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

NP TA

Deadline: Wednesday, 2019/11/20 23:59

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. Design the rwg server using the concurrent connection-oriented paradigm with fifo and shared memory.

These three servers must support all functions in project 1.

2 Scenario of Part One

You can use telnet or the client program we provided to connect to your server.
Assume your server is running on nplinux1 and listening at port 7001.

```
bash$ ./client 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 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

Please refer to section 4 for details.

3.2 Scenario

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

```
bash$ ./client myserver.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 Pikachu
*** User from 140.113.215.62:1201 is named 'Pikachu'. ***
% 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       Pikachu     140.113.215.62:1201  <- me
2       (no name)   140.113.215.63:1013
*** User from 140.113.215.63:1013 is named 'Snorlax'. *** # User 2 inputs 'name Snorlax'
% who
<ID>    <nickname>  <IP:port>            <indicate me>
1       Pikachu     140.113.215.62:1201  <- me
2       Snorlax     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       Pikachu     140.113.215.62:1201  <- me
2       Snorlax     140.113.215.63:1013
3       (no name)   140.113.215.64:1302
% yell Who knows how to do project 2? help me plz!
*** Pikachu yelled ***: Who knows how to do project 2? Help me plz!
*** (no name) yelled ***: Sorry, I don't know. :-(  # User 3 yells
*** Snorlax yelled ***: I know! It's too easy!  # User 2 yells
% tell 2 Plz help me, my friends!
*** Snorlax told you ***: Yeah! Let me show you how to send files to you! # User 2 tells to User 1
*** Snorlax (#2) just piped 'cat test.html >1' to Pikachu (#1) *** *** # Broadcast message of user pipe
*** Snorlax 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
*** Pikachu (#1) just received from Snorlax (#2) by 'cat <2' ***
<!test.html>
<TITLE>Test<TITLE>
<BODY>This is a <b>test</b> program
for rwg.
</BODY>
% tell 2 It's works! Great!
*** Snorlax told you ***: I can send the result of the program to you too!
*** Snorlax (#2) just piped 'removetag0 test.html >1' to Pikachu (#1) ***
*** Snorlax told you ***: You can receive by your program! Try 'number <2'!
% number <2
*** Pikachu (#1) just received from Snorlax (#2) by 'number <2' ***
  1 Error: illegal tag "!test.html"
  2
  3 Test
```

```
  4 This is a test program
  5 for ras.
  6
% tell 2 Cool! You're genious! Thank you!
*** Snorlax told you ***: You're welcome!
*** User 'Snorlax' left. ***
% exit
bash$
```

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

```
bash$ ./client myserver.nctu.edu.tw 7001 # The server port number
****************************************
** Welcome to the information server **
****************************************
*** User '(no name)' entered from 140.113.215.63:1013. ***
% name Snorlax
*** User from 140.113.215.63:1013 is named 'Snorlax'. ***
*** User '(no name)' entered from 140.113.215.64:1302. ***
% who
<ID>    <nickname>  <IP:port>            <indicate me>
1       Pikachu     140.113.215.62:1201
2       Snorlax     140.113.215.63:1013  <- me
3       (no name)   140.113.215.64:1302
*** Pikachu yelled ***: Who knows how to do project 2? help me plz!
*** (no name) yelled ***: Sorry, I don't know. :-(
% yell I know! It's too easy!
*** Snorlax yelled ***: I know! It's too easy!
*** Pikachu told you ***: Plz help me, my friends!
% tell 1 Yeah! Let me show you how to send files to you!
% cat test.html >1 # write to the user pipe
*** Snorlax (#2) just piped 'cat test.html >1' to Pikachu (#1) ***
% tell 1 You can use 'cat <2' to show it!
*** Pikachu (#1) just received from Snorlax (#2) by 'cat <2' ***
*** Pikachu told you ***: It's works! Great!
% tell 1 I can send the result of the program to you too!
% removetag0 test.html >1
*** Snorlax (#2) just piped 'removetag0 test.html >1' to Pikachu (#1) ***
% tell 1 You can receive by your program! Try 'number <2'!
*** Pikachu (#1) just received from Snorlax (#2) by 'number <2' ***
*** Pikachu told you ***: Cool! You're genious! Thank you!
% tell 1 You're welcome!
% exit
bash$
```

4 Spec Details

4.1 Working Directory

```
your_working_directiry
    |---- bin
    | |-- cat
    | |-- ls
    | |-- noop
    | |-- number
    | |-- removetag
    | |-- removetag0
    |
    |---- user_pipe
```

```
| |-- (your user pip 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    140.113.215.64:1302
  ```

  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.
  and 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 does not exist, print the following message:

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

  Example:

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

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

```
If user 3 does not 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 'IamUser'.
  [terminal of mine]
  % yell Hi everybody
  *** IamUser yelled ***: Hi everybody
  %

  [terminal of all other users]
  % *** IamUser yelled ***: Hi everybody
  ```

- name <newname>:
  You can change your name by this command.
  Broadcast the message with the following format:

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

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

  ```
  *** User '<newname>' already exists. ***
  ```

  Example:

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

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

  If Mike is on-line, and I want to have the name "Mike" too.
  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 comes in, broadcast as follows:

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

  When a user leaves, broadcast as follows:

  ```
  *** User '<username>' 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 User pipe

1. The formats of using user pipe are '(command) >n' and '(command) <n'. '>n' pipes message into the pipe, and '<n' reads the message from the pipe.
2. Broadcast message when a user pipe is used. When a user writes into a user pipe successfully, broadcast as follows:

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

   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:

   ```
   *** <sender_name> (#<sender_id>) just received from <receiver_name> (#<receiver_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. ***
   ```

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

   ```
   user 1 login
   user 2 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. ***
   ```

   The user student2 (#2) can use the following to receive from the pipe #1->#2.

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

3. '>n' or '<n' has no spaces 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 several pipes (e.g. user pipe, ordinary pipe, numbered-pipe) or file.

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

ps. the following situations may happen:

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

4.5 Other Requirements

1. Initial setting: You have to do this when new client connects. Environment variables: remove all the environment variables except PATH Example: PATH=bin:.

   !!!!! remove all the environment variables except PATH !!!!!

   Notice that every client wiil have its own environment variables settings Example: [client A] PATH=.

   [client B] PATH=bin:.

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

4. 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.
```

5. 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.

6. 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. ***
```

7. You can only implement this project with C and C++, other third-party libraries are NOT allowed.
8. 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.
9. You should set flag SO_REUSEADDR in server socket.

```
Hint: use function setsockopt
```

6 Submission

- New E3:
  1. 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)
  2. You must provide Makefile. Three executable files named np_simple (server 1), np_single_proc (server 2) and np_multi_proc (server 3) should be produced after typing **make** command **in top layer of the directory**.
  3. All servers should listen on the port assigned by the first argument.

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

  4. zip the directory and upload the .zip file to new E3.

Attention !! we only accept .zip format

```
Example:
0856053
    |-- Makefile
    |-- np_simple.cpp       # Server 1
    |-- np_single_proc.cpp  # Server 2
    |-- np_multi_proc.cpp   # Server 3
    |...
    |...
```

- Bitbucket:
  Create a private repository with name: ${Your_Student_ID}_np_project2 inside the nctu_np_2019 team, and set the ownership to nctu_np_2019. You can push anything onto bitbucket, but make sure to commit at least 5 times.

  ```
  Example: 0856053_np_project2
  ```

- We take plagiarism seriously

Enjoy the project!