CSE-381: Systems 2
Exercise #5
Max Points: 20

You should save/rename this document using the naming convention `MUid_Exercise5.docx` (example: `raodm_Exercise5.docx`).

Objective: The objective of this exercise is to:

- Use anonymous pipes to create software systems by interconnecting processes
- Named pipes aka FIFOs (First-In-First-Out).
- Work on practice problems for Exam #1

Submission: Once you have completed this exercise, upload via the CODE plug-in

- This document saved as PDF file and named with the convention `MUid_Exercise5.pdf`.
- The completed C++ source file developed in this exercise.

You may discuss the questions with your instructor or TA.

Name: RAObot

The various command associated with this exercise are to be executed on the Linux server for this course. You will not need to use NetBeans for this part of the exercise.

Part #1: Using anonymous pipes (aka FIFOs) [5 points]

Estimated time to complete: 25 minutes

It may be helpful to quickly recollect concepts of working with pipes from the following review video: https://youtu.be/yTjMXr_dwCg

Background: Anonymous pipes provide a convenience mechanism for IPC between processes running on the same computer. Anonymous pipes are also convenient mechanisms to interconnect processes on a Linux machine to perform tasks -- **giving rise to a software system or a "software pipeline"**. Most of the command-line programs and utilities in Linux are designed to facilitate the use of anonymous pipes as they read and write data to standard input (`std::cin`) and standard output (`std::cout`) respectively. The `bash` shell enables "wiring" or "connecting" standard I/O streams via the `|` (pipe) operator to setup utility software pipelines for performing sophisticated tasks as shown in the example below:

Example: List all processes for user `raodm`

```
$ ps -fe | grep raodm
```

DUE Before: **The end of your lab session**

There are too many useful Linux commands/tools to cover, but here are a few common ones that you will be using in this exercise (**copy-pasted from `CommonMethodsAndCommands.pdf`**):

Command	Description	Example
`cut`	Cut out columns specified by `-f` ; columns delimited by 1 character specified by `-d`	`$ cut -d "," -f 1,2,4 faculty.txt` `$ ps -fea | cut -d " " -f1`
`grep`	Search for string/regular-expression in file or standard input	`$ grep "ra?dm" test.txt`
`sort`	Sort file/inputs based on key column specified via `-k` ; columns delimited by 1 character specified via `-t`	`$ sort -k 2 -t , faculty.txt` `$ cut -d" " -f2,3 classes.txt | sort -t, -k 1`
`join`	Join 2 files based on common values in 2 given columns. Each file must be sorted on common value/column specified by `-1` (for file 1) and `-2` (for file 2); columns delimited by 1 character specified via `-t`	`$ join -t, -1 2 -2 1 classes.txt courses.txt` `$ sort -k2 -t, classes.txt | join -t, -1 2 -2 1 - courses.txt`

Exercise: This exercise is purely a problem solving one that requires you to innovate by using the above 4 commands to perform the following operations (that would typically be performed in an interview setting as well):

1. For this part of the exercise you are supplied with the following 3 Comma Separated Values (CSV) files: `classes.txt`, `courses.txt`, and `faculty.txt`. These are simple text files. View the files to understand the data in them. `Scp` these 3 files to the Linux server as suggested below:

   ```
   $ scp classes.txt courses.txt faculty.txt muid@os1.csi.miamioh.edu:
   ```

   Next develop a sequence of 1 or more commands (commands are connected via pipes) to extract the data for the following problems. Sample outputs are show for each problem below.

2. List just the faculty names (Hint: `cut`):

   Sample output:

   ```
   Campbell
   Kiper
   Rao
   ```

   Copy-paste command and the output from your command in space below:

   ```
   $ cut -d"," -f2 faculty.txt
   Campbell
   Kiper
   Rao
   ```

3. List the classes (as in `cse278`, `cs301`, etc.) taught in `Fall17` (Hint: `grep` + `cut`)

   Sample output:

```
cse278
cse278
cse301
cse443
```

   Copy-paste command and the output from your command in space below:

```
$grep "Fall17" classes.txt | cut -d"," -f2
cse278
cse278
cse301
cse443
```

4. List the classes and names of courses taught in `Spring18`:

   Sample output:

```
cse278,Systems 1
cse301,Software Architecture
cse381,Systems 2
```

   Note: The following command will come in handy. Run the command below and observe how the join command combines 2 files based on the values of the course number (*e.g.*, cse278). **Note**: You will notice a warning about files not being sorted. That is expected. See the use of `sort` and `join` in the table of examples above.

```
join -t, -1 2 -2 1 classes.txt courses.txt
```

   Copy-paste command and the output from your command in space below:

```
$ grep "Spring18" classes.txt | join -t, -1 2 -2 1 - courses.txt | cut
-d"," -f1,4
cse278,Systems 1
cse301,Software Architecture
cse381,Systems 2
```

5. Optional challenge (come back to this one at the end if you have time): List classes, course name, and faculty names of courses taught in `Spring18`:

   Sample output:

```
cse278,Systems 1,Campbell
cse301,Software Architecture,Kiper
cse381,Systems 2,Rao
```

   Copy-paste command and the output from your command in space below:

```
$ grep "Spring18" classes.txt | join -t, -1 2 -2 1 - courses.txt | join
-t, -1 3 -2 1 - faculty.txt | cut -d"," -f2,4,5
cse278,Systems 1,Campbell
cse301,Software Architecture,Kiper
cse381,Systems 2,Rao
```

Part #2: Using named pipes (aka FIFOs) [4 points]

Estimated time to complete: 25 minutes

Background: A named pipe (also known as a FIFO because it performs First-In-First-Out type IPC) is a commonly used pipe concept on Linux and Unix-like systems. It is one of the different methods for inter-process communication. The concept of named pipes is also found in Microsoft Windows, although the semantics differ substantially. Unlike a traditional pipe that is "unnamed" (as it exists anonymously and persists only for as long as the interacting process is running, named pipes exist beyond the lifetime of processes), a named pipe is system-persistent and exists beyond the life of the process. It can be deleted once it is no longer being used. Processes generally attach to the named pipes **as if they were normal files** to perform inter-process communication (IPC) by reading and writing data to it. Usually pipes are used to exchange text data. However, the pipes may also be used for interchange of binary data.

Exercise: This exercise requires you to create a FIFO and use it for exchanging data between two processes via the following experimental procedure:

1. Create a FIFO using the following command at the shell prompt (where `MUid` is your Miami University unique ID):

   ```
   $ mkfifo /tmp/MUid
   ```

   Next, setup the permission on the pipe using `chmod` such that: user (*i.e.*, you) can read-and-write while group and other users can only write to your FIFO (i.e., `/tmp/MUid`). To verify the named pipe has been set with correct permissions, run `ls -l /tmp/MUid` to list the file entry and verify the permissions.

   Copy-paste the `chmod` command and the `ls -l` output to validate the permissions on the pipe have been set correctly:

   ```
   $ chmod u+rw-x,og+w-rx /tmp/raodm
   $ ls -l /tmp/raodm
   prw--w--w- 1 raodm uuidd 0 Feb 21 13:03 /tmp/raodm
   ```

2. Now use the FIFO that was just created to perform some simple IPC between two different processes. For this we will create one process that sends data to the FIFO and another one that reads data from the FIFO as directed below:

a. Open two different terminal window and log into the Linux server (now you will have 2 different terminals, both logged onto the Linux server).
b. In one of the windows print the contents of the FIFO using the following command (where *MUid* is the FIFO that was created earlier):

```
$ cat /tmp/MUid
```

c. In the other terminal write data to the FIFO using the following command:

```
$ ps -fe > /tmp/MUid
```

d. You should observe the output displayed in window in which the earlier `cat` command was run.

3. Observe how the FIFO enables communication between processes using following steps:
   a. Run "`ls -l`" and record the size (number before date) of the FIFO in the space below:

      Size (in bytes) of the FIFO was: 0

   b. Don't run `cat` to read data of FIFO. Instead, just write output of `ps -fe` command to the named pipe (as shown earlier). This process may appear to hang, which is normal -- this is because the FIFO's internal memory buffer is full and it is waiting for another process to read data to free-up space.
   c. After redirecting data from the `ps -fe` command, from another terminal check the size of the FIFO reported by "`ls -l`". Its size should remain unchanged (because all the data is stored in memory by the OS).
   d. Finally, you can `cat` the data in your FIFO (in another terminal) and observe the full output `ps` is returned as the data was stored in internal OS buffers.

Play with permissions. Now remove write permissions on your FIFO for everyone using the `chmod` command. Verify permission settings on the FIFO using `ls -l` command:

Copy-paste the `chmod` command and the `ls -l` output to validate the permissions on the pipe have been set correctly:

```
$ chmod a-w /tmp/raodm
$ ls -l /tmp/raodm
pr-------- 1 raodm uuidd 0 Feb 21 13:03 /tmp/raodm
```

4. Try to write some data to the FIFO and ensure that permission is denied to write data to the FIFO.

5. You may re-enable write permissions on the FIFO for everyone and try using the named pipe.

6. Ask a peer in your laboratory session to write some data to your named pipe (i.e., `/tmp/MUid`) and see how different users can write data to it. They will not be able to read data sent to you (because, remember, you have disabled read permissions for everyone other than yourself)

7. Finally delete the FIFO that you created using the `rm` command as shown below:

```
$ rm /tmp/MUid
```

Part #3: Practice problem for Exam #1 [5 points]

Estimated time to complete: 25 minutes

The programming exercise question(s) below use the following input format – The data is read from a file. The file contains typical Linux file information, with 1-line per file. Each line consists of 4 tab-separated columns in the following format:

```
permissions    userID    groupID    path
```

1. **permissions**: Standard 9-characters (`rwxrwxrwx`) Linux permissions. You are required to know the format and meaning of the permissions. If you are not sure, review the pertinent slides from `07_IPC.pdf`.
2. **userID**: The ID of the user that owns the file *(e.g.*: `raodm`, `bob`, etc.)
3. **groupID**: The ID of the group with whom the file is associated (e.g.: `admin`, `staff`, etc.)
4. **path**: Full path to the file. This is guaranteed to be 1 word (no spaces, special characters, etc.)

Notes: Review the supplied input file. Notice that the input file format is designed to be simple so that it is trivial to read and process input **So, don't complicate input processing**.

Setting up exercise in NetBeans

1. Via `NetBeans`, on the Linux server `os1.csi.miamioh.edu`, create a Miami University C++ Project named `exercise5`.
2. Download the supplied files and `scp` them to your NetBeans project folder on the server. Of course, you can `scp` all of the files in one shot as shown below –

```
scp ex5.cpp file_list1.txt muid@os1.csi.miamioh.edu:NetBeansProject/exercise5
```

3. Add source files to your `NetBeans` project.

Exam Style Question

Implement the `main` method to print path of files in a data file (1st command-line argument) that are readable by a given user (2nd command-line argument). For example, given the data in `file_list1.txt`, for user `bob`, the program should print the path as shown in the adjacent output. **Note: Full-points only for concise solutions**.

```
$ ./exercise5 file_list1.txt bob
/usr/share/url_tester
ps
./joke1
./joke4
/usr/share/bookmarks.db
```

Part #4: Submit files to Canvas

Submission: Once you have completed this exercise, upload via the CODE plug-in

- This document (duly filled with the necessary information) saved as PDF file and named with the convention `MUid_Exercise5.pdf`.
- The completed C++ source file developed in this exercise.

Upload each file individually onto Canvas. Do not upload zip/7zip/tar/gz or other archive file formats for your submission.