CSC 552 Project 2

Points: 25
Due: TBD. Submit late at your own risk policy in effect
Purpose: Implement pipes, message queues; catch a signal

Description: In this project you will service the requests that were handled in **printReports** in Project 1 via a message queue with a pipe used to support multiple input options. You will write two programs, a server to access the binary data file and a client that takes requests from the user and fulfills them by communicating with the server via a message queue.

The server will set up a message queue. If that is not successful, it will exit. Otherwise, the server will open the binary encoded data file and wait for messages on the queue. When a message for the server arrives, it will be dequeued, the file will be accessed accordingly, and a message will be placed on the queue for the client. Its content will depend upon the nature of the client's request. The server will maintain a log. Each operation will be logged with the client pid and a description of the action taken.

Upon initiation, a client first acquires the id of the message queue that the local server reads from. The call to msgget() to get this id will always use getuid() as the key. If there is no such message queue (return value of msgget() helps determine this), the server is not available (not up yet, not up at all, system error, etc;) and the client should gracefully terminate.

The client program presents a menu to the user. This menu permits lookup of records in the server's file as well as adding new records and updating of pertinent fields. In your chosen data set, choose at least two fields that can be updated. The menu should have options as in the box. Actions are as follows:

```
N)ew Record
D)isplay Record
C)hange Record
S)how Log
```

- New record: input all info for a record, send to server on message queue. Server adds it to the end of the data file.
- Display record: Ask server how many records, receive the count, prompt the user for a value in that range, get the value (error check to be sure it's in range), send the request for that record to the server, receive and display record, well labeled. There MUST be a header that describes the data.in the output
  - Note: If the user enters -999 as the record to display, the entire contents of the data file will be sent to the client for display. You probably want the server to first tell the client how many records to expect.
- Change record: Ask server how many records, receive the count, prompt the user for a value in that range, get the value (error check to be sure it's in range), send the request for that record to the server, receive and display record, well labeled. Provide the user with a menu by which they choose which field to update, input the new value, error check it as appropriate, send data back to server (send the complete record, as other clients may have made requests in the interim). The server overwrites the record in the data file. Be sure to tell it which record for the write back.
- Show log: The server will send the content of the log file to the client, one entry at a time, for display (well-labeled, of course). You probably want the server to first tell the client how many records to expect.

For a new or changed record, the server will send an acknowledgement through the message queue, which will be reported to the user.

Messages in the system must be properly routed. A message to the server will identify it as recipient by the client using a message type value of 1. Messages to clients will have the recipient identified with its pid as the mtype.

NOTES

- The server takes the binary data file name as its command-line argument.
- In the log file each operation will be recorded, along with the pid of the client requesting it.
- Only one record may be in the memory of a client or server process.
- An item passed from the client to the server should contain:
    - the sender's pid
    - the command
        - There are more commands than just the menu options. For example, to display a record, the first transmission gets the number of records. The 2nd transmission asks for a particular record. One suggested setup could include commands to:
            - Obtain the number of records (this is used in two places)
            - Providing a record from the file, by number (also used in two places).
            - Replace a record, by number
            - Others
    - Other necessary text.
- Packets transferred between clients and server should be carefully designed to be predictable in content and size.
- The files mser.cpp and mcli.cpp, under demo/msq on the web and acad, demonstrate use of the message queue.
- Submit a README file. Its content must include:
    - Doxygen link
    - A coherent description of your implementation, including any design decisions that you feel anyone who would read or use your program should be aware of.
    - The specifics of the command setup you implemented in the server, including a chart of each mnemonic and the command it represents.
    - A guide to the various packets used on the message queue.
    - You may submit an (only one) Excel document for the readme. This is not required. You can put the narrative on one sheet and the commands and packet designs on one or two others if that better organizes your presentation of the project.
- It is best if your code is fully portable, as Project 3 may (almost certainly will) be expected to work between machines (acad, a VM csc552, and possibly mcgonagall).

Turnin

All code files plus a makefile. Name your c or cpp files p2cli.c or cpp and p2ser.c or cpp. The executables must be named p2ser and p2cli. The make utility, if invoked without a target, should create both executables.