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

NP TA

Deadline: Sunday, 2018/11/18 23:59

1 Introduction

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

1. Design a concurrent connection-oriented server. This server allows one client connect to it, and supports all performance in project 1.
2. Design the server of the chat-like systems, called remote working systems (rws). In this system, users can meet with/work with/talk to/make friends with other users. Basically, this system supports all functions, as you did in project 1. In addition, all users can see what all on-line users are working.

   You need to use the single-process concurrent paradigm to design this server.
3. Design the rws server using the concurrent connection-oriented paradigm with shared memory.

2 Scenario of Part One

You can use telnet or the client program we given to connect to your server.
Assume your server is run on nplinux1 and listen 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 an user pipe is used.
2. Broadcast message of login/logout information.
3. New commands:
   - who: show the information of all users
   - tell: send message to another user
   - yell: send 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 run on nplinux1 and listen at port 7001.

```
bash$ ./delayclient nplinux1.cs.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 Apple
*** User from 140.113.215.62/1201 is named 'Apple'. ***
% 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       Apple       140.113.215.62/1201  <- me
2       (no name)   140.113.215.63/1013
*** User from 140.113.215.63/1013 is named 'Banana'. *** # User 2 inputs 'name Banana'
% who
<ID>    <nickname>  <IP/port>            <indicate me>
1       Apple       140.113.215.62/1201  <- me
2       Banana      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       Apple       140.113.215.62/1201  <- me
2       Banana      140.113.215.63/1013
3       (no name)   140.113.215.64/1302
% yell Who knows how to do project 2? help me plz!
*** Apple yelled ***: Who knows how to do project 2? Help me plz!
*** (no name) yelled ***: Sorry, I don't know. :-(  # User 3 yells
*** Banana yelled ***: I know! It's too easy!  # User 2 yells
% tell 2 Plz help me, my friends!
*** Banana told you ***: Yeah! Let me show you how to send files to you! # User 2 tells to User 1
*** Banana (#2) just piped 'cat test.html >1' to Apple (#1) *** # Broadcast message of user pipe
*** Banana told you ***: You can use 'cat <2' to show it!
% cat <5    # mistyping
*** Error: the pipe #5->#2 does not exist yet. ***
% cat <2    # receive from the user pipe
*** Apple (#1) just received from Banana (#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!
*** Banana told you ***: I can send the result of the program to you too!
*** Banana (#2) just piped 'removetag0 test.html >1' to Apple (#1) ***
*** Banana told you ***: You can receive by your program! Try 'number <2'!
% number <2
*** Apple (#1) just received from Banana (#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!
*** Banana told you ***: You're welcome!
*** User 'Banana' left. ***
% exit
*** User 'Apple' left. ***
bash$
```

Now, let us see what happened to the second user.

```
bash$ ./delayclient nplinux1.cs.nctu.edu.tw 7001
****************************************
** Welcome to the information server **
****************************************
*** User '(no name)' entered from 140.113.215.63/1013. ***
% name Banana
*** User from 140.113.215.63/1013 is named 'Banana'. ***
*** User '(no name)' entered from 140.113.215.64/1302. ***
% who
<ID>    <nickname>  <IP/port>           <indicate me>
1       Apple       140.113.215.62/1201
2       Banana      140.113.215.63/1013  <- me
3       (no name)   140.113.215.64/1302
*** Apple 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!
*** Banana yelled ***: I know! It's too easy!
*** Apple told you ***: Plz help me, my friends!
% tell 1 Yeah! Let me show you how to send files to you!
% cat test.html >3 # write to the user pipe
*** Banana (#2) just piped 'cat test.html >1' to Apple (#1) ***
% tell 1 You can use 'cat <2' to show it!
*** Apple (#1) just received from Banana (#2) by 'cat <2' ***
*** Apple told you ***: It's works! Great!
% tell 1 I can send the result of the program to you too!
% removetag0 test.html >3
*** Banana (#2) just piped 'removetag0 test.html >1' to Apple (#1) ***
% tell 1 You can receive by your program! Try 'number <2'!
*** Apple (#1) just received from Banana (#2) by 'number <2' ***
*** Apple told you ***: Cool! You're genious! Thank you!
% tell 1 You're welcome!
% exit
*** User 'Banana' left. ***
bash$
```

4 Spec Details

4.1 Important Setting

- The structure of your working directory:

```
    your_working_dir
    |-----bin
    |  |---cat
    |  |---ls
    |  |---noop
    |  |---number
```

```
| |---removetag
| |---removetag0
|
|-----user_pipe
| |---(save your user pipe file here)
|-----test.html
```

- Your server should allow one parameter <**port**> for setting the port your server listens to.

4.2 Format of the Commands

- who:
  Show the information of all users.
  You have to print a tab between each of tags.
  Notice that the first column does not print socket fd but user id.

  ```
  <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
  ```

  Notice that the user's id should be assigned in the range of number 1-30.
  The server should always assign a smallest unused id to new connected user.

  E.g.,

  ```
  <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 user you want to send message to 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 is exist,
[terminal of user id 3]
% *** IamUser told you ***: Hello World.

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

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

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

  Example:

  ```
  Assume my name is 'IamUser'.
  [terminal of mine]
  % yell Hi everybody

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

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

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

  Notice that the name CAN NOT be the same as the name which on-line users have, or you will get the following message:

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

  Example:

  ```
  [terminal of mine]
  % name IamUser

  [terminal of all 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 changing will be failed.

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

  The length of a user's name is at most 20 characters.
  Also, there are only alphabet and digits in names.

4.3 Login/logout message

- Whenever a user comes in, broadcast as follows:

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

  Whenever 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 whenever the user pipe is used. Whenever a user write into the user pipe successfully, broadcast as follows:

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

   If the pipe exists already, 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 exists, 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 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 <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 the test case, so you don't need to worry about it: (1) output to several pipe (e.g. user pipe, ordinary pipe, numbered-pipe) or file.

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

   ps. the following situations can be happened:

   ```
   % 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 e.g., PATH=bin:.

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

   Notice that every client wiil have its own environment variables settings.
   E.g.

   ```
   [client A]
   % setenv PATH .
   % printenv PATH
   .

   [client B]
   % printenv 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 be 30 users online simultaneously. However, there will be more than 30 users login totally 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 server of 3rd server (which using shared memory): For more simple implementation, we recommend to use file as user pipe.
   In other words, whenever using user pipe, input from / output to a file.
   The file should be put under the directory "user_pipe".

   For the server of 2rd server (single process server), use pipe to implement the 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].

   E.g.

   ```
   % 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 pipe to you, but you don't receive the message and disconnect from the server the pipe should be closed. that means next time someone connects to server, he gets the same client id, but he can't get the message because the pipe is gone.

6. You need to care about the related position between "% " and Broadcast message.
   "% " will only sent when:

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

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

   E.g.

   ```
   ****************************************
   ** 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. ***
   ```

5.1 About Submission

1. E3

   (a) Create a directory named your student ID, put your files in the **same directory layer**.

   (b) You should name your server program into:

   - 1st server: np_simple
   - 2nd server: np_single_proc

- 3rd server: np_multi_proc

(c) You must provide a **Makefile**, which compiles your source code into 3 servers. The servers should be under the same directory as the source codes. We will use these servers for demo.

(d) Upload **only** your code and Makefile.
Do not upload anything else (e.g. **removetag**, **noop**, **test.html**, **ls**...)

(e) **zip** the directory and upload the .zip file to the E3 platform
Attention!! we only accept .zip format

```
e.g.
Create a directory 0756000, the directory structure may be:
0756000
    |-- Makefile
    |-- server_simple.cpp
    |-- server_simple.h
    |...
    |...
zip the folder 0756000 into 0756000.zip, and upload 0756000.zip onto E3
```

2. Bitbucket:

   (a) Create a **private** repository: ${your_student_ID}_np_project2 inside the **nctu_np_2018** team, under **np_project2**. Set the ownership to **nctu_np_2018**

   ```
   e.g. 0756000_np_project2
   ```

   (b) For each project, you need to commit on bitbucket for **at least 5 times**.

   (c) You can push anything you need onto bitbucket (including **removetag**, **noop**, **test.html**...), as long as the size of the file is reasonable.

3. **We take plagiarism seriously.**

```
All projects will be checked by a cutting-edge plagiarism detector.
You will get zero points on this project for plagiarism.
Please don't copy-paste any code from the internet, this may be
considered plagiarism as well.
Protect your code from being stolen.
```

5.2 Notes

1. NP project should be run on NP servers (to be announced), otherwise, your account may be locked.
2. Any abuse of NP server will be recorded.
3. Don't leave any zombie processes and un-used shared memory in the system.
4. You will lose points for violating any of the rules mentioned in this spec.
5. Enjoy the project!

If you find some commands confusing or not workable, please let us know.

Due date: 11/18 (Sunday)