Homework #5

Fall 2015

Due date: 2015/12/27

Concurrent Mail server

In this homework, we are trying to build a TCP mail server to serve clients through command line.

Requirements

- **Basic version**
  - Write a concurrent TCP server to handle all connections from clients.
  - For communication, you would receive messages from client and messages are **Strings**.
    - In basic version, all commands, options and arguments are divided by 1 whitespace.
    - E.g. “init –u xxx”, “ls –l”, “rm –r xx” and so on.
  - Commands
    - After a client connects to the server, he can send following commands to server.
    - **Free commands**
      - These commands could be used any situation without registering mail address.
      - exit
        - When server receive this command, it should clean the information of this client and send back: “exit\n”.
          Finally, close the connection between server and this client.
      - init –u <account_name>
        - User wants to register a mail address by < account_name > he gave.
        - < account_name > is a string that only be legal in alphabets, digits, ’_’ and ‘-’.
        - The< account_name > could not be duplicate.
          If two user use same <username>, the latter would be invalid. And you need to return a message:

        "This account has been registered\n"
    - If successful, return the mail address back:
      "<user_name>@nctu.edu.tw\n"
- ls –u
    - list all registered accounts in the server
    - If there are registered accounts, return their mail addresses back.
      "studnet1@nctu.edu.tw\n
      studnet2@nctu.edu.tw\n"
    - If there is no registered account, return back:
      "no accounts\n"

◆ **Constrained commands**

- These commands would work after registering mail address.
    - If this client hasn't registered, return back: "init first\n"
- ls –l
    - List mails this accounts received.
        - If there are mails, return their indexes and titles back.
        - If a mail account hasn't read, adding "(new)" to the end of mail title.
          "1. Mail1\n
            2. Mail2 (new)\n
            3. Mail3\n"
        - If there is no mail, return back: "no mail\n"
- ls –a
    - Show the information about this account
    - Return back its account name, mail address and number of mails.
      "Account: Jack\n
      Mail address: Jack@nctu.edu.tw\n
      Number of mails: 0\n"
- rm –d <N: number of index>
    - Delete N-th mail in the mail box.
    - N is a number, only be legal in digits.
    - If N-th mail doesn't exist, return back: "args error\n"
    - If Successful, return back: "done\n"
- rm –D
    - Delete all mails in the mail box
    - If there is no mail in mail box, it also works.

            - Return back: “done\n”
        - rd –r <N: number of index>
            - Read N-th mail in the mail box.
            - N is a number, only be legal in digits.
            - If N-th mail doesn’t exist, return back: ”args error\n”
            - If Successful, return back in following format:
              “From: <sender>\n
                To: <receiver>\n
                Date: <sending time>\n
                Title: <title>\n
                Content: <content>\n”
        - wt –d <receiver> -t <title> -c <content>
            - write a mail to <receiver>
            - In basic version, the order of these options are fixed.
            - <receiver>, <title> and <content> are string, only be legal in in alphabets, digits, ’_’, ‘-’, ‘:’, ’.’ and ‘@’.
            - If Successful, return back: “done\n”
    - **Trouble shooting**
        - In concept, when server receive a message, it should check command first. And then, check option tag. Finally, check arguments of each options.
        - If the command is illegal, return back : “command error\n”
        - If the option is illegal, return back : “option error\n”
        - If the arguments is illegal, return back : args error\n”

- **Advanced version**
    - In advanced version, there are some different rules to basic version:
    - **Free whitespace**
        - In basic version, all commands, options and arguments are divided by 1 whitespace. But in advanced version, the numbers of whitespaces will be random (at least 1).
            - E.g. “   ls   -l ”, “ init –u   Jack”
    - **Arguments with double quotation**
        - In basic version, the whitespace is illegal in Arguments.
          But if we use double quotation to embrace this argument, the whitespace would be legal.
            - E.g. “wt -d \”jack@nctu.edu.tw\” –t \”Hello\” –c \“Nice to see you.\”   ”

    - Notice that, if the argument is a number, you shouldn’t use double quotation to embrace it.
        - E.g. “rd –r \”2\” ” => “args error\n”
- **Reply and Forward**
    - Add two commands in your server.
    - re –c <content> -n <N: number of index>
        - Reply N-th mail back with content <content>.
        - <content> is a string, same as rule as wt’s <content>.
        - N is a number, only be legal in digits.
        - If other accounts receive a reply mail, the mail should add “re:” at begin of mail title.
            - E.g. “ls –l” => “1. re:XXXX\n”
              E.g. ”rd –r 1” => “…..\n   Title: re:XXXX\n….”
    - fwd –d <receiver> –c <content> -n <N: number of index>
        - Forward N-th mail to <receiver>.
        - <content> and <receiver> are strings, same as rule as wt’s.
        - N is a number, only be legal in digits.
        - If other accounts receive a forward mail, the mail should add “fwd:” at begin of mail title.
            - E.g. “ls –l” => “1. fwd:XXXX\n”
              E.g. ”rd –r 1” => “…..\n   Title: fwd:XXXX\n….”
- **Free orders of arguments**
    - In basic version, the orders of arguments in wt are fixed. In advanced version, the orders would be free.
        - E.g. wt -d –t –c , wt –d –c –d and so on.
    - Same as re and fwd.
        - E.g. re –c –n, re –n –c , fwd –c –d -n

Demo

- Run basic version correctly. (50%)
- Free whitespace. (6%)
- Arguments with double quotation (6%)
- Free orders of arguments(8%)
- Reply and Forward (10%)
- Oral defense (20%)

Note

1. We will provide a TCP client and sample inputs. Also, you could use “telnet” to connect your server for early development.
2. You could use any programming language to write your own code.