CODICE UTILE PER ESAME

Scaletta di creazione file:

1. Inclusione dei file
2. Definizione delle variabili.
3. Controllo sul numero di parametri.
4. Creazione di un file temporaneo se richiesto
5. Allocare l'array delle n pipe.
6. Creazione delle n pipe.
7. Generazione dei figli
    1. Controllare se fork()<0.
    2. Entrare nel codice del figlio (if(PID==0))
        1. Chiusura delle pipe non utilizzate.
        2. Apertura del file se necessario.
        3. Se ci sono I nipoti ripetere procedura per I figli.
    3. Comunicazione dal figlio al padre (scrittura sulla pipe).
    4. Far ritornare il valore di ritorno.
8. Chiusura dei lati della pipe non usati dal padre.
9. Recupero delle informazioni da parte del padre(lettura dalla pipe).
10. Wait del padre
11. return 0;

01) File da includere	#define _POSIX_SOURCE #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <fcntl.h> #include <string.h> #include <sys/wait.h> #include <sys/stat.h> #include <sys/types.h> #include <ctype.h> #include <time.h> #define PERM 0644 typedef int pipe_t[2];
02) Variabili comuni	Int N, PID, I, j, fdw, PIDFiglio, status, ritorno; char c, Cx;

03) Controllo sul numero di parametri	if (argc!=N){
	puts("Error: number of parameters");
	exit(1);
	}

04) Creazione del file temporaneo	if ((fdout=open("/tmp/Filetemp", O_CREAT|
	O_WRONLY|O_TRUNC, 0644)) < 0){
	printf("Errore nella creazione del file %s\n",
	"/tmp/creato");
	exit(3);}

05) Allocazione array n pipe	piped = (pipe_t *) malloc (N*sizeof(pipe_t)); if (piped == NULL)

	{
	printf("Errore nella allocazione della memoria\
	n");
	exit(2);
	}

06) Creazione delle n pipe	for (i=0; i < N; i++)
	{
	if(pipe(piped[i]) < 0)
	{
	printf("Errore nella creazione della pipe\n");
	exit(3);
	}
	}

07.0.0) Creazione figlio con controllo (N figli)
07.1.1) Con chiusura delle pipe
	for (i = 0; i < N; i++){
	if((PID=fork())<0){
	puts("Error: son creation.");
	exit(4);
	}
	else if(PID==0){
	printf("Sono il figlio %d e sono associato al file
	%s\n", getpid(), argv[q+1]);
	for(int i=0;i<M;i++){
	/*close(piped[i][0]);
	if(q!=i)
	close(piped[i][1]);*/
	}
	//Codice figlio
	exit(ValoreDiRitorno);
	}
	}

07.2.2) Apertura file	if((fd=open(NomeFile,O_RDONLY))<0){
	puts("Error: open();");
	exit(2);
	}

07.3) Scrittura sulla pipe	write(piped[i][1], &valore, sizeof(valore));
08) Chiusura dei lati della pipe non usati dal padre.	for (i=0; i < N; i++) close(piped[i][1]);

09) Lettura dalla pipe	for (i=0; i < N; i++){
	read(piped[i][0], &valore, sizeof(valore));
	}

10) Ricezione dati dal figlio al padre (N figli)	for(int i=0;i<N;i++){
	pidFiglio=wait(&status);
	if(pidFiglio<0){
	puts("Errore nella wait");
	exit(5);
	}
	if((status & 0xff)!=0)
	printf("Figlio con pid %d terminato in modo

	anomalo\n",pidFiglio);
	else{
	ritorno=(int)((status >> 8)&0xFF);
	printf("Figlio PID=%d ha ritornato %d(255=Error)\n",pidFiglio,ritorno);
	}
	}

Controllo se i parametri sono singoli caratteri	for (int i=0; i < N; i++)
	if (strlen(argv[i+2]) != 1){
	printf("Errore nella stringa %s che non e' un singolo carattere\n", argv[i+2]);
	exit(1);
	}

Creazione figlio con controllo (1 figlio)	if((PID=fork())<0){
	puts("Error: fork();");
	exit(3);
	}
	else if(PID==0){
	//Codice figlio
	exit(ValoreDiRitorno);
	}

Ricezione dati dal figlio al padre (1 figlio)	PIDFiglio=wait(&status);
	if(PIDFiglio<0){
	puts("Error: wait();");
	exit(4);
	}
	if((status & 0xFF)!=0){
	printf("Figlio PID:%d terminato in modo anomalo.\n",PIDFiglio);
	}
	else{
	ritorno=(int)((status >> 8)& 0xFF);
	printf("Figlio PID=%d ha ritornato %d\n (se 255 errore)", pidFiglio, ritorno);
	}

Utilizzo numero random	int myrandom(int n){
	return (rand() % n);
	}
	int main(){
	Int Nrand;
	//codice
	srand(time(NULL));
	Nrand=myrandom(100);
	//codice
	}

Valori di ritorno delle funzioni utilizzate:

- ✗ Write: Struttura → write(file descriptor, carattere da scrivere, dimensione del carattere);
  - Ritorna -1 se c’è stato un errore in scrittura o gli N byte scritti.

- ✗ Read: Struttura → read(file descriptor, carattere in cui inserire il valore letto, dimensione del valore da leggere);
  - Ritorna -1 se c'è stato un errore in lettura o gli N byte letti.
- ✗ Pipe: Struttura → pipe(vettore[indice]);
  - Ritorna 0 se ha successo, -1 se fallisce.
- ✗ Fork: Struttura → fork();
  - In caso di successo ritorna il PID creato del figlio al padre e 0 al figlio, se fallisce ritorna -1 al padre.
- ✗ Open: Struttura → open(Nome file, modalità);
  - Ritorna il file descriptor o -1 in caso di fallimento nell'apertura.

In caso di schema di sincronizzazione a ring utilizzare il seguente codice:

- Chiusura delle pipe nel figlio:
  - for (j=0;j<Q;j++){
    if (j!=q)
    close (pipes[j][0]);
    if (j != (q+1)%Q)
    close (pipes[j][1]);
    }
- Aspettare l'ok dal figlio precedente:
  - nr=read(pipes[q][0],&ok,sizeof(char));
- Controllare sia andato a buon fine:
  - if (nr != sizeof(char)){
    printf("Figlio %d ha letto un numero di byte sbagliati %d\n", q, nr);
    exit(-1);
    }
  -
- Dare l'ok al figlio successivo:
  - nw=write(pipes[(q+1)%Q][1],&ok,sizeof(char));
- Controllare sia andato a buon fine:
  - if (nw != sizeof(char)){
    printf("Figlio %d ha scritto un numero di byte sbagliati %d\n", q, nw);
    exit(-1);
    }
- Chiusura delle pipe dal lato del padre:
  - for(q=1;q<Q;q++){
    close (pipes[q][0]);
    close (pipes[q][1]);
    }
- Invio del primo segnale al primo figlio:
  - nw=write(pipes[0][1],&ok,sizeof(char));
- Controllo sul segnale:
  - if (nw != sizeof(char)){
    printf("Padre ha scritto un numero di byte sbagliati %d\n", nw);
    exit(5);
    }
- Chiusura dell'ultima pipe aperta:
  - close(pipes[0][1]);