Network Programming Project 2 - Remote Working Ground (rwg) Server

NP TA

Deadline: Tuesday, 2020/11/10 18:20

1 Introduction

In this project, you are asked to design 3 kinds of servers:

1. Design a **Concurrent connection-oriented** server. This server allows one client connect to it.
2. Design a server of the chat-like systems, called remote working systems (rwg). In this system, users can communicate with other users. You need to use the **single-process concurrent** paradigm to design this server.
3. (Bonus) Design the rwg server using the **concurrent connection-oriented** paradigm with **shared memory**.

These three servers must support all functions in project 1.

2 Scenario of Part One

You can use telnet to connect to your server.
Assume your server is running on nplinux1 and listening at port 7001.

```
bash$ telnet nplinux1.cs.nctu.edu.tw 7001
% ls | cat
bin test.html
% ls |1
% cat
bin test.html
% exit
bash$
```

3 Scenario of Part Two

3.1 Introduction of Requirements

You are asked to design the following features in your server.

1. Pipe between different users. Broadcast message whenever a user pipe is used.
2. Broadcast message of login/logout information.
3. New built-in commands:
   - who: show information of all users.
   - tell: send a message to another user.
   - yell: send a message to all users.
   - name: change your name.
4. All commands in project 1

More details will be defined in chapter 4.

3.2 Scenario

The following is a scenario of using the rwg system.
Assume your server is running on nplinux1 and listening at port 7001.

```
bash$ telnet nplinux1.nctu.edu.tw 7001
****************************************
** Welcome to the information server **
****************************************                    # Welcome message
*** User '(no name)' entered from 140.113.215.62:1201. ***  # Broadcast message of user login
% who
<ID>    <nickname>  <IP:port>           <indicate me>
1       (no name)   140.113.215.62:1201 <- me
% name Jamie
*** User from 140.113.215.62:1201 is named 'Jamie'. ***
% ls
bin test.html
% *** User '(no name)' entered from 140.113.215.63:1013. *** # User 2 logins
who
<ID>    <nickname>  <IP:port>           <indicate me>
1       Jamie       140.113.215.62:1201 <- me
2       (no name)   140.113.215.63:1013
% *** User from 140.113.215.63:1013 is named 'Roger'. ***    # User 2 inputs 'name Roger'
who
<ID>    <nickname>  <IP:port>           <indicate me>
1       Jamie       140.113.215.62:1201 <- me
2       Roger       140.113.215.63:1013
% *** User '(no name)' entered from 140.113.215.64:1302. *** # User 3 logins
who
<ID>    <nickname>  <IP:port>           <indicate me>
1       Jamie       140.113.215.62:1201 <- me
2       Roger       140.113.215.63:1013
3       (no name)   140.113.215.64:1302
% yell Who knows how to make egg fried rice? help me plz!
*** Jamie yelled ***: Who knows how to make egg fried rice? help me plz!
% *** (no name) yelled ***: Sorry, I don't know. :-(         # User 3 yells
*** Roger yelled ***: HAIYAAAAAAA !!!                 # User 2 yells
% tell 2 Plz help me, my friends!
% *** Roger told you ***: Yeah! Let me show you the recipe   # User 2 tells to User 1
*** Roger (#2) just piped 'cat EggFriedRice.txt >1' to Jamie (#1) *** # Broadcast message of user pipe
*** Roger told you ***: You can use 'cat <2' to show it!
cat <5                                                       # mistyping
*** Error: user #5 does not exist yet. ***
% cat <2                                                     # receive from the user pipe
*** Jamie (#1) just received from Roger (#2) by 'cat <2' ***
Ask Uncle Gordon
Season with MSG !!
% tell 2 It's works! Great!
% *** Roger (#2) just piped 'number EggFriedRice.txt >1' to Jamie (#1) ***
*** Roger told you ***: You can receive by your program! Try 'number <2'!
number <2
*** Jamie (#1) just received from Roger (#2) by 'number <2' ***
1 1 Ask Uncle Gorgon
2 2 Season with MSG !!
% tell 2 Cool! You're genius! Thank you!
% *** Roger told you ***: You're welcome!
*** User 'Roger' left. ***
exit
bash$
```

Now, let's see what happened to the second user:

```
bash$ telnet nplinux1.nctu.edu.tw 7001 # The server port number
***************************************
** Welcome to the information server **
***************************************
*** User '(no name)' entered from 140.113.215.63:1013. ***
% name Roger
*** User from 140.113.215.63:1013 is named 'Roger'. ***
% *** User '(no name)' entered from 140.113.215.64:1302. ***
who
<ID>    <nickname>  <IP:port>           <indicate me>
1       Jamie       140.113.215.62:1201
2       Roger       140.113.215.63:1013 <- me
3       (no name)   140.113.215.64:1302
% *** Jamie yelled ***: Who knows how to make egg fried rice? help me plz!
*** (no name) yelled ***: Sorry, I don't know. :-(
yell HAIYAAAAAAA !!!
*** Roger yelled ***: HAIYAAAAAAA !!!
% *** Jamie told you ***: Plz help me, my friends!
tell 1 Yeah! Let me show you the recipe
% cat EggFriedRice.txt >1 # write to the user pipe
*** Roger (#2) just piped 'cat EggFriedRice.txt >1' to Jamie (#1) ***
% tell 1 You can use 'cat <2' to show it!
% *** Jamie (#1) just received from Roger (#2) by 'cat <2' ***
*** Jamie told you ***: It's works! Great!
number EggFriedRice.txt >1
*** Roger (#2) just piped 'number EggFriedRice.txt >1' to Jamie (#1) ***
% tell 1 You can receive by your program! Try 'number <2'!
% *** Jamie (#1) just received from Roger (#2) by 'number <2' ***
*** Jamie told you ***: Cool! You're genius! Thank you!
tell 1 You're welcome!
% exit
bash$
```

4 Spec Details

4.1 Working Directory

```
your_working_directory
    |---- bin
    | |-- cat
    | |-- ls
    | |-- noop
    | |-- number
    | |-- removetag
    |
    |---- user_pipe               # for bonus part
    | |-- (your user pipe files)
    |
    |-- test.html
```

4.2 Format of the Commands

- who:
  Show information of all users.

  Output Format:

  ```
  <ID>[Tab]<nickname>[Tab]<IP:port>[Tab]<indicate me>
  (1st id)[Tab](1st  name)[Tab](1st IP:port)([Tab](<-me))
  (2nd id)[Tab](2nd  name)[Tab](2nd IP:port)([Tab](<-me))
  (3rd id)[Tab](3rd  name)[Tab](3rd IP:port)([Tab](<-me))
  ...
  ```

  Example:

  ```
  % who
  <ID>    <nickname>  <IP:port>   <indicate me>
  1   IamStudent   140.113.215.62:1201 <-me
  2   (no name)   140.113.215.63:1013
  3   student3    140.113.215.62:1201
  ```

  Note that the delimiter of each column is a **Tab**, and the first column represents the login user-id.
  The user's id should be assigned in the range of 1-30.
  Your server should always assign the **smallest unused id** to a new user.
  Example:

  ```
  <new user login> // server assigns this user id = 1
  <new user login> // server assigns this user id = 2
  <user 1 logout>
  <new user login> // server assigns this user id = 1, not 3
  ```

- tell <user id> <message>:
  The user will get the message with following format:

  ```
  *** <sender's name> told you ***: <message>
  ```

  If the receiver of the message doesn't exist, print the following message:

```
    *** Error: user #<user id> does not exist yet. ***
```

Example:

```
        Assume my name is 'IamStudent'.
        [terminal of mine]
        % tell 3 Hello World.
        %

        If user 3 exists,
        [terminal of user id 3]
        % *** IamStudent told you ***: Hello World.

        If user 3 doesn't exist,
        [terminal of mine]
        % tell 3 Hello World.
        *** Error: user #3 does not exist yet. ***
        %
```

- yell <message>:
  Broadcast the message.
  All the users(including yourself) will get the message with the following format:

  ```
      *** <sender's name> yelled ***: <message>
  ```

  Example:

  ```
          Assume my name is 'IamStudent'.
          [terminal of mine]
          % yell Good morning everyone.
          *** IamStudent yelled ***: Good morning everyone.
          %

          [terminal of all other users]
          % *** IamStudent yelled ***: Good morning everyone.
  ```

- name <new name>:
  Change your name by this command. Broadcast the message with the following format:

  ```
      *** User from <IP>:<port> is named '<new name>'. ***
  ```

  Notice that the name CAN NOT be the same as other users' name, or you will get the following message:

  ```
      *** User '<new name>' already exists. ***
  ```

  Example:

  ```
          [terminal of mine]
          % name Mike
          *** User from 140.113.215.62:1201 is named 'Mike'. ***
          %

          [terminal of all other users]
          % *** User from 140.113.215.62:1201 is named 'Mike'. ***
  ```

```
If Mike is on-line, and I want to change name to Mike, this name change will fail.

[terminal of mine]
%  name Mike
*** User 'Mike' already exists. ***
%
```

In all of the test cases, the maximum length of a user's name is **20** characters and will only consists of **alphabet and digits**.

4.3 Login/Logout message

- When a user login, broadcast as follows:

  ```
  *** User '<user name>' entered from <IP>:<port>. ***
  ```

  When a user logout, broadcast as follows:

  ```
  *** User '<user name>' left. ***
  ```

  Example:

  ```
  [terminal of all users]
  *** User '(no name)' entered from 140.113.215.63:1013. *** # user logins
  *** User '(no name)' left. *** # user logouts
  ```

4.4 Welcome Message

```
****************************************
** Welcome to the information server. **
****************************************
```

4.5 User Pipe

1. The formats of using user pipe are '(command) >n' and '(command) <n'. '>n' pipes result(stdout) of command into the pipe, '<n' reads contents from the pipe.

2. Broadcast message when a user pipe is used. When a user writes into user pipe successfully. broadcast as follows:

   ```
   *** <sender_name> (#<sender_id>) just piped '<command>' to <receiver_name> (#<receiver_id>) ***
   ```

   Notice: Command should be whole command line

   ```
   Example:
   % cat <2 | number | number | cat
   *** student1 (#1) just received from student2 (#2) by 'cat <2 | number | number | cat' ***
   ```

   If the pipe already exists, show the following error message:

   ```
   *** Error: the pipe #<sender_id>->#<receiver_id> already exists. ***
   ```

   Whenever a user receives from the user pipe successfully, broadcast as follows:

   ```
   *** <receiver_name> (#<receiver_id>) just received from <sender_name> (#<sender_id>) by '<command>' ***
   ```

If the pipe does not exist, show the following error message:

```
*** Error: the pipe #<sender_id>->#<receiver_id> does not exist yet. ***
```

If the sender or receiver does not exist, show the following error message:

```
*** Error: user #<user_id> does not exist yet. ***
```

Examples:
student1 (#1) pipes a command into student2(#2) via a pipe #1->#2.

```
        user1 login
        user2 login
        % cat test.html >2
        *** student1 (#1) just piped 'cat test.html >2' to student2 (#2) ***
        % cat test.html >2
        *** Error: the pipe #1->#2 already exists. ***
```

student2(#2) can receive from the pipe #1->#2.

```
        % cat <1
        *** student2 (#2) just received from student1 (#1) by 'cat <1' ***
        ...some output... # message from pipe #1->#2.
        % cat <1
        *** Error: the pipe #1->#2 does not exist yet. ***
        % cat <3
        *** Error: user #3 does not exist yet. ***
```

3. '>n' or '<n' has no space between them. So, you can distinct them from ">filename" easily.

4. The following situations will not appear in any test cases, so you don't need to worry about them: (1) output to/input from several pipes (e.g. user pipe, ordinary pipe, number pipe) or file.

```
        % ls >2 | number
        % ls >2 |1
        % ls >2 > aa.txt

        % cat test.html |1      // number pipe to next command
        % cat <2                // input from number pipe and user pipe
```

p.s. the following situations may happen:

```
        % cat <2 | number
        % cat <2 |1
        % cat <2 > a.txt
        % cat <2 >1
        % cat >1 <2
```

5. Show the message of '<n' first, then show the message of '>n'.
Example:

```
% cat <2 >1
*** student3 (#3) just received from student2 (#2) by 'cat <2 >1' ***
*** student3 (#3) just piped 'cat <2 >1' to student1 (#1) ***

% cat >1 <2
*** student3 (#3) just received from student2 (#2) by 'cat >1 <2' ***
*** student3 (#3) just piped 'cat >1 <2' to student1 (#1) ***
```

4.6 Other requirements

1. Initial setting: You have to do this when new client connects.

   - Environment variables: remove all the environment variables except **PATH**.
     Initial value of **PATH** is **bin:.**
     Notice that every client wiil have its own environment variables settings.
     Example:

     ```
     [client A] PATH=.
     [client B] PATH=bin:.
     ```

   - user name: default user name is **(no name)**

2. All behaviors required by project 1 are still required in this project for each user. All commands in project 1 should be working.

5 Specification

1. The maximum number of online users is 30. In other words, there will only be 30 users online simultaneously. However, there will be more than 30 users logged in in some test cases. For example: 30 users login -> 1 user logouts -> 1 user logins ...

2. The message of tell/yell will not exceed 1024 characters.

3. Commands [who], [tell], [yell], [name] are single line commands, which means there will be no pipe connected to these commands, just like [printenv] or [setenv].
   Example:

   ```
   % ls | yell    // It is illegal. This will not appear in testcases.
   % who |2       // It is illegal. This will not appear in testcases.
   ```

4. If someone pipes to you, but you didn't receive the message and disconnected from the server the pipe should be closed. That means a new client who gets your client id can't receive the message because the pipe is gone.

5. You need to take care of the relative position between "% " and Broadcast message.
   "% " will only sent when:

   (a) The client connected to it. In this situation, the order of the messages will be:

   ```
   welcome message -> login Broadcast message -> "% "
   ```

   Example:

```
****************************************
** Welcome to the information server. **
****************************************
*** User '(no name)' entered from 140.113.215.63:1013. ***
%
```

(b) Current command finished. (Except a line end with number pipe)

```
% ls
bin
test.html
%
```

Example:
After 4 users login, the message of user 1 will be:

```
****************************************
** Welcome to the information server. **
****************************************
*** User '(no name)' entered from 140.113.215.63:1013. ***
% *** User '(no name)' entered from 140.113.215.64:1014. ***
*** User '(no name)' entered from 140.113.215.65:1015. ***
*** User '(no name)' entered from 140.113.215.66:1016. ***
```

6. You can only implement this project with C and C++, other third-party libraries are NOT allowed.

7. Your server 1 (single client) should listen for connection again after the user logout. The next user should be able to login after the previous user logout.

8. You should set flag SO_REUSEADDR in server socket.
Hint: use function setsocketopt

9. (Bonus) For the 3rd server (which use share memory), you must use fifo to implement user pipe. The fifo files should be put under directory "user_pipe".

6 Submission

- New e3:
  (a) Create a directory named your **student ID**, put your source code files into the directory. DO NOT upload anything else (e.g. noop, removetag, test.html, **.git**, **__MACOSX**)
  (b) You must provide Makefile.Two executable files named np_simple (server 1), np_single_proc (server 2) should be produced after typing **make** command **in top layer of the directory**.
  (c) All servers should listen on the port assigned by the first argument.

```
Example:
./np_single_proc 12345 # Listen on port 12345
```

  (d) zip the directory and upload the .zip file to new E3.
  Attention !! we only accept .zip format

```
Example:
0856022
    |-- Makefile
    |-- np_simple.cpp       # Server1
    |-- np_single_proc.cpp  # Server2
    |...
```

(e) (Bonus) Same rules as above. One executable files named np_multi_proc should be produced after typing **make**. Demo of bonus part will be hold at the end of this semester.

- Bitbucket:
  Create a private repository with name: ${Your_Student_ID}_np_project2 inside the nctu_np_2020 team, and set the ownership to nctu_np_2020. You can push anything onto bitbucket, but make sure to commit at least 5 times.

  ```
  Example: 0856022_np_project2
  ```

- We will take plagiarism seriously

Enjoy the project !!