<u>Advanced Operating Systems - Homework 2</u>

Motivation:

You have learned system protection in Chapters 4 and 5. This homework asks you to implement some ideas from both chapters by constructing TCP/IP connections between computers.

Homework Description:

In the UNIX file system, each file has the access rights of read, write, and execution for the file owner, group members, and other people. For example, suppose that a student Ken creates a file named "homework2.c" with reading and writing permissions. He allows his group member, Barbie, in the same group "AOS", to read the file but disallows other people to touch the file. Then, you can see the file description in Ken's directory:

```
-rw-r-----    Ken    ASO    29334    Nov 5 2015    homework2.c
```

Specifically, "homework2.c" has 29,334 bytes and it is created by Ken in ASO group on Nov. 5, 2015. Barbie can read the file since the 2nd 'r' is set on. Other people cannot touch the file since the 3rd group of r/w/x bits are all set off.

Homework 2 has two parts. In the first part, you have to create one server to store and manage files for clients. Three groups of clients should be created, called "AOS_students", "CSE_students", and "other_students", where each group contains at least two clients. Similar to the UNIX file system, you need to specify the reading and writing permission of each file, for the file owner, group members, and other people. When a file is allowed to read (or write), the client is able to download (or upload) that file. When a client is requesting an operation on a file without permission, the server should prohibit it and print out a message to show the reason. Each client can dynamically create a file but should specify all the access rights. For example, Ken can execute the following commands:

1) **new** homework2.c rwr-r-
2) **read** homework2.c
3) **write** homework2.c o/a
4) **change** homework2.c rw----
5) **information** homework2.c

The 1st command, "new", helps Ken create a new file on the server, where the last parameter assigns the file's permissions ('r' and 'w' respectively represent reading and writing permissions, while '-' indicates no permission). The 2nd command, "read", allows Ken to download the file from the server (only if he has the corresponding permission and the file does exist). The 3rd command, "write", allows Ken to upload (and revise) an existing file, where the third parameter can be either 'o' or 'a', which allows Ken to either overwrite the original file or append his data in the end of the file, respectively. Similarly, Ken can write the file only if he has the corresponding permission and the file exists. The 4th command, "change", is used to modify the file's permissions. The revised permissions take effect for the following operations **AFTER** the change command. The 5th command, "information", shows the information of the file (including its permissions, creator, group, file size, date created, and filename). Notice that all clients operate the files in the same directory on the server side. The server has to use **CAPABILITY LISTS** in order to manage the permissions of files. **You have to show TAs how the capability lists change for each operation on the server side.**

When a client is writing a file, other clients cannot read or write the same file. In addition, when a client is reading a file, other client cannot write that file. However, it is safe for multiple clients to simultaneously read the same file. Therefore, in the second part of the homework, you are asked to apply the above rules in your server-client architecture. **You need to show all the above behaviors to TAs**. In particular, your server must be able to connect multiple clients and allow multiple clients to read/write files "concurrently" (Therefore, your files should be large enough to show the above behaviors). Using fork() or multiple threads in your homework is highly encouraged.

Requirements:

You are asked to use the **standard C socket library** in UNIX-compatible systems. Makefile must be provided. TA has the right to deduct points if you do not provide a CORRECT makefile. Besides, DON'T forget to demonstrate your programs to TA (otherwise, you will get zero point).

Grading Policy:

You don't have to submit any report. The first and second parts of this homework deserve 60% and 40%, respectively. Extra bonus can be given (depending on TA) if you can adopt multiple processes/threads on the server side, or you have special design in the homework. The due day of this homework will be 2015/12/10.