/ CSci4061 F2018 Assignment 2*

* section: 1:00-2:15 lecture*

* Group: 25*

* date: 11/10/18*

* name: Berni D, Xiangyu Zhang, Peigen Luo (for partner)*

* id: 5329383, 5422919, 5418155*

* submit: duong142*

*/

README ASSIGNMENT 2

1. Description:

In this project, we implement a simple version of chat group depending on different terminals(processes). We have a server process and at most 10 users' processes. The server which can see all things going on during all the talks happening between users and server can broadcast messages to all users. They can also see how many users are there and we list all their user_ID out, kick out users and exit the server process which will make all users offline at the same time. In user terminals(processes), users can chat to all other active users, see who's online, chat to a particular user and manually get out of the chat group.

2. Who did what:

Berni:

- Entire programme structure design
- pipe handling
- server command handling - list,kick, exit, other texts,ctrl+c
- server's functions include student's functions
- change structure of get and receive connections
- implement server, child, and user processes
- user command handling - list, exit, p2p, other texts, seg, ctrl+c
- all of error handling
- clean up processes and pipes

- multi-users function
- comment in code
- testing
- segmentation fault (Extra Credits)

Xiangyu:

- Basic server functionality
- Error handling (library calls)

Peigen:

- Server parent and child process basics,
- client process basic
- User command line handling.
- Signal

3. Compile method:

1.Open up a terminal, enter in source file. Firstly, ***make***. Then ***./server***. Then you will see ***<<admin*** in the screen. The server is now running properly.

2. Open up a new terminal, go to the same source file. Type in ***./client xxx***. Here xxx is a random username. Like Tom, Jon or something. Right now, go back to server terminal, you will see ***xxx entered chat group*** shows up on command line.

3. Now basic compile is done. New user can open up other new terminals to join the group, with an upper limit of 10.

4. How to Use:

There are different keywords we can use to play with this group. Let's go through all of them.

In Server:

i. ***\list*** : This command will be used to print the names of all the active users. '<no users>' will be printed if there are no users currently.

ii. ***kick** <**username***> : This command will be used to terminate a particular user's session. For example: \kick abc

iii. ***exit*** : All children should be terminated. The command \exit abc is not accepted as an assumption. In our propram: \exit abc = \exit

iv. <***any-other-text***> : Any other text entered should be sent to all the active users' with the prefix, "Notice:".

In User:

i. ***list*** : Same as above. The list of active users will be generated and sent to the requester and be printed our. It will print '<no users>' if there are no users currently.

ii. ***p2p** <**username***><***message***> : This command is used by a user to send messages to a particular user. For example \p2p abc xyz. Wrong username or msg will be catch by appropriate error msg

iii. ***exit*** :When this command is received, a child process will be killed. The command \exit abc is not accepted as an assumption. In our propram: \exit abc = \exit

iv. <***any-other-text***> : Any other text entered should be sent to all the active users. All the users' processes should print out this text as is (prefixed by the sending user).

EXTRA CREDITS:

- **Ctrl+C in server process**: server terminated -> all users are disconnected by identifying broken pipe -> all users exit.
- **Ctrl+C in an user process**: that user terminated -> server disconnected that user by identifying broken pipe -> that user exit but server work normally, other user work normally.

- **<\seg> in a user process**: that user terminated -> server disconnected that user by identifying broken pipe -> that user exit but server work normally, other user work normally.

5. How program work exactly

Server is the brain of this program, all data handling, decision making, error handling and others happen here. So we should run server first.

Firstly, the server will try to get connection with clients. It will keep on trying until we run client in another terminal, where there is a function called “connect to server” then connection is created and two pipes for parent-child communication will be created as well. Then user can send messages to server, server can also send messages to user. There are several specific command used in both server and user.

For server:

i. ***\list*** : When this command is received, the SERVER creates a string with the names of all the active users and prints them. ’<no users>’ will be printed if there are no users currently.

ii. ***\kick <username>*** : When this command is received, the SERVER should terminate the session for this user by killing its child process. This user should be removed from the user list as well.

iii. ***\exit*** : When this command is received, the SERVER should clean up all of the users, terminate all their processes, and cleanup their pipes and wait for all child processes to terminate. Each child process should cleanup the pipes for a user and exit.

iv. ***<any-other-text>*** : Any other text entered should be sent to all the active users’ pipes with the prefix, “Notice:”. All users’ processes should print out this text as is.

For users:

i. ***\list*** : The list of active users will be generated and sent to the requester via the child process' pipe. The user process will print this list and print '<no users>' if there are no users currently.

ii. ***\p2p <username><message>*** : This command is used by a user to send messages to a particular user. When this command is received, the SERVER should search for the specified user in its user list, extract the message from the command string and send it to the addressed user through a pipe write. An error should be printed in the appropriate window if the username does not exist.

iii. ***\exit*** : When this command is used on a user's process, the SERVER should clean-up that user, remove it from the user list, terminate the child process associated to the user, and cleanup the corresponding pipes.

iv. ***<any-other-text>***: Any other text entered should be sent to all the active users. All the users' processes should print out this text as is (prefixed by the sending user).

NOTICE:

There are few ways to implement exit or kick (it's an open choice for student):
After discussing to professor, it came up 2 simple options that:

1. Send a command to a user, user receives command and clean up itself (closing pipe, then exit). However this way is too simple and we want something in lower level language, behind the screen. Therefore, we choose option 2 for more practicing about sending and receiving signal
2. Send signal KILL or INTERRUPT to user (at lower level). This option require SERVER somehow know USER pid. Therefore, we modified USER structure send user_pid to SERVER. Again, this is for practicing reasons. In the real world, for security reasons, Server is not suppose know User PID.

WE focus on OPTION 2.

6. Explicit assumptions

- There is no user name: SERVER
- Name, msg can only contain alphanumeric characters. Special characters such as n, <, >, & are not allowed.
- You can assume that msg will have at most a single command line
- Real message = <Prefix>: <msg>. By this assumption, we understand that size of real message may bigger MAX_MSG. We still assume real message size = MAX_MSG in our code. To be safe the MSG testing should not exceed **240** characters
- There are only type commands mentioned in description files.
    - For example, only enter \exit. May not handler \exit <any other text>. Otherwise, \exit <any other text> acts like \exit.
    - For example: \exit abc = \exit. User and server may not enter different command's patterns \kick <username>.
    - For example, no error checking for \kick abc abcd. Similar for \exit, \list, \kill <username>. \kick abc abcd = \kick abc
- User will not be more than 10.

7. Error handling

For the error handling parts, we've done the following things to handle potential errors.

1. We check the return value of almost all system calls(except some tedious ones) in our program to check for error conditions. We use many if-else statements to judge if it returns ERROR(-1). If it is, we then perror or fprintf(stderr,...) the error message about that specific functions, and exit after printing the error message.
2. We also handle the exit on the server. After executing the \exit command on the server, the main server process and its child processes (if exist) exit properly, cleaning up all of the users, waiting for all child processes and freeing up any used resources.
3. We handle the invalid user name as well. For example, if we give a wrong name to send /p2p, or if we give a wrong name to /kick, our function will print an error message User not found or No user indicated to show we've handled invalid user name. Also, if we want to add a user who has the same name with an existing one, after it connects to the server, we check the valid name. If name is not valid, we kick it out or disconnect it.

4. There is no” Broken-Pipe” error, every time we read from a pipe, we check if read == 0 to identify a broken pipe. If pipe is Broken => execute an appropriate cleanup process.
5. Every time the cleanup processes are executed, SERVER waits child, identify and close proper pipes and resources. We have been tested really hard and made sure everything is clean before exit.