
```

```
        if (strcmp("-k", argv[a])==0) {
            strcpy(opt[o++], "key");
            if (a+1<argc) {
                strcpy(val[o-1], argv[++a]);
            }
        }
    }
    // dump options (keys/values):
    for (a=0; a<o; a++) {
        printf("opt[%d]: %s,%s\n", a, opt[a], val[a]);
    }
}
```

C - esercizi per casa

1. Scrivere un'applicazione che data una stringa come argomento ne stampa a video la lunghezza, ad esempio:
`./lengthof "Questa frase ha 28 caratteri"`
deve restituire a video il numero 28.
2. Scrivere un'applicazione che definisce una lista di argomenti validi e legge quelli passati alla chiamata verificandoli e memorizzando le opzioni corrette, restituendo un errore in caso di un'opzione non valida.
3. Realizzare funzioni per stringhe `char *stringrev(char str)` (inverte ordine caratteri) e `int stringpos(char str, char chr)` (cerca *chr* in *str* e restituisce la posizione)

(In tutti i casi si può completare l'esercizio gestendo gli eventuali errori di immissione da parte dell'utente come parametri errati o altro)

C - funzioni stringhe <string.h>

Dato che le stringhe sono riferite con un puntatore al primo carattere non ha senso fare assegnamenti e confronti diretti, ma si devono usare delle funzioni.

La libreria standard string.h ne definisce alcune come ad esempio:

`char *strcat(char *dest, const char *src)` append di src in coda a dest
`char *strchr(const char *str, int c)` cerca la prima occorrenza di c in str
`int strcmp(const char *str1, const char *str2)` confronta str1 con str2
`size_t strlen(const char *str)` calcola la lunghezza di str (*)

(*) `size_t` è un intero senza segno definito dalla stessa libreria

C - piping via bash

—

C - piping via bash

- In condizioni normali l'applicazione richiamata da bash ha accesso ai canali stdin, stdout e stderr comuni (tastiera/video).
- Se l'applicazione è inserita via bash in un “piping” (come in `ls | wc -l`) allora:
 - Accede all'output del comando a sinistra da stdin
 - Invia il suo output al comando di destra su stdout

```
#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;
}
```

C - esempio piping da bash

Esempio di una semplice applicazione che legge da `stdin` e stampa su `stdout` invertendo minuscole [a-z] con maiuscole [A-Z]

```
/* swap "stdin" lowercase [a-z] and uppercase [A-Z] */  
  
#include <stdio.h>  
  
int main() {  
    int c, d;  
    // loop into stdin until EOF (as CTRL+D)  
    while ((c = getchar()) != EOF) { // read from stdin  
        d = c;  
        if (c >= 'a' && c <= 'z') d -= 32;  
        if (c >= 'A' && c <= 'Z') d += 32;  
        putchar(d);                // write to stdout  
    };  
    return (0);  
}
```

CONCLUSIONI

Comprendendo il funzionamento dei vari tipi di variabili, in particolare la gestione dei puntatori e dei vettori, e sfruttando poi le funzioni illustrate è possibile realizzare delle applicazioni che manipolano argomenti passati via CLI o anche interagire con processi terzi (in particolare attraverso il file-system o via bash con il piping).