# CECS 326 - PROJECT REPORT 1

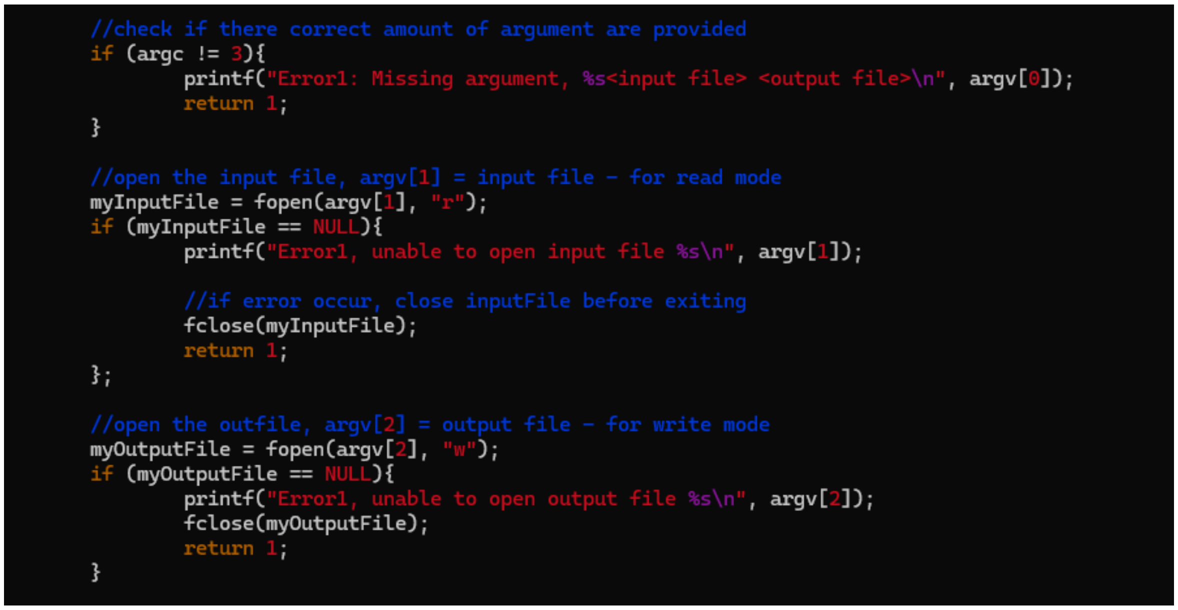
Name(s): My Lu Student ID number: 029895591
Name(s):Fozhan Babaeiyan Student ID number: 029701865

**Project 1: Warm up of Interprocess Communication**

## Variable use/declare:

- **fd[2]** : file descriptors for pipe : create a communication channel for two end
    - **fd[0]** : read end of the pipe
    - **fd[1]** : write end of the pipe
- **FILE *myInputFile, *myOutputFile** : pointer to FILE object, for the input/output file
    - **myInputFile** : input file, the file you wanna read from
    - **myOutputFile** : output file, the file you wanna write to
- **Char myBuffer[256]** : value to hold the file content
- **ssize_t bytesRead** : store the result of functions that return the number of bytes read/write
    - Source : https://jameshfisher.com/2017/02/22/ssize_t/#:~:text=In%20short%2C%20ssize_t%20is%20the,%23include%20%3Csys%2Ftypes.
    - Function ssize_t read :
        - ssize_t read(int fildes, void *buf, size_t nbyte);
    - Function ssize_t write:
        - ssize_t write(int fildes, const void *buf, size_t nbyte);

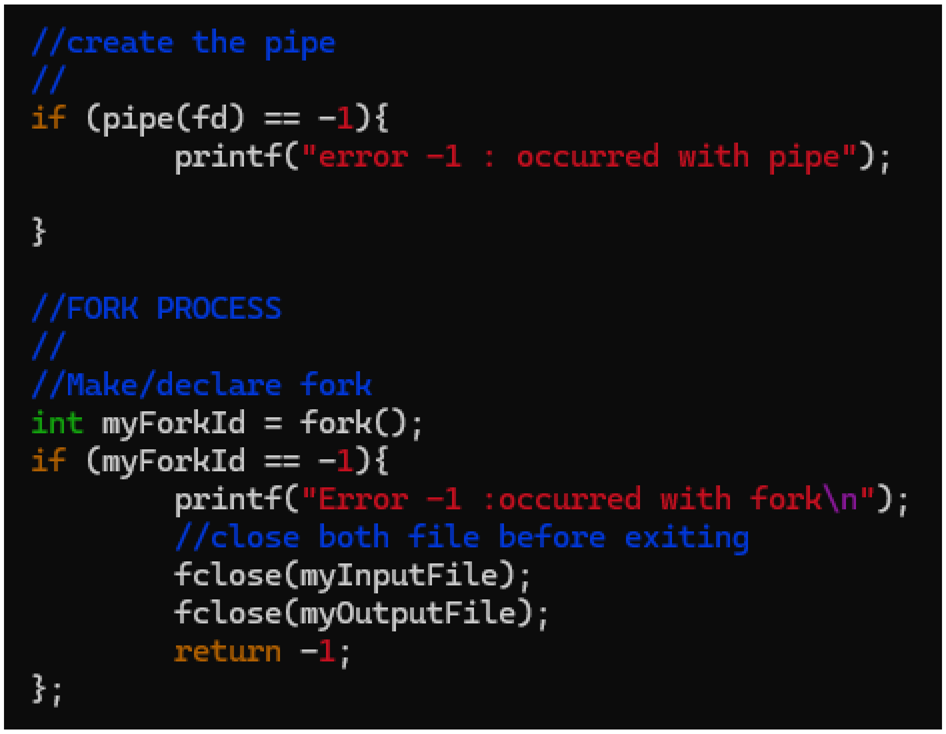
## Code/Function used:



```
//check if there correct amount of argument are provided
if (argc != 3){
        printf("Error1: Missing argument, %s<input file> <output file>\n", argv[0]);
        return 1;
}

//open the input file, argv[1] = input file - for read mode
myInputFile = fopen(argv[1], "r");
if (myInputFile == NULL){
        printf("Error1, unable to open input file %s\n", argv[1]);

        //if error occur, close inputFile before exiting
        fclose(myInputFile);
        return 1;
};

//open the outfile, argv[2] = output file - for write mode
myOutputFile = fopen(argv[2], "w");
if (myOutputFile == NULL){
        printf("Error1, unable to open output file %s\n", argv[2]);
        fclose(myOutputFile);
        return 1;
}
```

- Checking if there are 3 argument parse in the command
  - Handle : missing file
  - Handle : input more than 3 files
- Checking if the Input File is valid file, check if we can open it
- Checking if the Output File is valid file, check if we can open it

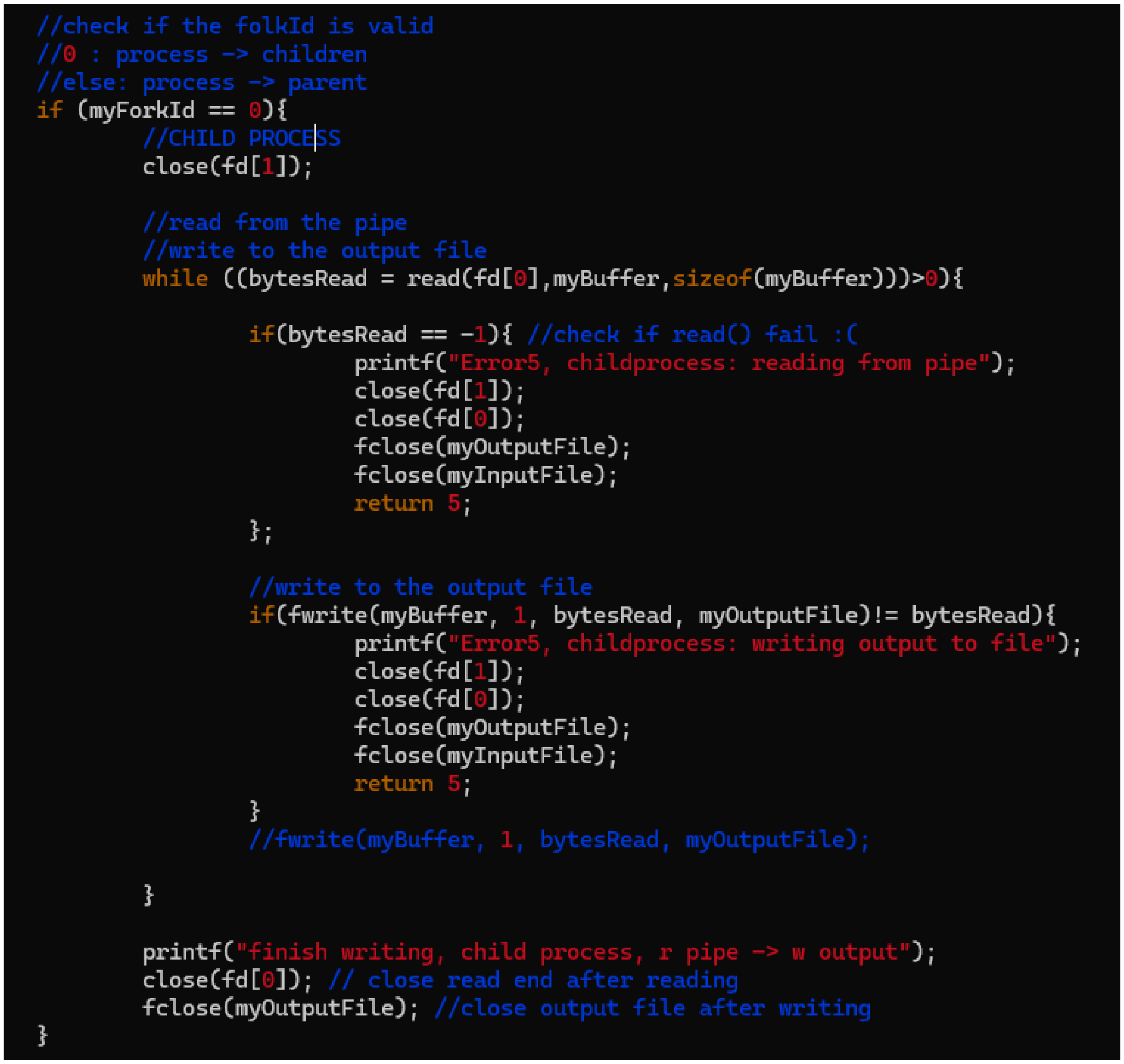
## Making Pipe and Fork Process



```
//create the pipe
//
if (pipe(fd) == -1){
        printf("error -1 : occurred with pipe");

}

//FORK PROCESS
//
//Make/declare fork
int myForkId = fork();
if (myForkId == -1){
        printf("Error -1 :occurred with fork\n");
        //close both file before exiting
        fclose(myInputFile);
        fclose(myOutputFile);
        return -1;
};
```

- Creating pipe :
    - Using **pipe(fd)**
    - Create a communication channel between the parent and the children
    - Purpose : parent write data into pipe, and the child will read data from pipe
    - **Parent** → write → **pipe** ← read ← **Children**
- Creating fork process:
    - Using **fork()**
    - Create a new process (aka: **Child Process**). Return **int** to determines whether the current process is the parent or child
- If there exist an **error**:
    - Close both file before exit
    - Return -1

## Child Process

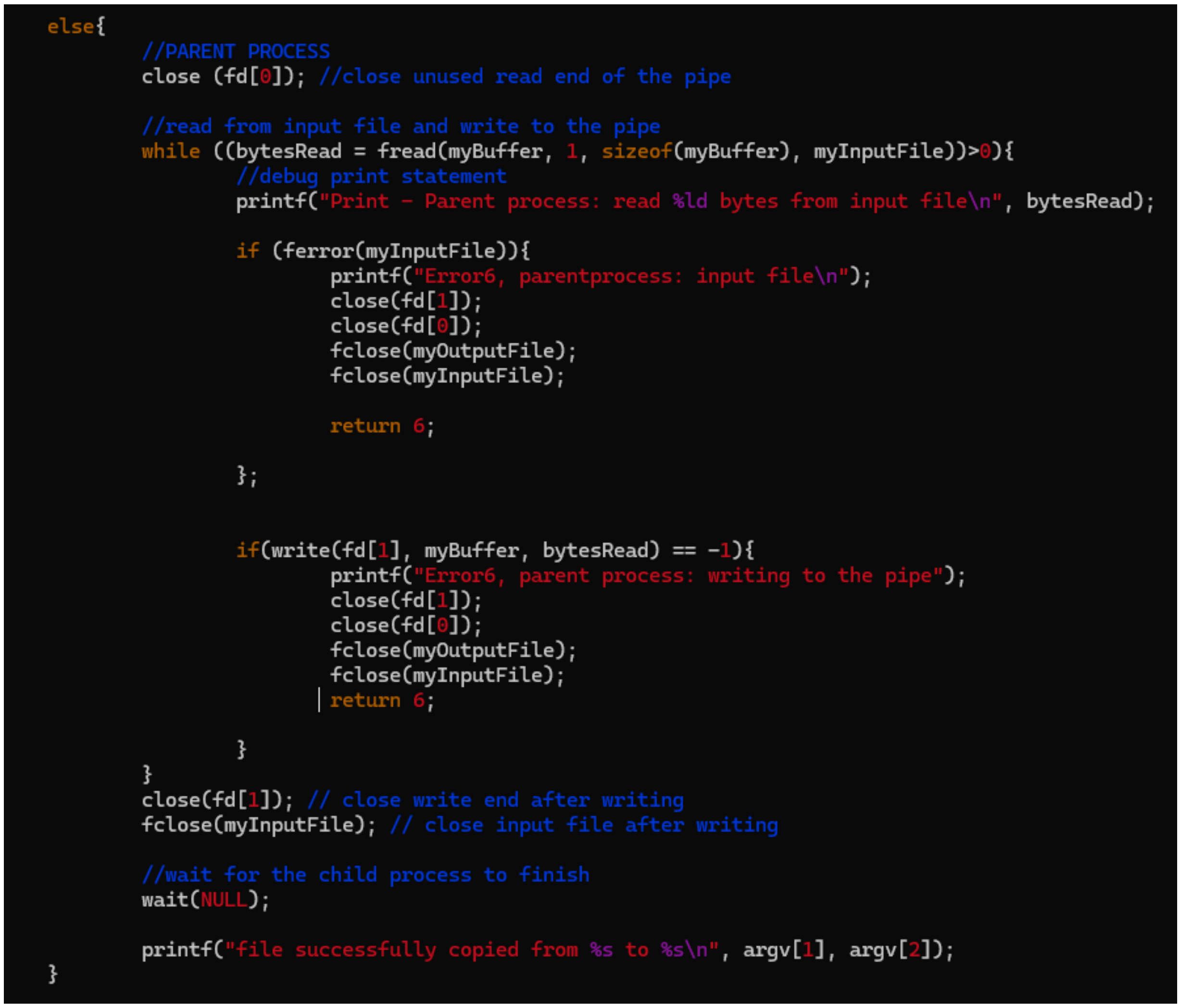


```
//check if the folkId is valid
//0 : process -> children
//else: process -> parent
if (myForkId == 0){
        //CHILD PROCESS
        close(fd[1]);

        //read from the pipe
        //write to the output file
        while ((bytesRead = read(fd[0],myBuffer,sizeof(myBuffer)))>0){

                if(bytesRead == -1){ //check if read() fail :(
                        printf("Error5, childprocess: reading from pipe");
                        close(fd[1]);
                        close(fd[0]);
                        fclose(myOutputFile);
                        fclose(myInputFile);
                        return 5;
                };

                //write to the output file
                if(fwrite(myBuffer, 1, bytesRead, myOutputFile)!= bytesRead){
                        printf("Error5, childprocess: writing output to file");
                        close(fd[1]);
                        close(fd[0]);
                        fclose(myOutputFile);
                        fclose(myInputFile);
                        return 5;
                }
                //fwrite(myBuffer, 1, bytesRead, myOutputFile);

        }

        printf("finish writing, child process, r pipe -> w output");
        close(fd[0]); // close read end after reading
        fclose(myOutputFile); //close output file after writing
}
```

- **close(fd[1])** : closing pipe to make sure there is no interference for children when it process reading from it
- while ((bytesRead = read(fd[0], myBuffer, sizeof(myBuffer))) > 0)
  - Continuously reading the data from pipe into the **myBuffer** until all the data has been read.
    - **read()** : function tries to read up to sizeof(myBuffer)
      - ssize_t read(int fildes, void *buf, size_t nbyte);
      - Source : https://pubs.opengroup.org/onlinepubs/009604499/functions/read.html
  - **While writing**:
    - We keep check the read-process if it fail in the middle of read

- We keep check the write-process:
    - fwrite() : Check all the bytes read from the pipe
    - If the number of bytes written is different → return -1 [error]
- **If** either error got raise, I will close up all the file before exit
- **Else**: it will printf statement to indicate that I successful copy the data from inputFile into outputFile
    - Close pipe end after reading
    - Close outputFile after writing
    - Exit

## Parent Process



```
else{
        //PARENT PROCESS
        close (fd[0]); //close unused read end of the pipe

        //read from input file and write to the pipe
        while ((bytesRead = fread(myBuffer, 1, sizeof(myBuffer), myInputFile))>0){
                //debug print statement
                printf("Print - Parent process: read %ld bytes from input file\n", bytesRead);

                if (ferror(myInputFile)){
                        printf("Error6, parentprocess: input file\n");
                        close(fd[1]);
                        close(fd[0]);
                        fclose(myOutputFile);
                        fclose(myInputFile);

                        return 6;

                };


                if(write(fd[1], myBuffer, bytesRead) == -1){
                        printf("Error6, parent process: writing to the pipe");
                        close(fd[1]);
                        close(fd[0]);
                        fclose(myOutputFile);
                        fclose(myInputFile);
                        return 6;

                }
        }
        close(fd[1]); // close write end after writing
        fclose(myInputFile); // close input file after writing

        //wait for the child process to finish
        wait(NULL);

        printf("file successfully copied from %s to %s\n", argv[1], argv[2]);
}
```

- close **(fd[0])** : closing read end pipe since **Parent Process** only writes to pipe
- while ((bytesRead = fread(myBuffer, 1, sizeof(myBuffer), myInputFile)) > 0)

- Continuously read from the inputPut file into **myBuffer** as long as **fread()** reads more than 0 bytes
- sizeof(myBuffer) : fread() attempts to read up to this many bytes at a time
- While reading:
  - We check if an error occur when reading file from the input file
  - While reading data from Input, we also write it into pipe, if write fail → it will print out the error statement and close all the file before exiting
  - Else: if will write the data from input file into pipe
    - InputFile ← **read** – **ParentProcess** – **write** → Pipe
    - When finish writing all the data into pile → closes the input file
- wait(NULL) :
  - Source reference:
    - https://stackoverflow.com/questions/60475312/fork-wait-and-pipe-in-c
  - The **Parent Process** waits for the **Children Process** to finish. Ensuring the Parent Process doesn't terminate, end before the Children Process complete from reading from pipe and write to Output File
- After finish both of the process complete, before exit, the SUCCESS print statement got print out, indicate the copy from input file to output file success

---

## Source Reference while working on the project

1. https://www.geeksforgeeks.org/fread-function-in-c/
2. https://www.educative.io/answers/what-is-a-pipe-in-c
3. https://stackoverflow.com/questions/47503798/write-on-pipe-in-c
4. https://stackoverflow.com/questions/60475312/fork-wait-and-pipe-in-c
5. https://stackoverflow.com/questions/16163154/read-from-pipe-line-by-line-in-c
6. https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_pipes.htm
7. https://youtu.be/Mqb2dVRe0uo?si=HD-9-vL5p9DkgGBk
8. https://youtu.be/cex9XrZCU14?si=WfumpqBM0qwQzMxN