# DW 3

## Reminder

**Frequent Mistakes With Pipes**

- **Closing All Read Descriptors Then Trying To Write On Pipe**: Kernel doesn't find where to write the data, causing a SIGPIPE signal or an EPIPE error.
- **Not Closing All Write Descriptors Then Trying To Read From That Pipe**: Kernel waits for the pipe to be written to, as it doesn't see an EOF (End Of File) signal.

**Solution**

- **Leave at least one read end of the pipe open before writing to it**, so the kernel knows where to write the data.
- **Always close all write ends of a pipe before reading from it**, so the kernel knows that the pipe has reached its EOF.

## Exercice 1

The goal of this exercise:

- **child_1**: Accepts input of n characters. Write only alphabetic characters to the pipe, converts lowercase letters to uppercase. Stops when the user inputs '0'.
- **child_2**: Prints the characters written to the pipe by **child_1**.
- **parent**: Creates the child processes and waits for them to finish.

The includes needed

- stdio.h : to printf and get user input
- stdlib.h : to use exit to terminate whole programe and to terminate child process
- unistd.h :to use fork and pipe primitive
- ctype.h : to use toupper function
- sys/wait.h : to use wait primitive (parent wait for child to terminate)

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<ctype.h>
#include<sys/wait.h>


int descriptor[2];

void child_1();

void child_2();


int main () {

 if (pipe(descriptor) == -1){
     printf("Error Pipe Creation Failed");
     exit(-1);
 }

 pid_t pid1;

 pid1 = fork();

 if(pid1 == -1){
     printf("Error Child 1 Processus Creation Failed");
     exit(-1);
  }

 else if(pid1==0){
 child_1();
 }

 else {

wait(NULL);

close(descriptor[1]);
 pid_t pid2;

 pid2 = fork();

 if(pid2 == -1){
   printf("Error Child 2 Processus Creation Failed");
   exit(-1);
 }

 else if (pid2==0){
  child_2();
 }

 else {
  wait(NULL);
  close(descriptor[0]);
  printf("\nEND EX1\n");
 }

 }

 return 0;
}
```

child_1 void function :

```
void child_1() {

 close(descriptor[0]);

 char car;

 printf("Input Char In CHILD 1\n");

 while ((car = getchar()) != '0') {
 if(car>='a' && car<='z'){
  car = toupper(car);
  write(descriptor[1],&car,1);
 }

 else if (car>='A' && car <='Z'){
  write(descriptor[1],&car,1);
 }

 }

 close(descriptor[1]);
 exit(0);

}
```

child_2 void function :

```
void child_2() {

  printf("\nPrinting Inputed Chars From CHILD1 in CHILD2 : ");

  char car;

  while(read(descriptor[0],&car,1) > 0){
   printf("%c",car);
  }

  printf("\n");

  close(descriptor[0]);
  exit(0);
}
```

```
rabah@UbuntuTex:~/Desktop/Documentations/University/3rd_ING_Software/1st_Semester/Oper
ating_System/DW/Solution/Questions/EX1$ gcc ex1.c -o ex1
rabah@UbuntuTex:~/Desktop/Documentations/University/3rd_ING_Software/1st_Semester/Oper
ating_System/DW/Solution/Questions/EX1$ ./ex1
Input Char In CHILD 1
a
v
-
+
2
A
B
B
0

Printing Inputed Chars From CHILD1 in CHILD2 : AVAB

END EX1
```