Group Project 1: Warm up of Interprocess Communication

CECS 326 – Operating Systems

You should submit the required deliverable materials on Canvas by **11:55pm, Feb 16th (Sunday), 2025.** We provide sample code files that you can complete the required functions. Note that you are flexible to use the sample file as you can design all functions by yourself without following the sample files. **Only one submission** of each group. This tutorial gives details in the format of C, you can also use Python for the submission.

1. Description

In this project, you will design and implement a file-copying program named ***filecopy.c*** that utilizes ordinary pipes for inter-process communication. The program will be passed two parameters: the name of the file to be copied and the name of the destination file. It will create a pipe (please find more info of pipe for interprocess communication by yourself), write the contents of the source file to the pipe, and then have a child process read from the pipe and write to the destination file.

Inter-process communication (IPC) is a fundamental concept in operating system design, allowing processes to exchange data and synchronize their actions. Pipes are one of the simplest forms of IPC, enabling communication between processes running on the same system. Pipes have a read end and a write end, allowing one process to write data into the pipe while another process reads from it.

The filecopy.c program demonstrates how pipes can be used to transfer data between processes efficiently. By leveraging pipes, the program avoids the need for intermediate storage of the entire file contents in memory, making it suitable for copying large files with minimal memory overhead.

This project will provide you with hands-on experience in system programming, file handling, and error handling in the C programming language. You will learn how to create pipes, fork processes, and manage file descriptors to implement efficient file copying functionality. Additionally, you will gain insights into the intricacies of inter-process communication and the coordination between parent and child processes.

Objective

The objective of this project is to gain practical experience in using pipes for inter-process communication in C programming and to understand the basics of file handling.

Features

- **Command-line Arguments**: The program will accept command-line arguments specifying the source file and the destination file.
- **Pipe Creation**: It will create an ordinary pipe to transfer data between the parent and child processes.
- **File Reading/Writing**: The parent process will read the contents of the source file and write them to the pipe. The child process will then read from the pipe and write the data to the destination file.
- **Error Handling**: Proper error handling will be implemented for file operations and pipe creation.

Project Structure

The project will consist of a single C source file (filecopy.c). Below is an outline of the main components:

- **Main Function**: The main function will parse command-line arguments, create a pipe, fork a child process, and handle file copying.
- **File Copying Logic**: This section will contain the logic for reading from the source file and writing to the pipe in the parent process, as well as reading from the pipe and writing to the destination file in the child process.
- **Error Handling**: Error handling will be implemented for file operations and pipe creation to ensure robustness.
- **Makefile (Optional)**: You can include a Makefile to facilitate compilation and linking of the program.

Project Implementation Steps

- **Parse Command-Line Arguments**: Extract the source file name and the destination file name from the command line.
- **Open Source and Destination Files**: Open the source file in read mode and the destination file in write mode.
- **Create a Pipe**: Use the pipe() system call to create an ordinary pipe.
- **Fork a Child Process**: Use the fork() system call to create a child process.
- **Parent Process**: In the parent process, read from the source file and write to the write end of the pipe.
- **Child Process**: In the child process, read from the read end of the pipe and write to the destination file.
- **Close File Descriptors:** Close all file descriptors and handle any errors that may occur during file operations or pipe creation.
- **Compile and Test: Compile the program and test it by copying various files to different destinations, e.g.** *./filecopy input.txt copy.txt*

One example of sample output: (you can define your own output)

Here's a sample output of the filecopy.c program when invoked with appropriate command-line arguments:

```
$ ./filecopy input.txt copy.txt
File successfully copied from input.txt to copy.txt.
```

```
$ cat input.txt
This is the content of the input file.
It contains some text that we want to copy.

$ cat copy.txt
This is the content of the input file.
It contains some text that we want to copy.
```

If any errors occur during the execution of the program, appropriate error messages will be displayed to provide information about the encountered issue. For example:

```
$ ./filecopy non_existing_file.txt copy.txt
Error: Unable to open source file 'non_existing_file.txt'.
```

3: The Required Deliverable Materials

(1) A README file, which describes how we can compile and run your code.
(2) Your source code, you may use a Makefile (for C) and be submitted in the required format.
(3) Your short report, which discusses the design of your program.
(4) A recorded video shows the output and runtime

3. Submission Requirements

You need to strictly follow the instructions listed below:

1) This is a **group project**, please submit a .zip/.rar file that contains all files, only one submission from one group.

2) Make a **video** to record your code execution and outputs. The video should present your name or time as identification (You are suggested to upload the video to YouTube and put the link into your report).

3) The submission should include your **source code** and **project report**. Do not submit your binary code. Project report should contain your groupmates name and ID.

4) Your code must **be able to compile**; otherwise, you will receive a grade of zero.

5) Your code should not produce anything else other than the required information in the output file.

6) If you code is **partially completed**, please explain the details in the report what has been completed and the status of the missing parts, we will grade it based on the entire performance.

7) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	15 pts
Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your group mates	5 pts
Code can be compiled and shows correct outputs	40 pts

4. Policies

1) Late submissions will be graded based on our policy discussed in the course syllabus.
2) Code-level discussion is **prohibited**. We will use anti-plagiarism tools to detect violations of this policy.
3) AI-tools such as ChatGPT, GPT3, etc. are prohibited in both code and contents.