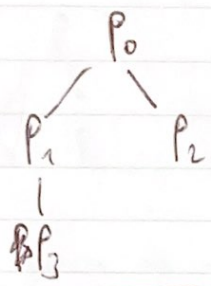
PROCESSO



P_3 INVIA PID A P_1
P_1 RICEVE PID DI P_3 E INVIA A P_2

Consegna

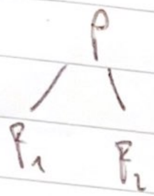
```
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>
#include <string.h>
#include <unistd.h>
```

} librerie

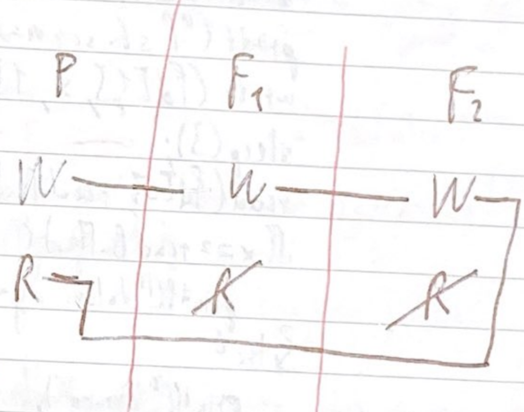
```
int main() {
    pid_t val_1;
    int fd[2];
    char readBuffer[100];
    int P1_status, P2_status, P3_status;
    pipe(fd);
    val_1 = fork();
    if(val_1 == 0) {
        pid_t val_2 = fork();
        if(val_2 == 0) { // P3
            close(fd[0]);
            printf("P3 sta scrivendo... "); int pid_3 = (int)getpid();
            write(fd[1], pid_3, sizeof(pid_3));
        } else { // P1
            printf("P1 legge.. ");
            read(fd[0], readBuffer, sizeof(readBuffer));
            printf("P1 manda... ");
            write(fd[1], readBuffer, sizeof(readBuffer));
        }
    } else {
```

```
pid_t val_32 = fork();
if(val_32 > 0) {  //P2
        close(fd[1]);
        printf("P2 sta leggendo...");
        read(fd[0], readBuffer, sizeof(readBuffer));
} else {  //P0
        printf("Posto leggendo...");
        read(fd[0], readBuffer, sizeof(readBuffer));
        printf("Posto servendo...");
        write(fd[1], readBuffer, sizeof(readBuffer));
    }
}
wait(&P1_status);
wait(&P2_status);
wait(&P3_status);
}
```

PROCESSO



- P genera un intero casuale x e lo invia al processo F1
- F1 riceve x da P e lo inoltra a F2
- F2 rimanda x a P
- P verifica che il valore ricevuto sia uguale a quello inviato e termina




```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
~~#include <string.h>~~

int main() {
    pid_t val_1;
    int F1status, F2status;
    int fd[2];
    char readBuffer[100];
    pipe(fd);
    val_1=fork();

    if(val_1==0){ //F1
        ~~printf("F1 legge");~~   close(fd[0]);
        ~~read(fd[0], readBuffer, sizeof(readBuffer));~~
        printf("F1 sta scrivendo");
        write(fd[1], ~~readBuffer~~ NULL, ~~sizeof(readBuffer)~~ NULL); exit(0);
    }else {
        pid_t val_2=fork();
```

Note a margine:

- pid_t val_1 → anche pid_1
- int F1status, F2status → child status
- int fd[2] → 2 stati (posizione 0 legge, 1 scrive, 2 errore)
- char readBuffer[100] → obbligatorio quando c'è pipe
- val_1=fork() → fork va fatto sempre dopo pipe
- close(fd[0]) → chiudo read per maneggiare FD1 — va sotto write
- NULL → metto NULL perché x devo inoltrarlo, non devo sovrascriverlo o fare doppioni nel buffer

```
if(val_2==0){  // P2
    close(fd[0]);
    printf("P2 sta scrivendo...");
    write(fd[1], NULL, 0);
    exit(0);

}else{      //P
    int x = (int) (Math.random * 100);
    ~~close(fd[0])~~;
    printf("P sta scrivendo...");
    write(fd[1], x, 1);
    sleep(3);  —> 3 secondi (abbastanza arbitrario)
    read(fd[0], readBuffer, sizeof(readBuffer));
    if(x == readBuffer){   —> confronto primo el su readbuffer
        printf("Valori uguali");
    }else{
        printf("Errore");
    }
  }
}
wait(&P1_status);
wait(&P2_status);
}
```

basta che sia "grande" (note on 100)

& (note on wait)

SSO

PADRE MANDA AL FIGLIO UN SEGNALE DI TERMINAZIONE, IL FIGLIO LO IGNORA

P_0
|
P_1

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>      #include <wait.h>

int main (){
    int child_status;          → serve per aspettare la terminazione del figlio

    pid_t pid = fork();        → ne faccio solo uno in quanto c'è solo un figlio

    if (pid==0){               //child process
        printf("Child process:\n");
        signal (SIGTERM, SIG_IGN);   → per ignorare il segnale
                                     (ignora l'azione (a meno che non sia SIGKILL))
        printf("C: child ID is %d\n", (int)getpid());
    }else {
        printf("Parent process:\n");
        printf("P: Parent ID is %d\n", (int)getpid());   → per avere il PID del padre
        printf("C: child ID is %d\n", (int) pid());      → PID del figlio
        int kill( pid, SIGTERM);
    }                          → tramite kill invio un segnale

    wait (&child_status);
    return 0;
}
```

PROTOTIPO FUNZIONE pthread_create

```
int pthread_create (pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine) (void *), void *arg);
```

Il primo parametro è l'ID univoco del thread, il secondo è un puntatore ad dei parametri di default che devono essere gestiti dal thread (tipicamente settiamo a NULL), il terzo è un puntatore alla thread function e all'ultimo troviamo un puntatore all'insieme di argomenti di input che in particolare specifica su quali dati dovrà lavorare il thread.

PROTOTIPO FUNZIONE pthread_join

```
int pthread_join (pthread_t *thread, (void**) return_value);
```

Il primo parametro rappresenta l'ID del thread che vogliamo aspettare, il secondo è un puntatore anno variabile void che punta al valore di ritorno del thread (può essere anche NULL).