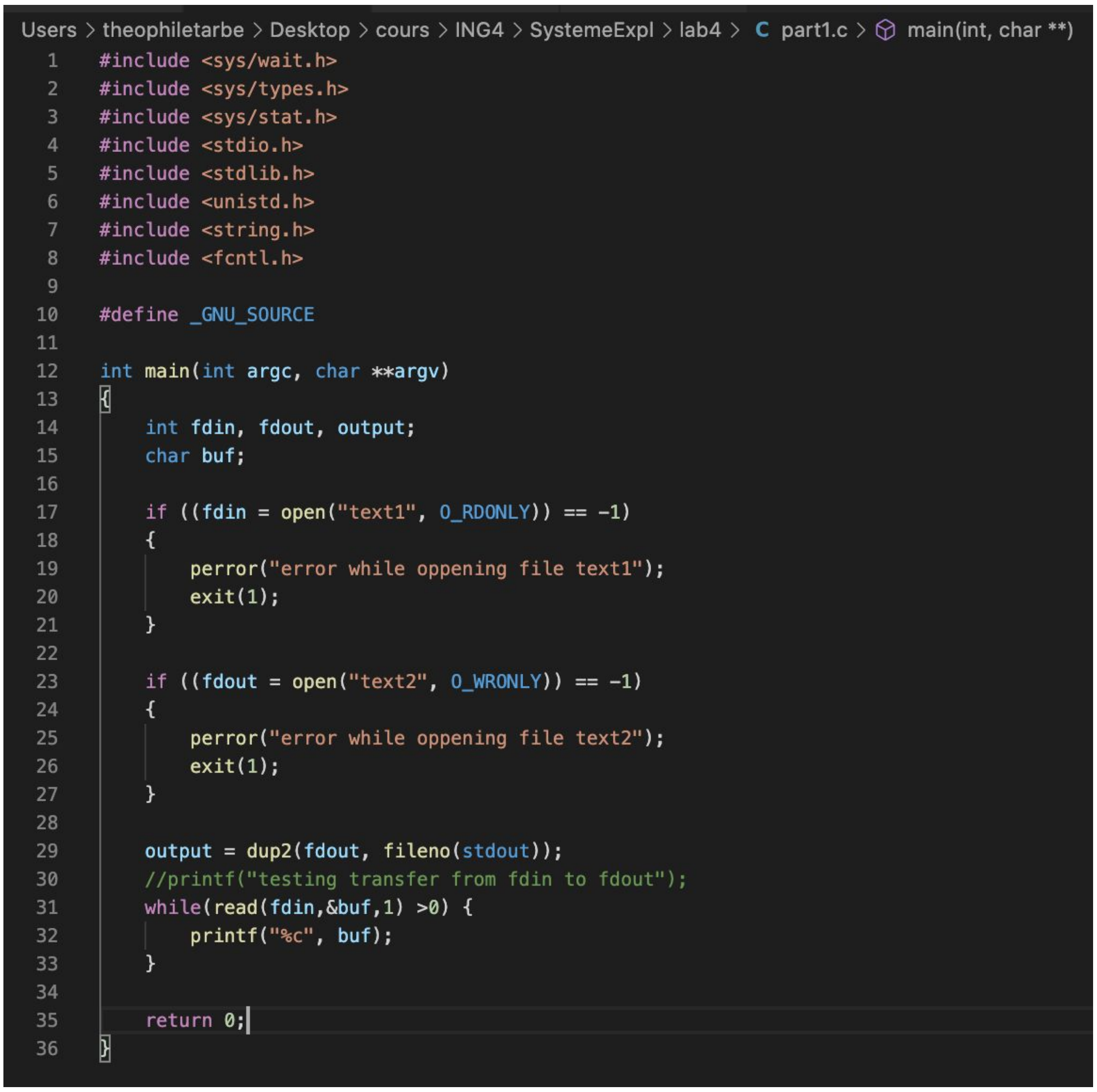
TARBE Théophile
QUIDET Victor
ING4 SI Gr01 (Inter)

LAB 4 : Advanced IO

1. File Descriptors

What happens when you run “cat text1 > text2” ?

`cat` is a standard Unix command to concatenate files and display their contents on the standard output. If we run cat `text1 > text2` in the terminal, it will create the file `text2` if it does not exist already.

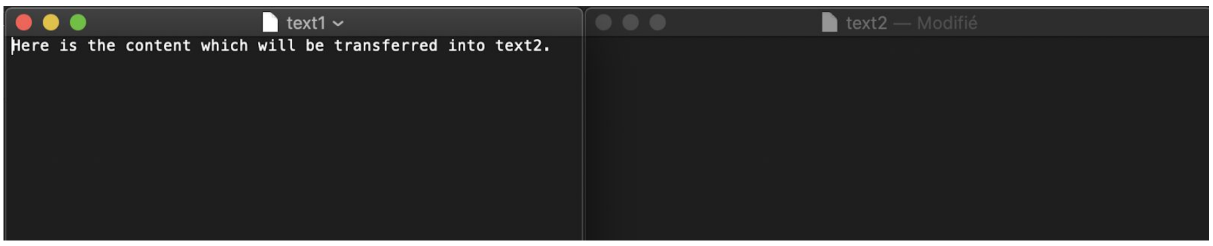


Users > theophiletarbe > Desktop > cours > ING4 > SystemeExpl > lab4 > part1.c > main(int, char **)

```c
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>

#define _GNU_SOURCE

int main(int argc, char **argv)
{
    int fdin, fdout, output;
    char buf;

    if ((fdin = open("text1", O_RDONLY)) == -1)
    {
        perror("error while oppening file text1");
        exit(1);
    }

    if ((fdout = open("text2", O_WRONLY)) == -1)
    {
        perror("error while oppening file text2");
        exit(1);
    }

    output = dup2(fdout, fileno(stdout));
    //printf("testing transfer from fdin to fdout");
    while(read(fdin,&buf,1) >0) {
        printf("%c", buf);
    }

    return 0;
}
```

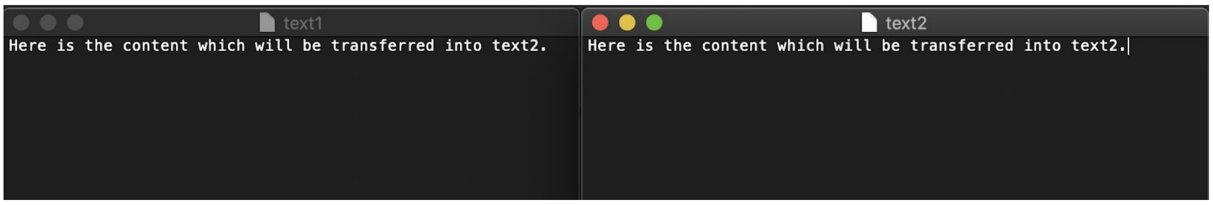
In this code, we open the 2 files `text1` & `text2`, the first one in read mode, the second one in write mode. Then, `dup2()` redirects the standard output of the file `text1` to second file `text2` which is in write mode.

Before compilation :



```
Here is the content which will be transferred into text2.
```

After compilation :



```
Here is the content which will be transferred into text2.
```

```
Here is the content which will be transferred into text2.
```

2. Pipes

1. What kind of interaction is there between these two functions (ps and more) ?

`1. ps`

It displays the status of current processes on the terminal. There are also extended, GNU-style options like :

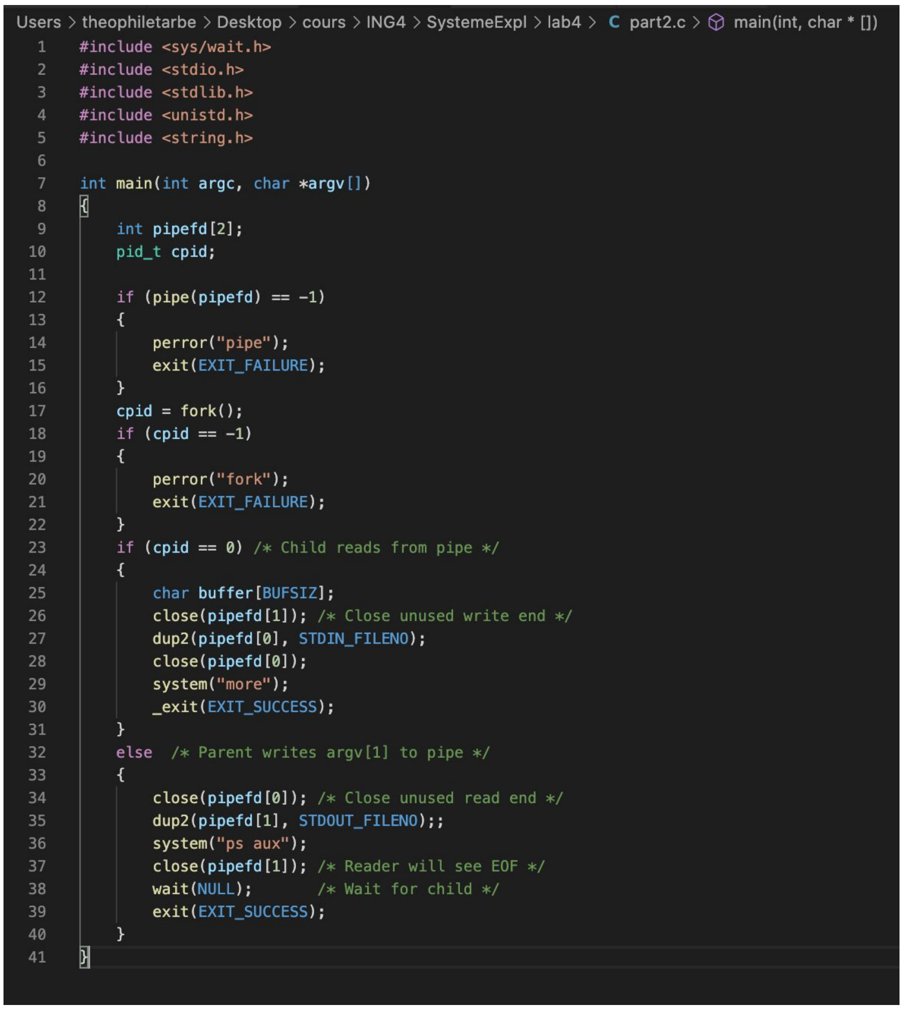
- a : also presents the processes of other users.
- u : presents the user's name and launch time.
- x : displays processes that do not have a control terminal.

`2. more`

It is a standard Unix command used to view (but not modify) the contents of a text file, page by page.

`3. ps aux | more`

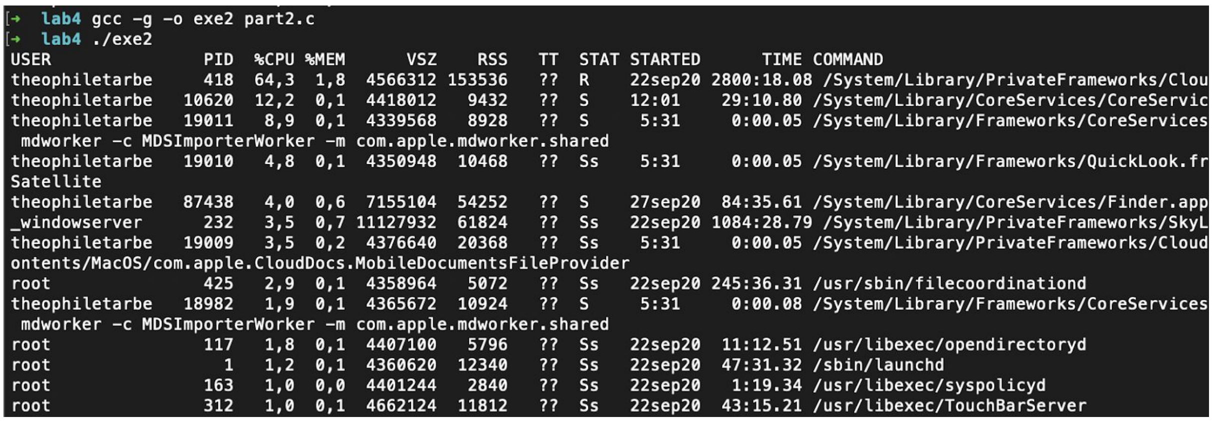
Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command. The `|` creates the pipe.



Users > theophiletarbe > Desktop > cours > ING4 > SystemeExpl > lab4 > C part2.c > main(int, char * [])

```c
#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 */
    {
        char buffer[BUFSIZ];
        close(pipefd[1]); /* Close unused write end */
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[0]);
        system("more");
        _exit(EXIT_SUCCESS);
    }
    else  /* Parent writes argv[1] to pipe */
    {
        close(pipefd[0]); /* Close unused read end */
        dup2(pipefd[1], STDOUT_FILENO);;
        system("ps aux");
        close(pipefd[1]); /* Reader will see EOF */
        wait(NULL);       /* Wait for child */
        exit(EXIT_SUCCESS);
    }
}
```

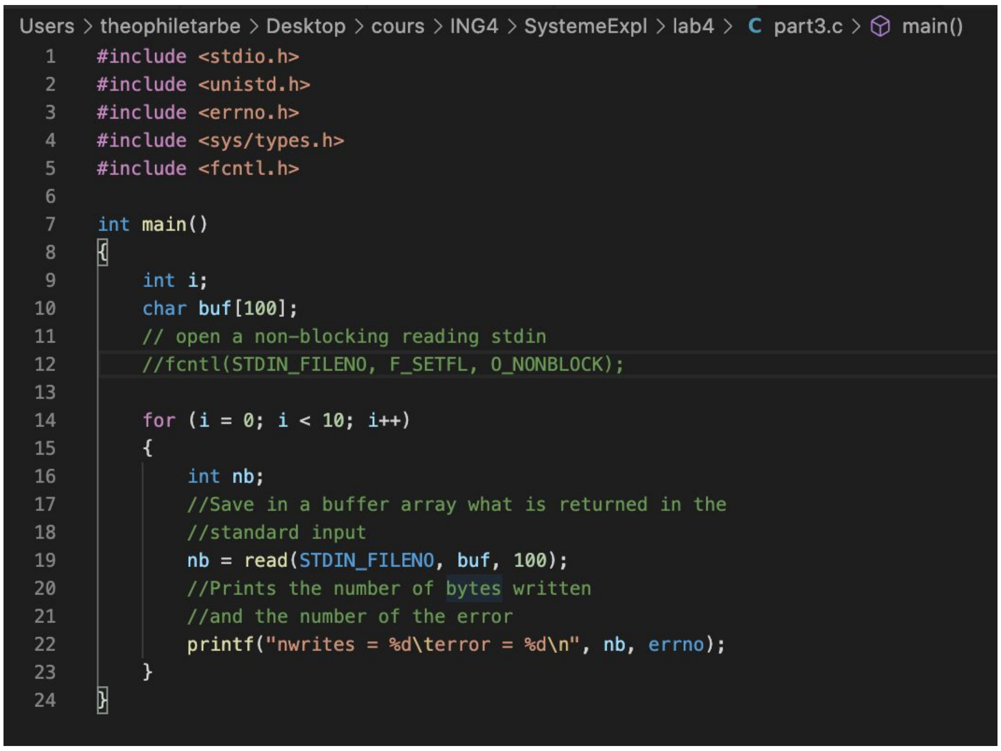
This code will compile the command `ps aux | more` as we can see on the terminal :



```
[→  lab4 gcc -g -o exe2 part2.c
[→  lab4 ./exe2
USER               PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
theophiletarbe     418  64,3  1,8  4566312 153536   ??  R    22sep20 2800:18.08 /System/Library/PrivateFrameworks/Clou
theophiletarbe   10620  12,2  0,1  4418012   9432   ??  S    12:01    29:10.80 /System/Library/CoreServices/CoreServic
theophiletarbe   19011   8,9  0,1  4339568   8928   ??  S     5:31     0:00.05 /System/Library/Frameworks/CoreServices
 mdworker -c MDSImporterWorker -m com.apple.mdworker.shared
theophiletarbe   19010   4,8  0,1  4350948  10468   ??  Ss    5:31     0:00.05 /System/Library/Frameworks/QuickLook.fr
Satellite
theophiletarbe   87438   4,0  0,6  7155104  54252   ??  S    27sep20  84:35.61 /System/Library/CoreServices/Finder.app
_windowserver      232   3,5  0,7 11127932  61824   ??  Ss   22sep20 1084:28.79 /System/Library/PrivateFrameworks/SkyL
theophiletarbe   19009   3,5  0,2  4376640  20368   ??  Ss    5:31     0:00.05 /System/Library/PrivateFrameworks/Cloud
ontents/MacOS/com.apple.CloudDocs.MobileDocumentsFileProvider
root               425   2,9  0,1  4358964   5072   ??  Ss   22sep20 245:36.31 /usr/sbin/filecoordinationd
theophiletarbe   18982   1,9  0,1  4365672  10924   ??  S     5:31     0:00.08 /System/Library/Frameworks/CoreServices
 mdworker -c MDSImporterWorker -m com.apple.mdworker.shared
root               117   1,8  0,1  4407100   5796   ??  Ss   22sep20  11:12.51 /usr/libexec/opendirectoryd
root                 1   1,2  0,1  4360620  12340   ??  Ss   22sep20  47:31.32 /sbin/launchd
root               163   1,0  0,0  4401244   2840   ??  Ss   22sep20   1:19.34 /usr/libexec/syspolicyd
root               312   1,0  0,1  4662124  11812   ??  Ss   22sep20  43:15.21 /usr/libexec/TouchBarServer
```

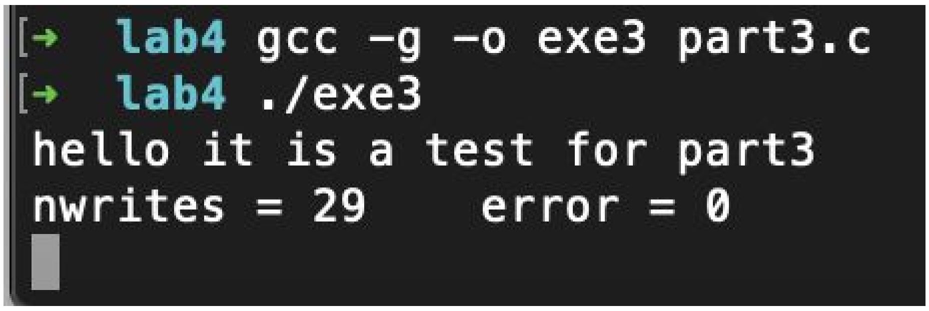
3. Non-Blocking Calls

• Test this code; what does it do ? add annotations to the significant lines



Users > theophiletarbe > Desktop > cours > ING4 > SystemeExpl > lab4 > C part3.c > main()

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

int main()
{
    int i;
    char buf[100];
    // open a non-blocking reading stdin
    //fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);

    for (i = 0; i < 10; i++)
    {
        int nb;
        //Save in a buffer array what is returned in the
        //standard input
        nb = read(STDIN_FILENO, buf, 100);
        //Prints the number of bytes written
        //and the number of the error
        printf("nwrites = %d\terror = %d\n", nb, errno);
    }
}
```

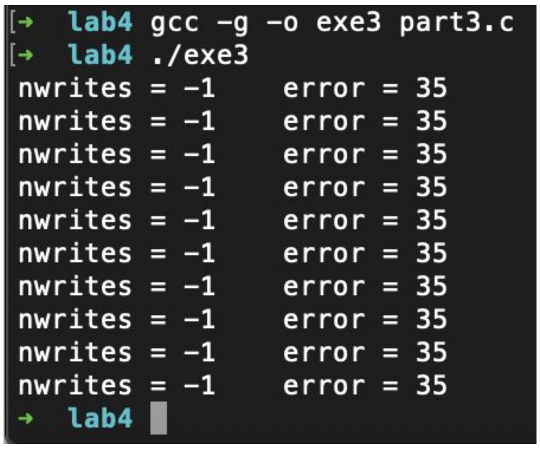


```
[→  lab4 gcc -g -o exe3 part3.c
[→  lab4 ./exe3
hello it is a test for part3
nwrites = 29    error = 0
```

→ Here the code prints the number of bytes written in input, and the error.

• What happens when you uncomment the fcntl line ? Explain.

Now if we uncomment the fcntl line it gives us :



```
[→  lab4 gcc -g -o exe3 part3.c
[→  lab4 ./exe3
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
nwrites = -1    error = 35
→  lab4 
```

→ The function set the state of the standard input on a non block which makes it impossible to read for the read function. As a result it gives us an error everytime.

Sources :

https://man7.org/linux/man-pages/man2/open.2.html

http://manpagesfr.free.fr/man/man2/open.2.html

https://man7.org/linux/man-pages/man2/dup.2.html

http://www.cs.loyola.edu/~jglenn/702/S2005/Examples/dup2.html

https://linuxhint.com/dup2_system_call_c/

https://www.geeksforgeeks.org/c-program-demonstrate-fork-and-pipe/

http://www.octetmalin.net/linux/tutoriels/ps-connaitre-afficher-processus-actifs-a-un-moment-donne-instant-en-ligne-de-commande.php

https://www.geeksforgeeks.org/pipe-system-call/?ref=rp