Lab4: Advanced IO

Subject

- Running a file : `gcc -o io io.c`
- Then use `./execName` to display results.

1- File Descriptors

To create a file named "text1" : `touch text1`.
When i run `cat text1 > text2` the content of the file "text1" is copied into a new file "text2".
I writed a program that :

1. open “text1” in read mode (man 2 open)
2. open “text2” in write mode
3. redirect standard output to text2 (man 2 dup2)
4. create a loop that reads text1 and writes it on standard output

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  unsigned int end = 0;
  while (end != 1) {
    //We open text1 in read mode
    int ftext1 = open("text1", O_APPEND);
    //We open text2 in write mode
    int ftext2 = open("text2", O_WRONLY);
    //We redirect the standart output from
    int copy = dup2(ftext1,ftext2);
  }
}
```

2- Pipes

`ps aux` can be used to see all the processes running on each user.
`more` is used to to display a large text in different screen.
I suppose that when we call `ps aux` the number of processes to print can be too large for a single windows so it use the function `more` to split it between multiple windows.
A pipe is a connection between two processes (ex: standard output of one process become the input of antoher process).

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

int main(int argc, char *argv[]) {
  int pipefd[2];
  pid_t cpid;
  if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
  }
  cpid = fork();
  if (cpid == -1) {
    perror("fork");
```

```
      exit(EXIT_FAILURE);
    }
    if (cpid == 0) {    /* Child reads from pipe */
      close(pipefd[1]); //close pipe write end
      dup2(STDIN_FILENO,pipefd[0]); //redirect stdin into the pipe
      system("more"); //execute more
      close(pipefd[0]);
      exit(EXIT_SUCCESS);
    } else { //parent
      /* Parent writes argv[1] to pipe */
      close(pipefd[0]); //close pipe read end
      dup2(STDIN_FILENO,pipefd[0]); //redirect stdin into the pipe
      system("ps aux");
      close(pipefd[1]);
      /* Reader will see EOF */
      wait(NULL);
      /* Wait for child */
      exit(EXIT_SUCCESS);
    }
  }
```

The output of this program is :

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0 167956  7604 ?        Ss   oct.16   0:30 /sbin/init s
root           2  0.0  0.0      0     0 ?        S    oct.16   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   oct.16   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   oct.16   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   oct.16   0:00 [kworker/0:0
root           9  0.0  0.0      0     0 ?        I<   oct.16   0:00 [mm_percpu_w
...
```

3. Non-Blocking Calls

This program wait for 10 input from the user and display the number of character
Code with annotations :

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <fcntl.h>


int main() {
  int i;
  char buf[100];
  // ouvrir un le stdin en lecture non bloquante
  //fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK); //Set the stdin status flag to O_NONBLOCK (it can't be stoped)
  for (i = 0; i < 10; i++) {
    int nb;
    nb = read(STDIN_FILENO, buf, 100); //Read the user input
    printf("nwrites = %d\terror = %d\n", nb, errno); //print the number of character and errors
  }
}
```

Output of this program

```
./nbcall
vkjdb
nwrites = 6    error = 0
gnfdn
nwrites = 6    error = 0
fsfb
```

```
nwrites = 5    error = 0
scqv
nwrites = 5    error = 0
vdbfh
nwrites = 6    error = 0
svdvsd
nwrites = 7    error = 0
vsdvsdb
nwrites = 8    error = 0
dvsdbsd
nwrites = 8    error = 0
vdsvdv
nwrites = 7    error = 0
vdsvds
nwrites = 7    error = 0
```

When i uncomment the line with `fcntl` the program doesn't wait for input and print 10 errors. This is due to the `fcntl` that put the `STDIN_FILENO` (input) into a `0_NONBLOCK` (can't be put in wait queue).

Output when uncommented :

```
./nbcall
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
nwrites = -1    error = 11
```