Lab Session 5: Interprocess communication and files
24296 - Operating systems

Here, we will use inter-process communication to allow several processes to work concurrently on reading and computing CRC of a data file (in blocks of 256 bytes), and both the CRC and data file are passed as arguments. The main file will create two pipes, one (pipe A) for father-children communication, and the other (pipe B) for children towards father communication. Then it will create four children. The schematic can be seen in Figure 1.

The father will then start reading request from the keyboard (standard input), which will have the following format:

```
[generate/get] BlockNumber
```

You can assume that the blocknumbers are always valid, and that the instruction is always either *generate* or *get*. For instance, a possible usage is:

```
generate 42
get 12
get 52
get 42
generate 9
```

The father will parse the instruction from the input string, and write the struct containing that information in pipe A. You can use the following definition

```
typedef struct {
    int blockNumber;
    int isGenerate; //1 if generate, 0 if need to get
} Command;
```

Then, the children will receive the instruction, and process them. For the instructions of type generate, they will read the data file the corresponding block, and write them in the corresponding position of the CRC file. You can use lseek for reading and writing the correct blocks. For the instructions of type get, the process will read the desired CRC number, and write the result in pipe B, together with the original block number.

```
typedef struct {
    int blockNumber;
    unsigned short crc;
} Result;
```

When the standard input of the father is closed (ctrl + D, for the case of the keyboard), the father will close its output pipe. Then, it will start reading from Pipe B, until it is close as well, and writing the result in the standard output. You'll The child will continuously read from pipe A, until it is closed. Then, they will close pipe B and finish. An example of the output of the program, corresponding to the previous input is (the CRC numbers are random):

```
The CRC of block 12 is 1345346
The CRC of block 52 is 75623412
The CRC of block 42 is 1345346
```

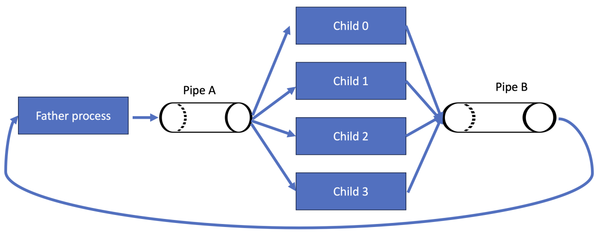

Figure 1: Enter Caption

Hints:

- Remember to close the pipes when needed, otherwise the process will deadlock
- You need to use the appropriate file locks to avoid race conditions.
- Remember that the position of the reading cursor of a file descriptor is shared if the open was done before the fork. It is easier if each child opens its own file descriptor

Delivery

Deliver a zip file (ONLY zip, don't use RAR) and include the previous multithreaded program (call it main.c), the rest of the files and headers required to compile your program, and a document answering the questions below:

1. Explain why two pipes are enough even if there are several children.
2. Explain how the ending is handled (when the standard input of the father closes).
3. An alternative of using file locks would be to use a named semaphore to make any access to the file exclusive. Which disadvantages has this solution?

0.1 System call cheat code

```
int open(char* name, O_RDWR, 0644);
int read(int fd,  char * buff, int n);
int write(int fd,  char * buff, int n);
close(int fd);

int fork();
int pipe(int fd[2]);
int wait(int* status);
execvp(char *name, char* argv[]);        // name is searched in PATH variable
_exit(int status);
off_t lseek(int fd, int offset, int whence)
// whence SEEK_SET to start from the beginning of the file

int file_lock_shared(int fd, int offset, int whence);
int file_lock_exclusive(int fd, int offset, int whence);
int file_lock_unlock(int fd, int offset, int whence);
```