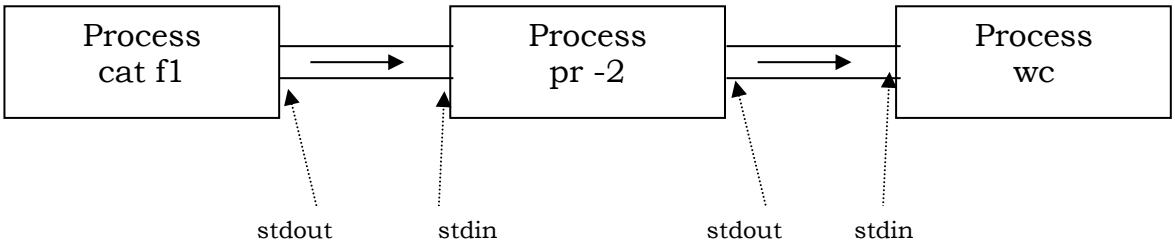
Exercise 1:

Write a C program that executes the shell command

cat filename1 | pr -2 | wc

where *cat* allows to display the contents of a file given in parameter; *pr -2* prints the result on 2 columns and *wc* counts the number of lines, words and characters printed.



Exercise 2:

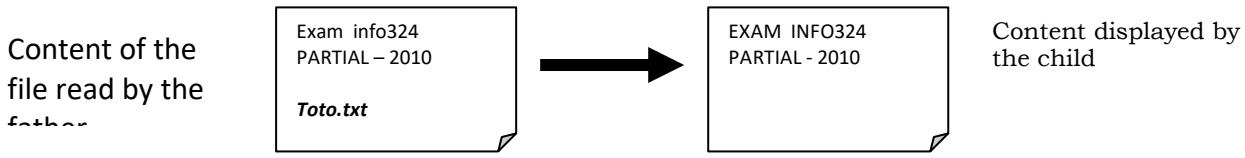
Consider the following program where a process father creates a process child. The process father writes on the screen the characters 'A' and 'B'. The process child writes the characters 'C' and 'D'. We suppose that the process creation is done with success.

```
#include<stdio.h>
#include<unistd.h>
void main()
{
      if(fork()){     /* father */
            printf("A");
            printf("B");
      }
      else{           /* child */
            printf("C");
            printf("D");
      }
}
```

1. Which is (are) the result(s) printed by executing this program?
2. In what follows**, the process father always must write "A" and "B", the child process always must write "C" and "D".**
    a. Change, the least possible, the program for that the only result be "CDAB".
    b. Change, the least possible, the program for that the only result be "ACBD"

Exercise 3:

The primitive system (**cat**) displays the contents of a file whose name is passed as parameter. Write a program where the parent process sends the contents of a file (**toto.txt**) to its child using the original cat. The child must display on the screen the contents of the file in capital letters.



Exercise 4:

1. Write a program that creates N child processes. The processes follow the following rules:
   - The child process with even PID execute the function FP()
   - The child process with odd PID execute the function FI().
   - The parent must wait the termination of all children before exit.

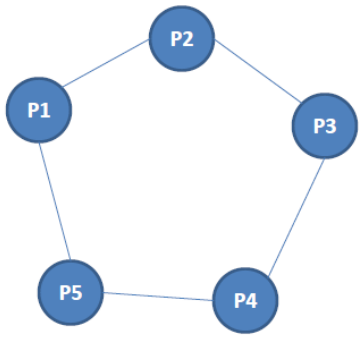
   You are not requested to write the functions FP() et FI().

2. Now, suppose that all child processes shared a file and the child (odd and even) use the functions FP() and FI() for accessing the file. We assume that in a given instant, one process (even or odd) can write in the file (the other processes, wanting to write to the file, must be blocked until the end of the current writing process). Using the pipes of communications, add few lines to the code of part (1) in order to achieve the above described synchronization.

3. In this part, we suppose that the even children are writers and the odd children are readers. If a writer process is writing to a file using FP(), we must prevent all other processes to access the file. In contrast, several readers can read simultaneously the file using FI() without any writer. Using pipes perform the described synchronization.

Exercise 5:

Assume now that the 5 child processes of Part A, are placed as presented in the graph, indicating that each process has two neighbors processes (e.g., neighbors of P2 are P1 and P3). We would like to apply synchronization on five processes such as: if the process Pi is executing its function Gi (), neither of these two neighboring process can execute its function. For example, if P2 is currently executing G2 (), then P1 and P3 should be prevented from executing their functions G1 () and G3 (), respectively.



Using communication pipes, write the code needed for this synchronization.