**Cose che dovresti sapere per evitare di rimanere bloccato all’esame per ore se non avessi questo magico riassuntino a sbloccarti**

* **quando ci sono le pipe**

typedef int pipe\_t[2];

pipe\_t piped;

**il commento è:**

pipe utilizzata per la comunicazione in pipeline: (specificare come avviene, in molti casi è sempre questo) tutti i figli eccetto il primo leggono da piped[i-1][0] e scrivono su piped[i][1]

**nel figlio la chiusura è questa:**

for (int j = 0; j < n; j++)

{

if(i==0 || j!=i-1)

{

close(piped[j][0]);

}

if( i!=j)

{

close(piped[j][1]);

}

}

quando il figlio ha finito di fare ciò che viene richiesto dal testo, dobbiamo fare delle azioni diverse a seconda se siamo il primo figlio o gli altri

Nel caso in cui i figli debbano passare avanti **una** struct dati:

struct dati {

int c1;

int c2;

};

if(i==0)

{

il figlio se deve passare una struct la inizializza e la passa al prossimo figli;

nb: il figlio per i==0 non deve leggere ma scrive soltanto

d.c1=i;

d.c2=dim;

}

else

{ /\*quando si usano le pipe si deve **sempre** controllare la lettura/scrittura da pipe, i figli con i!=0 leggono sempre da piped[i-1][0]\*/

if((nr=**read(piped[i-1][0],&d,sizeof(struct dati)) != (sizeof(struct dati))**)

{ printf("Errore nella lettura dalla pipe\n");

exit(-3); }

d.c1=i;

d.c2=dim;

}

//entrambi scrivono sulla pipe in posizione i

if((nw=**write(piped[i][1],&d,sizeof(struct dati)))!=(sizeof(struct dati))**)

{

printf("Errore nella scrittura sulla pipe\n");

exit(-4);

}

nel caso in cui i figli si debbano **passare un array di struct:**

typedef int pipe\_t[2];

pipe\_t piped;

struct dati \*d //puntatore a struct

if(i==0)

{

d=malloc(i+1)\*sizeof(struct dati) );

//controllo sempre l’allocazione della struct

if(d==NULL)

{

printf("Errore nella allocazione dell'array di struct del figlio\n");

exit(-2);

}

d[i].c1=i;

d[i].c2=dim;

}

else

{

//i figli con i!=0 allocano una struct in più

d=malloc((i+1)\*sizeof(struct dati));

if(d==NULL)

{

printf("Errore nella allocazione dell'array di struct del figlio\n");

exit(-2);

}

**(occhio nelle read/write non ci va &d ma solo d, perchè è già un puntatore)**

if((nr=**read(piped[i-1][0],d,(i\*sizeof(struct dati))))!= (i\* sizeof(struct dati))**)

…….

assegnamento dei valori nella struct i-esima

}

if((nw=**write(piped[i][1],d,(i+1)\*sizeof(struct dati)))!=((i+1)\* sizeof(struct dati))**)

{

printf("Errore nella scrittura sulla pipe\n");

exit(-4);

}

chiusure nel padre:

for (i = 0; i < n; i++)

{

close(piped[i][1]);

if(i!=n-1)

{

close(piped[i][0]);

}

}

//recupero della sola struct:

if((nr=**read(piped[n-1][0],&d,sizeof(struct dati)))!= (sizeof(struct dati))**)

{

printf("Errore nella lettura dalla pipe\n");

exit(7);

}

//recupero dell array di struct:

if((nr=**read(piped[n-1][0],d,n\*sizeof(struct dati)))!= (n\* sizeof(struct dati))**)

{

printf("Errore nella lettura dalla pipe\n");

exit(7);

}

* **quando ci sono i ring**

//INDICI RING [(q + 1) % Q] --> sia per chiusura (vedi sotto),

//sia per scrittura per sincronizzazione alla pipe del figlio successivo write(pipe[(q + 1) % Q])

//CHIUSURA PIPE FIGLI

for(j = 0; j < N; ++j){

if(j != q){

close(pipefp[j][0]);

}

if(j != (q + 1) % Q)

{

close(pipefp[q][1]);

}

}

//IL PADRE PARTE A CHIUDERE DA q = 1 fino a Q

for(q = 1; q < Q; ++q){

close(pipefp[q][0]);

close(pipefp[q][1]);

}

//INNESCO PER FAR PARTIRE IL RING

//ok è un carattere che serve per dare l’ok tra tutti i processi

/\* il padre invia la sincronizzazione al primo figlio \*/

if((nw = write(pipefp[0][1], &ok, sizeof(char))) != sizeof(char)){

printf("Errore nella sincronizzazione da padre al primo figlio\n");

exit(4);

}

//PIPE RING DEL PADRE --> non chiude pipe[0][0]

//fa cose... poi CHIUDE

close(pipefp[0][1]);