MybiboBox, A Concurrent File Access System

Description :

Your task is to design and implement a file server that enables multiple clients to connect, access and modify the contents of files in a specific directory

The project should be implemented as a server side and a client side programs

Server side :

biboServer <dirname> <max. #ofClients>

the Server side would enter the specified directory (create dirname if the dirname does not exits), create a log file for the clients and prompt its PID for the clients to connect. The for each client connected will fork a copy of itself in order to serve the specified client (commands are given on the client side). If a kill signal is generated (either by Ctrl-C or from a client side request) Server is expected to display the request, send kill signals to its child processes, ensure the log file is created properly and exit. An example of the Server screen output might be in the following form:

```
> biboServer Here 3

>> Server Started PID 104065…
>> waiting for clients...
>> Client PID 106055 connected as “client01”
>> client01 disconnected..
>> Client PID 106065 connected as “client02”
>> Client PID 106074 connected as “client03”
>> Client PID 106076 connected as “client04”
>> Connection request PID 106088… Que FULL
>> client03 disconnected..
>> Client PID 106088 connected as “client05”
>> kill signal from client05.. terminating...
>> bye
>
```

Client side :

biboClient <Connect/tryConnect> ServerPID

the client program with Connect option request a spot from the Server Que with ServerPID and connects if a spot is available (if not the client should wait until a spot becomes available, tryConnect option leaves without waiting if the Que is full). When connected the client can perform the following requests :

- help
  display the list of possible client requests
- list
  sends a request to display the list of files in Servers directory

(also displays the list received from the Server)

readF <file> <line #>

requests to display the # line of the <file>, if no line number is given
the whole contents of the file is requested (and displayed on the client side)

writeT <file> <line #> <string> :

request to write the content of "string" to the #th line the <file>, if the line # is not given writes to the end of file. If the file does not exists in Servers directory creates and edits the file at the same time

upload <file>

uploads the file from the current working directory of client to the Servers directory (beware of the cases no file in clients current working directory and file with the same name on Servers side)

download <file>

request to receive <file> from Servers directory to client side

quit

Send write request to Server side log file and quits

killServer

Sends a kill request to the Server

An example of the Server screen output might be in the following form:

```
> biboClient connect 104065

        >> Waiting for Que.. Connection established:
        >> Enter comment : help

                Available comments are :

        help, list, readF, writeT, upload, download, quit, killServer

        >> Enter comment : help readF

                readF <file> <line #>
                        display the #th line of the <file>, returns with an
                        error if <file> does not exists

        >> Enter comment : list

                Home.txt
                Work.bin
                ls.txt
                important.exe

        >> Enter comment : upload important.exe
          file transfer request received. Beginning file transfer:
          124354 bytes transferred

        >> Enter comment : quit
          Sending write request to server log file
          waiting for logfile ...
```

```
  logfile write request granted
  bye..
>
```

Requirements:

- Your Server program should be able to handle multiple processes accessing files simultaneously.
- Your program should enforce mutual exclusion to prevent race conditions.
- Your program should ensure data consistency and avoid data corruption.
- Your program should allow for file read and write operations.
- Your program should be able to handle large files (i.e. > 10 MB).
- Your program should be able to handle different file formats (i.e. text, binary).
- Your program should use multiple processes for file access and synchronization.
- Signals should be handled properly on both client and Server sides

Suggested Steps:

1. Design a high-level architecture for your system, including the number of processes and how they will communicate with each other.
2. Implement a file I/O module that can handle read and write operations for different file formats.
3. Implement a synchronization module that provides mutual exclusion and prevents race conditions.
4. Integrate the file I/O and synchronization modules to create a concurrent file access system.
5. Test your system with multiple processes accessing files simultaneously and verify that data consistency is maintained.

Deliverables :

- A document describing your system architecture, design decisions, and implementation details.
- A source code for your concurrent file access system (for the Server and Client) and a makefile to compile the project as a whole.
- A test plan and results demonstrating that your system meets the requirements.

The project requires inter-process communication and multiple synchronization primitives. Implementations lacking these requirements would not be graded.

Best of Luck..