**Final Project Report - Pipe Method**

**Description of design:**

"A parent process (the producer) passes 100 random integer values to the child process (the consumer) via a pipe. The child process is created by the fork() function in the main method. The producer uses the "write" end of the pipe (denoted by fd[1]) to write data to the consumer, whereas the consumer uses the "read" end of the pipe (denoted by fd[0]) to read data from the pipe."

**System Calls to be used (for each IPC method selected):**

System calls used by the main method:

(1) line 17 pipe():This is a system call that requests the operating system to create a pipe

(2) line 22 fork():This is a system call that create a duplicate child process

System calls used by the Producer:

(1) line 33 fopen():This is a system call to request the system to create/open a file

(2) line 45 fclose():This is a system call to request the system to close a file

(3) line 50 write():This is a system call to request the system to write the specified data into a pipe(fd[1])

(4) line 51 close():This is a system call that close a file discriptor

System calls used by Consumer:

(1) line 57 fopen():this is a system call to request the system to create/open a file

(2) line 73 fclose():():This is a system call to request the system to close a file

(3) line 66 read():This is a system call to request the system to read the specified data from a pipe(fd[0])

(4) line 74 close():This is a system call that close a file discriptor

**Source code**:

#include <stdio.h>

#include <string.h>

#include <time.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#define BUFFER\_SIZE 100

int main() {

int fd[2]; //a file descriptor

pid\_t pid;//process id variable

//create the pipe, file descrptors: fd[0] read, fd[1] read,

if(pipe(fd)==-1){

fprintf(stderr,"Pipe fail");

return 1;

}

//fork a child process

pid = fork();//fork() only work on UNIX like sys

if(pid<0){ //error occurred

fprintf(stderr,"Fork failed");

return 1;

}

/\*-------------------Producer Method-------------------\*/

if(pid>0){

//Generate and Store all randomNumbers in to producer\_randomNumber.txt

FILE\* file;

file = fopen("producer\_randomNumber.txt", "w");

if (file == NULL) {

printf("fail to create a producer file!");

return -1;

}

int randomNUmbers[BUFFER\_SIZE];

srand(time(NULL));

for(int i = 0; i < BUFFER\_SIZE; i++) {

int randomNumber = rand() % 1000 + 1;

randomNUmbers[i]=randomNumber;

fprintf(file,"%d\n",randomNumber); //write to the .txt file

}

fclose(file);

//close the unused end of pipe

close(fd[0]);

//Write to the pipe

write( fd[1],randomNUmbers,sizeof(randomNUmbers));

close(fd[1]);

}

/\*-------------------Consumer Method-------------------\*/

if(pid==0){

FILE \*file;

file = fopen("consumer\_randomNumber.txt","w");

if(file==NULL){

printf("fail to create a producer file!");

return -1;

}

//close the write end of pipe

close(fd[1]);

int receivedArray[BUFFER\_SIZE];

read(fd[0],receivedArray,sizeof(receivedArray));

printf("Child process received: ");

printf("\n");

for(int i=0; i< BUFFER\_SIZE; i++){

printf("%d",receivedArray[i]);

fprintf(file,"%d\n",receivedArray[i]); //write to the .txt file

printf("\n");}

fclose(file);

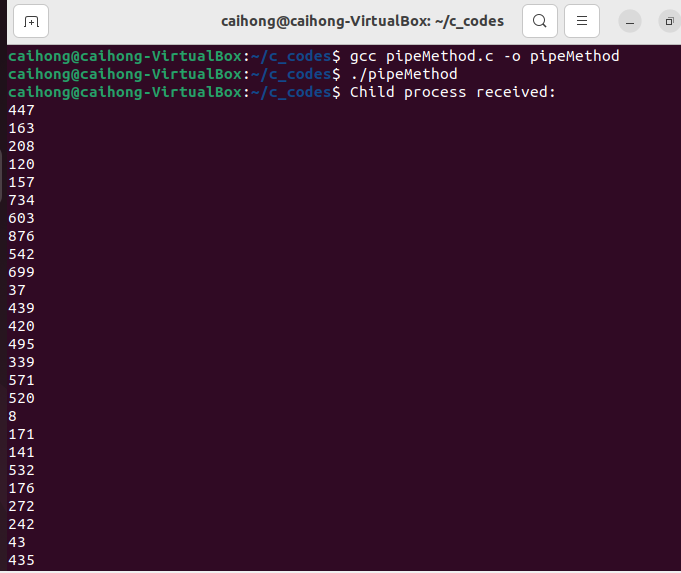
close(fd[0]);

}

return 0;

}

**Result**:

****

**Overall Discussion:**

The code from textbook 3.7.4 can be used to cover 90% of this project. The read() and write() methods might be somewhat confusing since they share the same names as the write and read methods in Java for OutputStream and InputStream, respectively. Also, when comparing the pipe method to the socket method, it appears that the pipe method can only be used for communication between processes in the same system, while sockets can be used for communication between two processes across a network. For this project, I created only one pipe. Given that a pipe can only be used one way, the producer must close the read end when it wants to write, and vice versa for the consumer process.