# SO. FUNZIONI IMPORTANTI

USER LINUX: SO

PASSWORD: SO

INDIRIZZO IP VIA ETHERNET [**so@192.168.178.128**](mailto:so@192.168.178.128)

## FUNZIONE RANDOM

**srand (getpid ()\*time(NULL))** //inizializzazione prima di usarla

**int**

**int value = 1 + rand () % 9** // valori da 1 a 10

**int value [0] = 1 + rand () % 9** // valori da 1 a 10 nel primo elemento dell’array

**int value [0] [0] = 1 + rand () % 9** // valori da 1 a 10 nel primo elemento della matrice

**for(int i=0; i < strlen (value); i++) value [i] = 1+ rand () % 9** // valori da 1 a 10 in tutto l’array

**for(int i=0; i < strlen (value); i++) { for(int j=0; j <strlen (value); j++){**

**value [i] [j] = 1+ rand () % 9 }** // valori da 1 a 10 in tutta la matrice

**char**

**char value = ‘A’ + ( rand () % 26 )** //valori da A a Z

**char value [0] = ‘****A’ + ( rand () % 26 )** //valori da A a Z nel primo elemento dell’array

**for(int i=0; i < strlen (value); i++) value [i] = A’ + ( rand () % 26 )** // valori da A a Z in tutto l’array

## PRINTF

N.B STAMPA A VIDEO DEI VALORI

**% d →**  INT , INT [ ], INT [ ] [ ] , \* INT // NELLA PRINTF SI USA PRIMA DEL NOME IL \*

**% f →**  FLOAT , FLOAT [ ], FLOAT [ ] [ ] ,\* FLOAT , DOUBLE // es. **%1.2f** DOVE 2 INDICA I NUM DOPO LA VIRGOLA

**% p →**  \* INT , \* CHAR , \* FLOAT // STAMPA INDIRIZZO

**% c →**  CHAR // SINGOLO ELEMENTO

**% s →**  CHAR [ ] // STAMPA INTERA STRINGA MA NELLA PRINTF SI USA SOLO IL NOME

**% ld →**  // NUMERI INTERI LUNGHI (STRLEN)

**% u →**  // INT SENZA SEGNO

**% x →**  INT , INT [ ], INT [ ] [ ] , \* INT // ESADECIMALE

**% e →**  FLOAT , DOUBLE //FORMATO ESPONEZIALE

**% o →** INT , INT [ ], INT [ ] [ ] , \* INT // INT IN OTTALE

ES. **PRINTF (“ STRINGA %s , MOLTIPLICAZIONE %d , PUNTATORE %d “ , STRINGA , VAL1\*VAL2 , \*PUNTATORE )**

## SCANF

// INSERIMENTO DA TASTIERA IN RUNTIME DEI VALORI

**SCANF (“ % d “ , & val1)** // INT

**SCANF (“ % d “ , &\*val1)** // \* INT

**SCANF (“ % f “ , & val1)** // FLOAT

**SCANF (“ % lf “ , & val1)** // DOUBLE

**SCANF (“ % c “ , & val1)** // CHAR

**SCANF (“ % s “ , val1)** // CHAR [ ]

es. **SCANF (“VALUE: %d”, &val1) PRINTF(“STAMPA: %d”,val1)**

// so@linux~ **./eseguibile**

**VALUE: 12** **STAMPA: 12**

es. **PRINTF (“VALUE: “) SCANF (“%d”, &val1) PRINTF(“STAMPA: %d”,val1)**

// so@linux~ **./eseguibile**

**VALUE: 12** **STAMPA: 12**

## COMANDI PER TERMINALE

**char c = getchar ( )** // ASPETTA TASTO DA TASTIERA E POI PROSEGUE , **LINUX**

**system(“clear” )** // SI USA NEL PROGRAMMA PER PULIRE IL TERMINALE , **LINUX**

**system(“pause” )** // ASPETTA TASTO DA TASTIERA E POI PROSEGUE , **DEV**

**system(“cls” )** // SI USA NEL PROGRAMMA PER PULIRE IL TERMINALE , **DEV**

## REPOSITORY GIT

**git clone** indirizzoweb // CREAZIONE

**git status** // CONTROLLA LO STATO

**git pull**  // SE LO STATO TUTTO OK POSSO VEDERE AGGIORNAMENTI

**git restore** nomefile // RIPRISTINA FILE O DIRECTORY

**git add** nomefile // AGGIUNGE FILE MANCANTI

## COMANDI LINUX

## **COMANDI** **UTILIZZO**

**apt-get install** nomeapp **installare app**

nomeapp **-- version**

**sudo su** **modalità sviluppatore**

**clear** **pulizia terminale**

**man** nomeapp **visualizza manuale**

**pwd visualizza nome della cartella o directory corrente**

**ls lista degli elementi della cartella corrente**

**ls-l lista dettagliata degli elementi della cartella corrente**

**cd** directory  **sposto nella directory selezionata**

**cd ..** directory  **sposto di 1 posizione nella director selezionata**

**cd .. / ..** directory **sposto di 2 posizioni nella director selezionata**

**mkdir** nuovacartella **crea una nuova cartella con nome** nuovacartella

**psaux visualizza i processi**

**top visualizza i processi**

**vim nomefile** .c / .cpp / makefile **crea e si apre un file di nome nomefile.** estensione per modificare press tasto **( a )** , per uscire press tasto **( ctrl+c ,** poi **:wq** per salvare )

**gcc** nomefile**.c -o** nomeeseguibile **mi compila il programma c** nomefile **in** nomeeseguibile

**g++** nomefile**.cpp -o** nomeeseguibile **mi compila il programma c++** nomefile **in** nomeeseguibile

**./**nomeeseguibile **lancio eseguibile**

**directoryeseguibile/**nomeeseguibile **identico a prima**

**cat** nomefile **visualizza a terminale qualunque file**

**cp** nomefile nomefilecopia **copiare file**

**mv** nomefile nomefilerinominato **rinominare file**

**mv** nomefile/**nuovadirectory spostamento file**

**rm** directory/nomefile **rimozione file**

**rm -r** directory **rimozione directory**

**rm -rf** nomefile **rimozione file**

**echo** “helloWord” **visualizza stringa sul terminale**

**echo** “helloWord” > nomefile **inserisce la stringa nel file**

**echo** “2024” >> nomefile **aggiunge la striscia al file**

**cat**  nomefile **| sort ordinamento nomefile**

**cat** nomefile **| wc-l lunghezza stringa**

**cat** nomefile **| sort >>** fileordinato **ordinamento e lo inserisci in** fileordinato

### COMANDO ls – l

Composto da :

**PRIMO BIT:**

* **d** directory
* **-** file normale

**TRE TERNE:**

1. È relativa al proprietario del file
2. È relativa al gruppo associato di cui fa parte il proprietario
3. È relativa ai permessi per gli altri utenti

**VALORI TERNE:**

* **r :** lettura (read)
* **w :** scrittura (write)
* **x :** esecuzione (eXecution)

**ESEMPIO ESEGUIBILE:**

so@linux ~ **ls -l nomeeseguibile:**

**− rwr rwr rwr**

in ottale sarebbe :  **111 111 111 = 7 7 7 PERMESSI MAX**

**ottale scala:** 000 = **0** , 001 = **1** ,010 = **2** , 011 = **3** ,100 = **4** ,101 = **5** , 110 = **6** ,111 = **7**

#### CHMOD

**chmod** ternaottale nomefile

**chmod +r** nomefile **aggiunge il permesso di lettura**

**chmod +w** nomefile **aggiunge il permesso di scrittura**

**chmod +x** nomefile **aggiunge il permesso di esecuzione**

**chmod -r** nomefile **toglie il permesso di lettura**

**chmod -w** nomefile **toglie il permesso di scrittura**

**chmod -x** nomefile **toglie il permesso di esecuzione**

**es.**

so@linux~ **ls - l** -rw-rw-r-- nomefile

so@linux~ **chmod – w** nomefile  **~ -r--r--r--** nomefile

so@linux~ **chmod + w** nomefile **~ -rw-rw-r--** nomefile

so@linux~ **chmod 111** nomefile **~ ---x--x--x** nomefile

so@linux~ **chmod 123** nomefile  **~ ---x-w--wx** nomefile

so@linux~ **chmod 777** nomefile  **~ -rwxrwxrwx** nomefile **massimo permesso**

#### IPCS

**ipcs** mostra nel terminale **Code Messaggi**, **SHM**, **Semafori** in utilizzo

**ipcs – q** mostra solo le Code Messaggi

**ipcs – m** mostra solo le SHM

**ipcs – s** mostra solo i Semafori

**ipcs – q** msqidcancella solo l’msqid in esecuzione

**ipcs – m** shmidcancella solo l’msqid in esecuzione

**ipcs – s** semidcancella solo l’msqid in esecuzione

**ipcs – a** cancella tutto quello che si può cancellare

## MAKEFILE

**cod ..** **files**: main.c

**all:** nomeeseguibile

**nomeeseguibile:** main.c

**gcc** main.c **-o** nomeeseguibile

**clean:**

**rm -rf** nomeeseguibile

**~**

**cod ..** **files**: main.c header.h procedure.c

**all:** nomeeseguibile

**nomeeseguibile:** header.h

**gcc** main.c procedure.c **-o** nomeeseguibile

**clean:**

**rm -rf** nomeeseguibile

**~**

**cod ..** **files**: main.c header.h procedure.c semafori.h semafori.c

**all:** nomeeseguibile

**nomeeseguibile:** header.h semafori.h

**gcc** main.c procedure.c semafori.c **-o** nomeeseguibile

**clean:**

**rm -rf** nomeeseguibile

**~**

MAKEFILE CON OGGETTI , MA FUNZIONA UGUALE A PRIMA , QUESTO USA PRINCIPALMENTE IL PROF

**cod ..** **files**: main.c header.h procedure.c semafori.h semafori.c

**all:** nomeeseguibile

**nomeeseguibile:** procedure.o semafori.o main.o

**gcc** procedure.o semafori.o main.o **-o** nomeeseguibile

**procedure.o:** procedure.c header.h

**gcc** -**c** procedure.c

**semafori.o:** semafori.c semafori.h

**gcc** -**c** semafori.c

**main.o:** main.c

**gcc -c** main.c

**clean:**

**rm -rf** \*.o

**rm -rf** nomeeseguibile

**~**

MAKEFILE CHE CREA 4 ESEGUIBILI , FUNZIONANTI ANCHE SINGOLARMENTE , PER AVVIARE PROGRAMMA COMPLETO ESEGUO “eseguibile”

**cod ..** **files**: main.c header.h procedure.c processo1.c processo2.c processo3.c

**all:** eseguibile processo1 processo2 processo3

**eseguibile:** header.h

**gcc** main.c procedure.c **-o** eseguibile

**processo1:** header.h

**gcc** processo1.c procedure.c **-o** processo1

**processo2:** header.h

**gcc** processo2.c procedure.c **-o** processo2

**processo3:** header.h

**gcc** processo3.c procedure.c **-o** processo3

**clean:**

**rm -rf** eseguibile processo1 processo2 processo3

**~**

## VARIABILI

NOME VARIABILE : **P**

1. **INTERO: INT** P , **FLOAT** P
2. **ARRAY: INT** P [ dim ]
3. **CARATTERE: CHAR** P
4. **STRINGA: CHAR** P [ dim ]
5. **MATRICE: INT** P [dim] [dim]
6. **PUNTATORE: INT\*** P , **CHAR\*** P

## 1.INTERO

#### DICHIARAZIONE

**INT** P; **FLOAT** P;

INIZIALIZZAZIONE

**INT** P = 0; **FLOAT** P = 0;

**INT** P =10; **FLOAT** P =10,2;

INIZIALIZZAZIONE DOPO LA DICHIARAZIONE

P = 0; P = 0;

P =10; P = 10,2;

SCANF

INT 🡪**SCANF ( “%d”, &P )**

FLOAT🡪**SCANF ( “%f”, &P )**

PRINTF

INT 🡪**PRINTF ( “VALUE %d”, P )**

FLOAT🡪 **PRINTF ( “VALUE %f”, P )** o **PRINTF ( “VALUE %1.2f”, P )** stampa 2 ele dopo il punto

## 2.ARRAY

DICHIARAZIONE

**INT** P [dim];

INIZIALIZZAZIONE

**INT** P [2] = 0;

**INT** P [2] = { 5 , 10 };

INIZIALIZZAZIONE DOPO LA DICHIARAZIONE

// ELEMENTO SINGOLO

**P[0]** = 5; // PRIMO ELEMENTO DELL’ARRAY

**P[1]** =10 // SECONDO ELEMENTO

// TUTTO L’ARRAY

for(int i=0; i<dim; i++) **P[i]** = numrandom;

s-> **P[dim]** = 0 // SE SI TROVA IN UNA STRUCT

SCANF

// ELEMENTO SINGOLO

**SCANF ( “%d”, &P[0] )**

**SCANF ( “%d”, &P[1] )**

// TUTTO L’ARRAY

for(int i=0; i<dim; i++) **SCANF ( “%d”, &P[i] )**

PRINTF

// ELEMENTO SINGOLO

**PRINTF ( “VALUE: %d”, P[0] )**

**PRINTF ( “VALUE: %d %d”, P[0], P[1] )**

// TUTTO L’ARRAY

**PRINTF(“VALUE: “)**

for(int i=0; i<dim; i++) **PRINTF ( “%d ”, P[i] )**

3.CHAR

DICHIARAZIONE

**CHAR** P;

INIZIALIZZAZIONE

**CHAR** P = 0;

**CHAR** P = ‘A’;

INIZIALIZZAZIONE DOPO LA DICHIARAZIONE

**P** = 0;

**P** = ‘A’;

s-> **P** = ‘0’ // SE SI TROVA IN UNA STRUCT

SCANF

**SCANF ( “%c”, &P )**

PRINTF

**PRINTF ( “VALUE %c”, P )**

4.STRINGA

DICHIARAZIONE

**CHAR** P [dim];

INIZIALIZZAZIONE

**CHAR** P [4] = 0;

**CHAR** P [ ] = { “CIAO” };

**CHAR** P [ ] = “CIAO”;

**CHAR** P [ ] = { ‘C’ , ‘I’ , ‘A’ , ‘O’ , ‘\0’ };

INIZIALIZZAZIONE DOPO LA DICHIARAZIONE

// ELEMENTO SINGOLO

**P[0]** = ‘A’; // PRIMO ELEMENTO DELLA STRINGA

**P[1]** =’B’ // SECONDO ELEMENTO

// TUTTA LA STRINGA

for(int i=0; i<dim; i++) **P[i]** = carattererandom;

s-> **P[dim]** = ‘0’ // SE SI TROVA IN UNA STRUCT

SCANF

// ELEMENTO SINGOLO

**SCANF ( “%c”, &P )**

// TUTTO L’ARRAY

**SCANF ( “%s”, P )**

PRINTF

// ELEMENTO SINGOLO

**PRINTF ( “VALUE: %c”, P[0] )**

**PRINTF ( “VALUE: %c %c”, P[0], P[1] )**

// TUTTA LA STRINGA

**PRINTF(“VALUE: %s“, P )**

## FUNZIONI STRINGA

**// LIBRERIA “STRING.H”**

**FUNZIONI:**

1. STRCMP
2. STRNCMP
3. STRCPY
4. STRNCPY
5. STRCAT
6. STRNCAT
7. STRLEN
8. STRCHR
9. STRRCHR

**1.STRCMP**  **COMPARA DUE STRINGHE RESTIUISCE:**

**0 ->** SE DUE STRINGHE SONO **“UGUALI”**

**>0 ->** SE LA PRIMA STRINGA È **“MAGGIORE DELLA SECONDA”**

**<0** -> SE LA SECONDA STRINGA È **“MAGGIORE DELLA PRIMA”**

**cod ..**  **x = 0**

char p[5] = ”CIA” , s[5] = ”CIA”

int x;

if(x=**strcmp**(p,s)**==0**) printf(“**p=s**”);

else if (x=**strcmp**(value1,value2)**>0**) printf("**p>s**");

else printf("**p<s**");

// so@linux~ **p=s**

~

**cod ..**  **x > 0**

char p[5 ]= ”CIAO” , s[5] = ”CIA”

int x;

if(x=**strcmp**(p,s)**==0**) printf(“**p=s**”);

else if (x=**strcmp**(value1,value2)**>0**) printf("**p>s**");

else printf("**p<s**");

// so@linux~ **p>s**

~

**cod ..**  **x < 0**

char p[5] = ”CIA” , s[5] = ”CIAO”

int x;

if(x=**strcmp**(p,s)**==0**) printf(“**p=s**”);

else if (x=**strcmp**(value1,value2)**>0**) printf("**p>s**");

else printf("**p<s**");

// so@linux~ **p<s**

~

**2.STRNCMP COMPARA DUE STRINGHE SOLO PER N CARATTERI**

**cod ..**  **comparazione per quattro caratteri**

char p[5] = ”CIAO” , s[10] = ”CIAOBELLO”

int x;

if(x=**strncmp**(p,s,**4**)**==0**) printf(“**p=s**”);

else if (x=**strcmp**(value1,value2)**>0**) printf("**p>s**");

else printf("**p<s**");

// so@linux~ **p=s**  dato che **CIAO** e **CIAO**BELLO sono uguali

~

**3.STRCPY COPIA IL CONTENUTO DI UNA STRINGA**

**cod ..**

char p[5] = ”CIAO” , s[5];

**strcpy** (**s , p**);

**destinazione** , **sorgente** // ora sia p e s conterranno la stessa stringa

**.**

**strcpy** ( **s** , **“CIAO”** ) // ora s conterrà la stringa CIAO

~

**4.STRNCPY COPIA IL CONTENUTO DI UNA STRINGA SOLO PER N CARATTERI**

**cod ..**

char p[10] = ”CIAOBELLO” , s[10];

**strncpy** (**s , p , 5**);

**destinazione** , **sorgente , caratteri** // ora p sarà **CIAOBELLO** mentre s sarà **CIAOB**

~

**5.STRCAT CONCATENA DUE STRINGHE**

**cod ..**

char p[10] = ”CIAO” , s[10] = ”BELLO”;

**strcat** (**p , s**);

**destinazione** , **sorgente** // ora la stringa nuova sarà in p

**.**

printf (” %s “ , **p** )

// so@linux~ **CIAOBELLO**

~

**6.STRNCAT CONCATENA DUE STRINGHE SOLO PER N CARATTERI**

**cod ..**

char p[10] = ”CIAO” , s[10] = ”BELLO”;

**strncat** (**p , s , 2**);

**destinazione** **,** **sorgente , caratteri** // ora la stringa nuova sarà in p

**.**

printf (” %s “ , **p** )

// so@linux~ **CIAOBE** // prende solo due caratteri da s

~

**7.STRLEN**  **RESTITUISCE LA LUNGHEZZA DELLA STRINGA**

**cod ..**  **vari utilizzi**

char p[10] = ”CIAO” , s[10] = ”BELLO”;

printf (” Lunghezza: **%ls** “ , **strlen(p)** )

// so@linux~ **Lunghezza: 4**

if ( **strlen (p)** == 4 ) printf (“ Lunghezza = 4 “)

// so@linux~ **Lunghezza= 4**

if ( **strlen (p)** == **strlen (s)**  ) printf (“ p & s , Stessa lunghezza “)

else printf (“ p & s , Diversa lunghezza “)

// so@linux~ **p & s , Diversa lunghezza**

if ( **strlen (p)** < 10 ) printf (“ Lunghezza < 10 “)

else if (**strlen (p)** > 10 ) printf (“ Lunghezza > 10 “)

else printf (“ Lunghezza = 10 “)

// so@linux~ **Lunghezza < 10**

char p [ 10 ] = “CIAO” , s [ 10 ]

for ( int i = 0 ; i < **strlen (p)** ; i++) s[i] = p[i]

printf (“ Stringa: %s “ , s )\

// so@linux~ **Stringa: CIAO**

~

**8.STRCHR RESTITUISCE UNA STRINGA , PARTENDO DAL VALORE TROVATO**

**cod ..**  **vari scenari**

char p[10] = ”CIAONE”

int s = ‘A’ // carattere

char\* ptr // puntatore

ptr = **strchr ( p , s )**

**puntatore** **,** **stringa(char) , carattere(int)** // ora la stringa nuova sarà in ptr

printf (” Stringa: **%s** “ , **ptr** ) // appena trova il carattere s stampa

// so@linux~ **Stringa: AONE**

// quindi appena trova il carattere s stampa , se non trova il carattere stampa **(null)**

// si può fare anche così

char p[10] =”CIAOCARO” int s=’O’

printf (” Stringa: **%s** “ , **strchr** ( **p** , **s** ) )

// so@linux~ **Stringa: OCARO**

~

**9.STRRCHR RESTITUISCE UNA STRINGA , PARTENDO DALL’ULTIMO VALORE TROVATO**

**cod ..**  **vari scenari**

char p[10] = ”CIAOONE”

int s = ‘O’ // carattere

char\* ptr // puntatore

ptr = **strrchr ( p , s )**

**puntatore** **,** **stringa(char) , carattere(int)** // ora la stringa nuova sarà in ptr

printf (” Stringa: **%s** “ , **ptr** ) // quando trova l’ultimo carattere s stampa

// so@linux~ **Stringa: ONE**

// quindi appena trova l’ultimo carattere s stampa, se non trova il carattere stampa **(null)**

// si può fare anche così

char p[10] =”CIAOCARO” int s=’O’

printf (” Stringa: **%s** “ , **strrchr** ( **p** , **s** ) )

// so@linux~ **Stringa: O**

~

## 6.PUNTATORE

I puntatori sono fondamentalmente delle variabili che rappresentano la locazione di altri elementi nel programma, come variabili, strutture, funzioni, array, file, ecc.…

Adesso precisiamo che ad ogni variabile corrisponde un **nome**, una **locazione di memoria** e l’**indirizzo della locazione di memoria**.

Quindi in C l’operatore **&**, che si utilizza anche nella funzione scanf, ci restituisce l’indirizzo di memoria di una variabile

Un indirizzo può essere assegnato solo a una speciale categoria di variabili denominate puntatori. Quindi i **puntatori sono variabili abilitate a contenere un indirizzo**.

Innanzitutto, diciamo che, una variabile di tipo puntatore viene dichiarata premettendo il **simbolo \*** (operatore unario) al nome della variabile. **L’operatore \*** viene anche denominato **operatore di in-direzione**.

Dunque, questa è la sintassi della dichiarazione dei puntatori in C:

**tipo \*nome\_var;**

### Esempi di tipo dei puntatori in c

**int \*pa**; // puntatore ad una variabile intera

**char \*pc**; //puntatore ad una variabile char

**float \*pf**; //puntatore ad una variabile float

**void \*p**; //puntatore a void

### Inizializzazione puntatore

**int \*pa=NULL;**

**char \*pc=NULL;**

**float \*pf=NULL;**

void \*p;

### Esempio utilizzo puntatori in C

cod.… INIZIALIZZAZIONE ASSEGNAZIONE E STAMPA

int \*pa=**NULL** // inizializzazione, **a=10**

pa=**&a**; // come puntare a una variabile

printf("Indirizzo della variabile: **%p** ", **pa**); // stampa l’indirizzo del val puntato

printf("Valore variabile: **%d** ", **\*pa**); // stampa il val puntato **...10**

printf("Valore variabile: **%d** ", **a**); // equivale alla stampa precedente **...10**

cod.… INIZIALIZZAZIONE ASSEGNAZIONE CAMBIO VALORE E STAMPA

int \*pa=**NULL** // inizializzazione, **a;**

pa=**&a** // come puntare a una variabile

**\*pa**=15// cambiare il valore puntato, equivale a **a=15**

printf("Indirizzo della variabile: **%p** ", **pa**);// stampa l’indirizzo del valore puntato

printf("Valore variabile: **%d** ", **\*pa**)// stampa il valore puntato **...15**

printf("Valore variabile: **%d** ", **a**)// equivale alla stampa precedente **...15**

cod.… **DUE** PUNTATORI

int \*pa=**NULL**,\*pb=**NULL** // inizializzazione, **a**

pa=**&a** // come puntare a una variabile

pb=pa // assegnazione tra due puntatori ora **pb=a**

**\*pb**=15// cambiare il valore puntato, equivale a **a=15,\*pa=15**

printf("Valore variabile: **%d** ", **\*pb**); // stampa il valore puntato **...15**

printf("Valore variabile: **%d** ", **a**); // equivale alla stampa precedente **...15**

cod.… ASSEGNAZIONE TRAPUNTATORI, **CONVERSIONE** DA **CHAR** A **INT**

int \*pa=**NULL**  char \*pb=**NULL** int a;

a=**50**

pa=**&a**

**pb=(char\*)pa; //** cast rende possibile l’assegnazione tra puntatori,ora pb è int & pb=a

printf("Valore variabile: **%d** ", **\*pb**);// stampa il valore puntato **...50**

**\*pa**=15// cambiare il valore puntato, equivale a **a=15,\*pb=15**

printf("Valore variabile: **%d** ", **\*pb**) // stampa il valore puntato **...15**

printf("Valore variabile: **%d** ", **a**) // equivale alla stampa precedente **...15**

cod.… ASSEGNAZIONE TRAPUNTATORI, **CONVERSIONE** DA **INT** A **CHAR**

int \*pa=**NULL** char \*pb=**NULL** char a

a=**’a’**

pb=**&a**

**pa=(int\*)pb //** cast rende possibile l’assegnazione tra puntatori,ora pa è char & pa=a

printf("Valore variabile: **%s** ", **pa**) //stampa il valore puntato **...’a’**

**\*pa**=’d’// cambiare il valore puntato, equivale a **a=’d’,\*pb=’d’**

printf("Valore variabile: **%s** ", **pa**);// stampa il valore puntato **...’d’**

printf("Valore variabile: **%c** ", **a**);// equivale alla stampa precedente ...’d’ **si usa %c**

cod.… ALLOCAZIONE **DINAMICA** E RILASCIO ALLOCAZIONE

struct nome\* s;

**s = ( Struct\* ) malloc ( sizeof ( Struct ) );**

s->valore\_int = 10;

s->valore\_char = ‘c’;

s->valore\_string[0] = ‘c’; s->valore\_string[1] = ‘i’;

s->valore\_array[0] = 1; s->valore\_array[0] = 2;

printf("Struct\_INT: **%d** , **%d:%d**", s->valore\_int , s->valore\_array[0], s->valore\_array[1] );

printf("Struct\_CHAR: **%c , %s**", s->valore\_char , s->valore\_string );

**free(s);**

// so@linux~ **./eseguibile**

**Struct\_INT: 10 , 1:2** **Struct\_CHAR: c , ci**

## LIBRERIE per usarle si usa **#include** (**< >**) o (**“ “**)

**LIBRERIE STANDARD**

#include **<stdio.h>** funzioni per operazioni di input e output

#include **<stdlib.h>** funzioni di conversione numeri in stringhe, generatori casuali e altre utilità

#include **<string.h>**  funzioni di elaborazioni stringhe

#include **<time.h>** funzioni di manipolazione date e valori temporali (random)

#include **<errno.h>**  libreria per usare funzione perror

**LIBRERIE SYSTEM CALL**

#include **<unistd.h>**  libreria per l’accesso alle system call

#include **<sys/types.h>**  libreria per l’accesso alle system call

#include **<sys/ipc.h>**  libreria per l’accesso alle system call

#include **<sys/wait.h>**  libreria per l’accesso alle funzione wait

#include **<sys/shm.h>**  libreria per usare scambio con Shared Memory

#include **<sys/sem.h>**  libreria per usare scambio con semafori

#include **<sys/msg.h>**  libreria per usare scambio con Code di Messaggio

**LIBRERIE PTHREAD**

#include **<pthread.h>**  libreria per l’uso del pthread

**DIRETTIVA CONDIZIONALE DEL COMPILATORE**

Seil file viene richiamato più volte nel corso di un programma, per evitare problemi è possibile utilizzare la direttiva condizionale del compilatore **#ifndef**

esempio:

**#ifndef** LIBRERIA\_H

**#define** LIBRERIA\_H

…. codice

**#endif**

Questo significa segnalare al compilatore che in questo programma, con istruzione:

**#define**  LIBRERIA\_H

Viene definita la costante LIBRERIA\_H;

La direttiva **#ifndef** avverte il preprocessore di controllare se è già è stata definita la costante, se tale costante non è stata ancora definita, allora devono essere compilate tutte le istruzioni comprese tra **#ifndef** e **#endif**, altrimenti quella stessa parte di codice non deve essere presa in considerazione

Principalmente si usa nella libreria principale **.h**

## SWITCH CASE

Funzione che con il relativo tasto può fare comandi. Premendo 1 si fa l’operazione nel **case 1**, 2 si fa l’operazione **case 2** e così via. **Il break** consente dopo l’operazione di uscire. **Il default** sono tutti i tasti che non sono inclusi da 1 e 2. **scelta** è un int, ma può anche essere un char.

// codice

int value1=20, value2=10, scelta;

printf(“ scelta … ” );

scanf( “%d”, scelta )

**switch(** scelta **) {**

**case 1:**

printf(“ SOMMA: %d” , value1+value2 );

**break;**

**case 2:**

printf(“ SOTTRAZIONE: %d” , value1-value2 );

**break;**

**default:**

printf(“ Comando effettuato “);

**}**

// so@linux~ **./eseguibile** premo 2 eseguo sottrazione

**scelta … 2 SOTTRAZIONE: 10 Comando effettuato**

~

## WHILE

Funzione che continua in loop fino a quando non si avvera la funzione tra parentesi.

// codice

int pid;

**while(** pid!=10 **){**

printf(“ %d ” , pid );

pid++;

**};**

// so@linux~ **./eseguibile** fino a quando pid sia uguale a 10 non esce dal while

**0 1 2 3 4 5 6 7 8 9**

~

DO-WHILE

Una variante del while è il **do...while**. La differenza sostanziale rispetto al **while** classico consiste nel fatto che la condizione viene valutata dopo aver eseguito le istruzioni.

Questo garantisce che il blocco di codice verrà **eseguito almeno una volta**, anche se la condizione non la soddisfa, per esempio se in questo caso pid fosse uguale a dieci eseguirebbe comunque ma solo una volta.

// codice

int pid=10;

**while( pid!=10 ){**

printf(“ %d ” , pid );

**};**

// so@linux~ **./eseguibile** non stamperà niente perché pid è già uguale a 10

// codice

int pid=10;

**do{**

printf(“ %d ” , pid );

**}while( pid!=10 );**

// so@linux~ **./eseguibile** entra nel do e stampa il valore di pid

**10**

~

## STRUCT

Le strutture del C sostanzialmente permettono l'aggregazione di più variabili, in modo simile a quella degli array, una struttura può contenere variabili di tipo diverso. Per denotare una struttura si usa la parola chiave **STRUCT** seguita dal nome identificativo della struttura, che è opzionale. Una struct viene definita come segue

// codice1 senza nome

**struct nome** **{**

**tipo1** nome\_variabile1;

**tipo2** nome\_variabile2;

….

**} ;**

**struct nome** nome2;dichiarazione nel main la struct si chiama nome2 ora

nome2.nome\_variabile = 10; assegnazione valore se la variabile è int

printf(“Variabile: %d” , nome2.nome\_variabile ) stampa valore della struct

// codice2 con doppio nome

**struct nome** **{**

**tipo1** nome\_variabile1; **tipo2** nome\_variabile2; ….

**}** nome2**;**

**struct** nome2;dichiarazione nel main la struct si chiama nome2 ora

nome2.nome\_variabile = 10; assegnazione valore se la variabile è int

printf(“Variabile: %d” , nome2.nome\_variabile ) stampa valore della struct

// codice3 con il typedef

**typedef struct {**

**tipo1** nome\_variabile1; **tipo2** nome\_variabile2; ….

**} nome;**

**nome** nome2;dichiarazione nel main la struct come struttura si chiama nome

nome2**.**nome\_variabile = 10; assegnazione valore se la variabile è int

printf(“Variabile: %d” , nome2.nome\_variabile ) stampa valore della struct

// codice4 quello **più utilizzato**, con il typedef e puntatore alla struttura e si usa **->**

**typedef struct {**

**tipo1** nome\_variabile1; **tipo2** nome\_variabile2; ….

**} nome;**

**nome\*** nome2;dichiarazione nel main puntatore alla struttura

nome2 = ( **nome\*** ) **malloc** ( sizeof ( **nome** ) ) allocazione dinamica obbligatoria

nome2**->**variabile\_string[0] = ‘a’; assegnazione valore, char[ ], si usa **->** no il **.**

printf(“Variabile: %s” , nome2->variabile\_string ) stampa valore della struct

**free**( nome2 ) rilascio allocazione, non obbligatoria, ma consigliata

oppure dichiarazione diretta: **struct nome** nome2 = { **10** , **‘c’** , **“stringa”** , **10,15** } sono tipi diversi vanno messi in base a come sono stati dichiarati nella struttura. L’array va messo alla fine si può avere problemi per la virgola tra più argomenti. ordine es. **int , char , char[ ] , int [ ]**

## SYSTEM CALL

Punti d’entrata diretti attraverso cui un processo attivo può ottenere dei servizi dal kernel

Esempi di system call per la gestione dei processi:

* Creazione ed eliminazione: **fork( ) , wait( ) , exit( ) , exec( )**
* Segnalazione del processo: **kill( )**
* Controllo processo: **ptrace( ) , nice( ) , spleep( )**

L’unico modo in cui un processo può essere creato e mediante la chiamata a sistema **fork( )**

### ID DEL PROCESSO

Ogni processo ha un unico identificatore di processo chiamato **PID** (PROCESS IDENTIFIER), numero intero compreso **tra 0 e 30.000** assegnatosi alla creazione

Un processo ottiene il suo **PID**  con la chiamata a sistema:

**int getpid ( )**

Ogni processo ha un processo padre

Un processo ottiene il **PID**  del padre con la chiamata a sistema:

**int gettpid ( )**

// codice

int pid, ppid

pid = **getpid( )**

printf (” Sono il processo con PID: **%d** “ , pid )

ppid = **getppid( )**

printf (” Il mio processo genitore ha PID: **%d** “ , ppid )

oppure: printf (“ Sono il processo con PID: **%d** “ , getpid( ) )

oppure: printf (“ Sono il processo con PID: **%d** “ , getppid( ) )

~

### ID DEI PROCESSI SPECIALI

* Processo **scheduler**: **pid=0**
* Processo **init**: **pid=1**
* Processo **pagedaemon**: **pid=2**

### FUNZIONE SLEEP

Sospende un processo per un certo numero di secondi:

**sleep (int sec)**

### SYSTEM CALL **“FORK”**

La creazione di nuovi processi è gestita mediante la chiamata di sistema **int fork ( )**, che crea una **copia esatta** (**figlio**) del processo originale (**padre**)

// codice

**int** pid oppure **pid\_t** pid

pid = **fork( )**

if (**pid<0**) //chiamata fallita

if (**pid>0**) //codice del padre

if (**pid==0**) //codice del figlio

~

Supponiamo di voler creare **cinque processi**:

// codice

**int** pid oppure **pid\_t** pid

for (int i=0;i<5;i++) {

pid = **fork( )**

**if**(pid<0) { perror (“ Fork errore ”); exit(1); }

**if**(pid==0) { printf(“ Figlio N.%d”,i+1 ); exit(0); }

for(int i=0; i<5; i++) **wait**(NULL)

return 0

~

SYSTEM CALL **“WAIT”**

Consente al **padre** di raccogliere l’eventuale stato di terminazione dei figli, **int wait (int\* stato)**, restituisce l’id del processo figlio che è terminato. Se non esistono figli restituisce **-1**

// codice

**pid\_t** pid , **int** stato

pid = **fork( )**

if (**pid<0**) perror(“error”) exit(-1)

if (**pid==0**) printf(“figlio”) exit(0)

if (**pid>0**) {

pid= **wait** (**&stato**)

**if(WIFEXITED**(stato)) printf(“terminazione volontaria con num **%d**”, **WEXITSTATUS**(stato))

**else if** (**WIFSIGNALED**(stato)) printf(“termi involontaria con num **%d**”, **WTERMSIG**(stato))

}

~

**WIFEXITED** (stato): restituisce vero, se il figlio è terminato volontariamente

**WEXITSTATUS** (stato): restituisce lo stato di terminazione

**WIFSIGNALED** (stato): restituisce vero, se il figlio è terminato involontariamente

**WTERMSIG** (stato): restituisce il numero dell’interruzione SW che ha causato la terminazione

SYSTEM CALL **“EXIT”**

Un processo termina con la chiamata di sistema **exit**

Quando **exit** viene invocata, uno stato di uscita numerico intero viene passato dal processo al kernel. Tale valore è disponibile al processo padre attraverso la chiamata di sistema **wait**

Un processo che termina normalmente restituisce uno stato di uscita pari a **0**

Quindi:

* **=0** , processo figlio terminato **volontariamente**
* **≠0** , processo figlio terminato **involontariamente**

SYSTEM CALL **“EXEC”**

L’unico modo in cui un programma può essere eseguito da UNIX è che il **processo esistente** invochi una chiamata di sistema **exec**

Esistono varie versioni della **exec**:

* Percorso completo dell’eseguibile, **parametri tramite lista:**

**int execl (** char\* path, char\* arg0, … , char\* argn, (char\*) 0 **)**

* Percorso completo dell’eseguibile, **parametri tramite array:**

**int execv (** const char\* path, char\* const argv[ ] **)**

* Nome dell’eseguibile, da cercare nelle cartelle di sistema, **parametri tramite lista:**

**int execlp (** char\* nomefile, char\* arg0, … , char\* argn, (char\*) 0 **)**

* Nome dell’eseguibile, da cercare nelle cartelle di sistema, **parametri tramite array:**

**int execvp (**const char\* path, char\* const argv[ ] **)**

## EXECL , comando CP

// codice, prende dalla linea di comando, il comando che deve effettuare ovvero **cp**, poi il file da copiare ovvero **cp.c** e il file copiato **cp2.c**

Int main ( **int** argc , **char\*** argv [ ] ) {

pid\_t pid = fork ( );

**if** (pid<0) { perror(“Fork Error”); exit(1); }

**if** (pid==0) {

printf( “Processo figlio, con PID: **%d**”,**getpid( )** );

printf( “argv[0]: **%s**”,**argv[0]** );

printf( “argv[1]: **%s**”,**argv[1]** );

printf( “argv[2]: **%s**”,**argv[2]** );

**execl(“/bin/cp” , ”cp” , argv[1] , argv[2] , NULL)**;

}

**if** (pid>0) { printf(“Processo genitore, con PID: **%d**”,**getppid( )** ); wait(**NULL**);

printf(“copia ok…”); exit(0); }

return 0;

}

// so@linux~ **./eseguibile cp.c cp2.c** copia cp.c in cp2.c

**Processo genitore, con PID: 2668 Processo figlio, con PID: 5162 argv[0]: ./eseguibile argv[1]: cp.c argv[2]: cp2.c copia ok…**

~

EXECL , comando MV

// codice, prende dalla linea di comando, il comando che deve effettuare ovvero **mv**, poi il file da rinominare ovvero **mv.c** e il file rinominato **mv2.c**

Int main ( **int** argc , **char\*** argv [ ] ) {

pid\_t pid = fork ( );

**if** (pid<0) { perror(“Fork Error”); exit(1); }

**if** (pid==0) {

printf( “Processo figlio, con PID: **%d**”,**getpid( )** );

printf( “argv[0]: **%s**”,**argv[0]** );

printf( “argv[1]: **%s**”,**argv[1]** );

printf( “argv[2]: **%s**”,**argv[2]** );

**execl(“/bin/mv” , ”mv” , argv[1] , argv[2] , NULL)**;

}

**if** (pid>0) { printf(“Processo genitore, con PID: **%d**”,**getppid( )** ); wait(**NULL**); exit(0); }

return 0;

}

// so@linux~ **./eseguibile mv.c mv2.c** rinomina mv.c in mv2.c

// so@linux~ **./eseguibile mv.c newdirectory/** sposta mv.c nella nuova cartella

// so@linux~ **./eseguibile mv.c ../** sposta mv.c in una directory precedente

// so@linux~ **./eseguibile mv.c ../newdirectory** sposta mv.c in una directory precedente

~

EXECL , comando LS-L

// codice, prende dalla linea di comando, il comando che deve effettuare ovvero **ls-l**, poi lo esegue

Int main ( ) {

pid\_t pid = fork ( );

**if** (pid<0) { perror(“Fork Error”); exit(1); }

**if** (pid==0) {

printf( “Processo figlio, con PID: **%d**”,**getpid( )** );

printf( “Esecuzione del comando ‘ls-l’ ”);

**execl(“/bin/ls” , ”ls” , “-l” , NULL)**;

}

**if** (pid>0) { printf(“Processo genitore, con PID: **%d**”,**getppid( )** ); wait(**NULL**);

printf(“Comando ls-l effettuato”); exit(0); }

return 0;

}

// so@linux~ **./eseguibile** esegue il comando ls-l

**Processo genitore, con PID: 2668 Processo figlio, con PID: 5559** total 24 -rwxrwxr-x 1 so so 17088 Set 27 18:26 eseguibile -rw-rw-r-- 1 so so 534 Set 27 18:26 mainls.c **Comando ls-l effettuato**

~

EXECL , comando ECHO

// codice, prende dalla linea di comando, il comando che deve effettuare ovvero **echo**, poi stampa nel terminale la stringa scelta

**execl(“/bin/echo” , ”echo” , “HelloWord” , NULL)**;

// so@linux~ **./eseguibile** esegue il comando echo

**Processo genitore, con PID: 2668 Processo figlio, con PID: 5559** HelloWord **Comando echo effettuato**

~

EXECL, 3 PROCESSI

// codice, programma che **crea 3 processi** per rendere ogni processo eseguibile si usa **EXECL**, nomi eseguibili ( **p1**, **p2**, **p3** )

Int main( ) {

pid\_t pid;

pid = fork ( );

if(pid==0) { **execl(“./p1” , “./p1” , NULL );** perror(“Exec Error”); exit(1); } else if(pid<0) { perror(“Fork Error”); exit(1); }

pid = fork ( );

if(pid==0) { **execl(“./p2” , “./p2” , NULL );** perror(“Exec Error”); exit(1); } else if(pid<0) { perror(“Fork Error”); exit(1); }

pid = fork ( );

if(pid==0) { **execl(“./p3” , “./p3” , NULL );** perror(“Exec Error”); exit(1); } else if(pid<0) { perror(“Fork Error”); exit(1); }

for( int i=0; i<3; i++ ) wait(NULL);

return 0;

}

~

# INTERPROCESSOR COMMUNICATION

## RISORSE IPC

Ogni risorsa IPC è identificata da un valore univoco nel sistema, denominata chiave ovvero **IPC\_KEY** Una **IPC\_KEY** può essere:

* **Cablata ,** nel codice ( problemi di consistenza )
* **Assente ,** la risorsa è accessibile solo al processo creatore e ai suoi figli:

es: **key\_t** nome\_key **= IPC\_PRIVATE**

* **Generata ,** dal sistema mediante la primitiva:

es: **key\_t** nome\_key **= ftok ( “./eseguibile” , ‘a’ )**

ftok ( char\* path , char id )

restituisce una chiave ottenuta combinando:

* L’inode number e il minor number del file indicato
* Il carattere indicato come secondo argomento

## COMUNICAZIONI TRA PROCESSI

La comunicazione tra processi mediante strutture dati rese disponibili dal kernel sono:

* **MEMORIA CONDIVISA ( SHM :** shared memory segments **)**
* **SEMAFORI ( SEM :** semaphore arrays **)**
* **CODE DI MESSAGGI ( MSG :** queues **)**

## MEMORIA CONDIVISA , SHM

Una memoria condivisa (**SHM**) è una porzione di memoria accessibile da più processi. L’utilizzo della memoria condivisa anche detta SHM avviene mediante le librerie **<sys/ipc.h>** e **<sys/shm.h>** ed è suddivisa così:

* **CREAZIONE** della shm ,
  + shmget
* **COLLEGAMENTO** della shm ,
  + shmat
* **USO** della shm
* **SCOLLEGAMENTO** e **ELIMINAZIONE** della shm , 
  + shmctl

## PRIMITIVE COMUNI

1. Primitiva **GET**
2. Primitiva **MAT**
3. Primitive **CTL**

### 1.PRIMITIVA **GET ,** creazione della SHM

utilizza la IPC\_KEY ed opportuni parametri, per restituire al processo il descrittore della risorsa, se non esiste la risorsa deve essere creata

Restituisce -1 in caso di fallimento

**int** nome\_descrittore = **shmget (** nome\_key , **dimensione** , **flag )**

**key:**

* ottenuto tramite **ftok**
* ottenuto tramite **IPC\_PRIVATE (** equivale a 0 **)** usata per creare una nuova entry priva di chiave

**dimensione:**

* ottenuta tramite **SIZEOF (** nome\_struttura **)**
* ottenuta tramite **NUMERO (** quindi se per esempio voglio due semafori metto **2 )**

**flag:**

indica la modalità di acquisizione della risorsa, in OR **“ | “**

* **IPC\_CREAT,** crea una nuova risorsa se non ne esiste già una con la chiave indicata, se esiste già è ininfluente. Si trova nella libreria <sys/ipc.h>
* **IPC\_EXECL,** utilizzabile in concomitanza con IPC\_CREAT, ritorna errore se la risorsa esiste già , è utile per evitare di inizializzare la risorsa più volte. Si trova anch’essa nella libreria <sys/ipc.h>

**es1**\_semafori\_shm: **int** ds\_sem = **shmget (** chiave\_sem **, 2 , IPC\_CREAT | 0664 )**

**es2**\_struct\_shm: **int** ds\_struct = **shmget (** chiave\_struct , **sizeof (** Struct **) , IPC\_CREAT | 0664 )**

**es3**\_crea un segmento con key 40 e 1kb di dimensione: key\_t chiave = 40; **int** ds\_struct = **shmget (** chiave , **1024 , IPC\_CREAT | 0664 )** se già esiste ri-utilizza il segmento esistente

**es4**\_crea un segmento con key ftok e 1kb di dimensione: key\_t chiave = ftok( “.” , ’k’ ) **int** ds\_shm = **shmget (** chiave , **1024** , **IP\_CREAT | IPC\_EXCL | 0664 )** if( **ds\_shm >= 0** ) // risorsa è stata creata if ( **ds\_shm <0** ) //risorsa esiste già la shmget deve essere rifatta no ipc **int** ds\_shm = **shmget (** chiave , **1024** , **0664 )**

**es5**\_crea un segmento senza una chiave, la risorsa è utilizzabile solo dal padre e i suoi figli: **int** ds\_shm = **shmget (** IPC\_PRIVATE , 1024 , IPC\_CREAT | 0664 )

2.PRIMITIVA **MAT ,** collegamento alla SHM

Collega il segmento di memoria allo spazio di indirizzamento del chiamante

Restituisce -1 in caso di fallimento

**void\*** nome\_funzione = **( void\* ) shmat (** nome\_descrittore **, shmaddr , flag )**

**void\*:**

* se il valore da passare è un **int** , allora **int\*** nome = **(** **int\* ) …..**
* se il valore da passare è un **char** , allora

**char\*** nome = **( char\* ) ……**

* se il valore da passare è una struttura **struct** , e si usa typedef con nome Struct , allora **Struct\*** nome = **(** **Struct ) …..**

**shmaddr:**

* indirizzo dell’area di memoria del processo chiamante al quale collegare il segmento di memoria condivisa.
* se è uguale a NULL, come quasi sempre, un valore viene automaticamente scelto

**int\*** nome = **( int\* ) shmat (** nome\_descrittore , **NULL , ….**

**flag:**

* opzioni di IPC\_RDONLY per collegare il sola lettura
* quasi sempre è uguale a zero

**int\*** nome = **( int\* ) shmat (** nome\_descrittore , **NULL , 0 )**

**es1**\_crea un segmento con key ftok dimensione 100 con IPC\_CREAT e EXCL valore da passare int

key\_t **chiave\_int** = ftok ( “.” , ‘a’ )

int **ds\_shm** = **shmget** ( **chiave\_int** , 100 , IPC\_CREAT | IPC\_EXCL | 0664 )

if ( **ds\_shm** < 0 ) {

**ds\_shm** = **shmget** (**chiave\_int** , 100 , 0664);

**int\*** var = **( int\* )** **shmat ( ds\_shm , NULL , 0 );**

} else {

**int\*** var= **( int\* ) shmat (ds\_shm , NULL , 0 );**

**\***var = valore;

}

printf(“ valore passato con SHM : %d “ , \*var );

**es2**\_crea un segmento con key ftok dimensione il sizeof della struct con IPC\_CREAT valore da passare tutta la struttura con nome Struct:

key\_t chiave\_shm = ftok ( “.” , ‘k’ );

int ds\_shm = **shmget** ( chiave\_shm , sizeof(Struct) , IPC\_CREAT | 0664 );

if ( ds\_shm < 0) { perror(“ Error shmget “ ); exit(1); }

Struct\* s = ( Struct\* ) **shmat** ( ds\_shm , NULL , 0 );

3.PRIMITIVA **CTL ,** scollegamento e eliminazione della SHM

La primitiva permette dato un descrittore come prima indicato di:

* verificare lo stato di una **risorsa**
* cambiare lo stato di una **risorsa**
* rimuovere lo stato di una **risorsa**