**FwriteRead**

#include <stdio.h>

#include <stdlib.h>

#define BUFFER\_DIM 1024

//PROGRAMMA CHE COPIA CONTENUTI DI UN FILE DENTRO UN'ALTRO TRAMITE UN BUFFER

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

FILE \*origine, \*destinazione;

int n;

unsigned char buffer[BUFFER\_DIM];

if(argc < 3){ //CONTROLLO

printf("inserire corretamente i parametri \n");

exit(1);

}

origine = fopen(argv[1], "r"); //SI APRE IL FILE DATO COME ARG. NELLA COMPILAZIONE

if(origine == NULL){ //SE IL FILE NON ESISTE CI DA ERRORE

printf("errore apertura file origine \n");

exit(1);

}

destinazione = fopen(argv[2], "w"); //CREA O APRE FILE DESTINAZIONE

if(destinazione == NULL){ //CONTROLLLO ERRORE

printf("errore apertura file destinazione \n");

exit(1);

}

while (!feof(origine)){ //WHILE FINCHE NON ARRIVE ALLA FINE DEL FILE ORIGINE

n=fread(buffer, 1, BUFFER\_DIM, origine); //LEGGO UN BYTE ALLA VOLTA DA ORIGINE TRAMITE BUFFER

if(n>0){ //SE CI RITORNA VALORE MAGGIORE DI 0 SIGNIFICA CHE STA ANCORA LEGGENDO

fwrite(buffer, 1, n, destinazione); //SCRIVO UN BYTE ALLA VOLTA DA ORIGINE A DESTINAZIONE TRAMITE N

}

}

//CHIUSURA DEI FILE

fclose(origine);

fclose(destinazione);

return 0;

}

**WEXISTATUS**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

int p;

int status;

int pid;

//Macro = funzione

int main(){

p = fork();

if(p != 0){ //padre

printf("io sono il padre P=%d, PID= %d MIO PADRE: %d\n", p, getpid(), getppid());

printf("aspetto la terminazione del mio figlio %d\n", wait(&status)); //restituisce la PID del figlio che ha terminato tramite risultato della wait a quale passo l'indirizzo di status

printf("il codice di terminazione del mio figlio risulta WEXISTATUS(status) = %d \n", WEXITSTATUS(status));

//se PID < 0 il padre non ha piu figli vivi

printf("aspetto la terminazione del mio figlio %d\n", wait(&status)); //restituisce la PID del figlio che ha terminato tramite risultato della wait a quale passo l'indirizzo di status

}

else { //figlio

printf("io sono il figlio P=%d, PID = %d, MIO PADRE: %d\n", p, getpid(), getppid());

pid = getpid();

exit(25);

}

//qui p = pid del figlio

printf("io sono il padre e vedo ancora la PID del mio figlio %d anche se è terminato \n", p);

}

**MEMORIA INDIPENDENTE**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

//i processi vengono gestiti dal kernel

//p vale sempre il PID del processo figlio

/\* i processi non sono sincronizzati: il figlio e padre contano insieme,

il figlio termina il suo conteggio e il padre continua fino ad arrivare alla fine \*/

/\*quanto il padre finisce il conteggio per primo il figlio diventa orfano (zombie)

e viene assegnato a lui, dalla cpu, un padre fitizio \*/

int main()

{

//qui nasce il padre con il main

int p = fork();

int contapadre, contafiglio;

if(p < 0) //errore p = -1

{

printf("il figlo non è nato \n");

exit(1);

}

else if (p > 0) //p del padre = PID del figlio (> 0) , p del figlio = 0

{

printf("io sono il padre. il mio PID = %d, il PID del mio padre = %d, il PID del mio figlio = %d \n", getpid(), getppid(), p);

for(contapadre = 0; contapadre <=50; contapadre++){

printf("conto del padre: %d \n", contapadre);

}

}

else {

printf("io sono il figlio. il mio PID = %d, il PID del mio padre = %d, il PID del mio figlio = %d \n", getpid(), getppid(), p);

for(contafiglio = 0; contafiglio <=55; contafiglio++){

printf("conto del figlio: %d \n", contafiglio);

}

}

return 0;

}

**GENERA FIGLI**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

//PADRE CHE GENERA DUE FIGLI

int p = getpid();

printf("io sono il padre P=%d, MIO PADRE: %d\n", p, getppid());

for(int i = 0; i<3; i++){

if(p!=0){

p=fork();

if(p==0){

printf("io sono il figlio %d P=%d, PID = %d, MIO PADRE: %d\n", i+1, p, getpid(), getppid());

}

}

}

//SE FACCIO UN'ALTRO FORK NASCONO ALTRI 3 FIGLI (1 PER PADRE , 1 PER FIGLIO 1 E 1 PER FIGLIO 2)

return 0;

}