README FILE

NAME: Tianyuan Chen

USC ID: 1131431070

1. What you have done in the assignment.

I write 4 *.c files to finish this project and fulfill all the requirements described in the description file. And I have tried different input files and they all works well.

2. What your code files are and each one of them does.

- **healthcenterserver.c:** This piece of codes handles the duties of health center server during phase 1-2.

In phase 1: (1) Health center server creates a TCP socket and listen in port numbered "21070". (2) When the health center server first boots, it reads the contents of "users.txt" and store the information in its memory. (3) Receive "authenticate patient password" from patient and compare with information in "users.txt". (4) If there is a match for both username and password, then health center server sends "success" to patient. If not, sends "failure".

In phase 2: (1) Health center server receives message "available" from patient, and send the list of available appointment to patient (server should always record all the available appointments). (2) Receive "selection #" from patient and compare whether this appointment has been reserved, if yes, send back "notavailable", if not, mark this appointment as reserved and send back corresponding doc name and port number. (3) Close socket and quit.

- **patient1/2.c:**

In phase 1: (1) Patient reads "patient1/2.txt", and stores all information in memory. (2) Creates a TCP socket and connect to the port that server is listening to. (3) Send a request to server with the format "authenticate username password". (4) Upon receive "success", patient will continue enter phase 2, upon receive "failure", patient will terminate the connection and exit.

In phase 2: (1) Patient sends "available" to server. (2) Receive available

appointment from server and print on screen. (3) User will enter a wanted appointment and patient will check whether it matches with anyone of the time indices displayed, if not, patient will ask user to input again. If yes, send "selection #" to server. (4) Receive message from server, if it is the doc name and port number, record it and terminate the TCP socket, if it is "notavailable", terminate the TCP socket and the program.

In phase 3: (1) Patient creates a UDP socket and reads "patient#insurance.txt". (2) Sends "insurance#" to doctor with the port number gained from phase 2.

- **doctor.c:**

2 doctors are under 1 main function using fork(). (1) Each doctor creates a UDP connection, and reads file "doc#.txt" and stores it into memory. (2) upon receive the insurance type of one patient, doctor checks its memory and sends back corresponding cost.

NOTE: There is an additional TXT file named "availnum.txt" (the function of this TXT file will be detailed described in 7.1.

3. What the TAs should do to run your programs?

Command:

make

(open 4 parallel windows connecting to nunki, and run each of following commands orderly in each window.)

./healthcenterserver

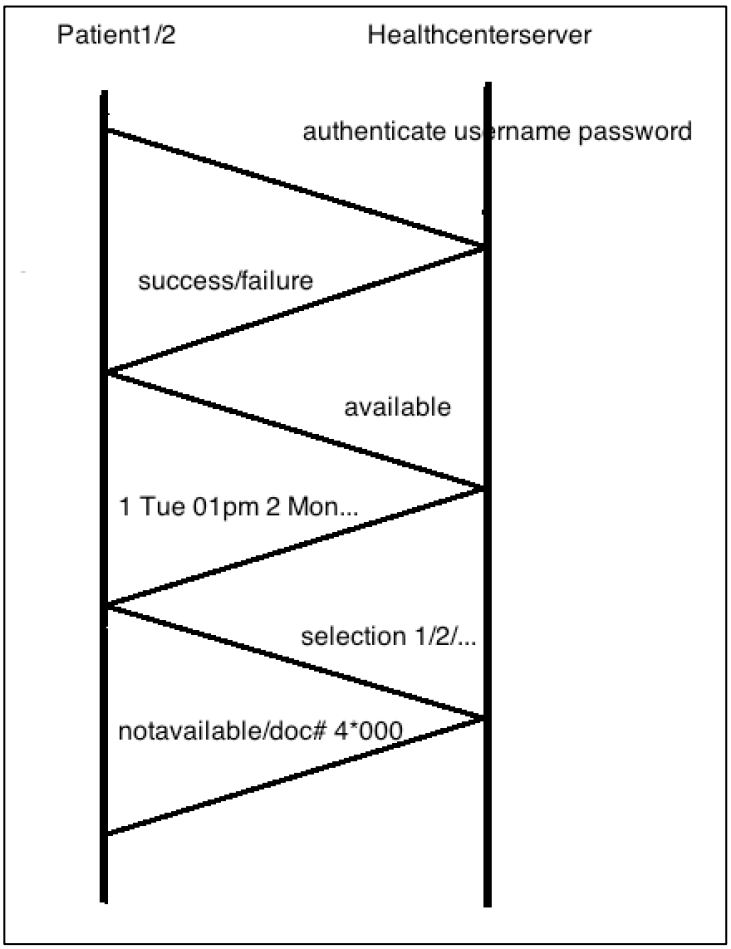
./doctor

./patient1 (should choose the proper appointment)

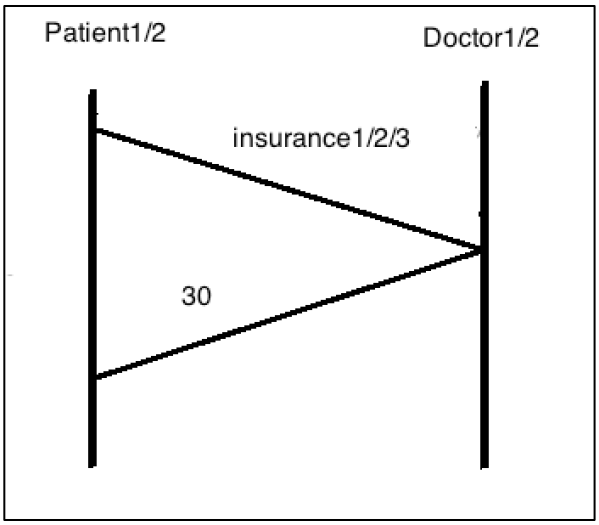
./patient2 (should choose the proper appointment)

4. The format of all the messages exchanged.

4.1. Server v.s. Patient



4.2. Patient v.s. Doctor



4.3. Patient1v.s. Patient2

2 patients are in different child processes, they exchange message through txt file, patient1 write information into “availnum.txt”, and when patient2 want to use it, simply read it out.

5. Idiosyncrasy

The whole project fulfills all the requirements in the description, but may fail in some further situation.

For example, due to the format that I exchange information between different child processes, the server only support 2 patient connects to it. More connection will lead to wrong record of the already reserved appointment. (but with small mediation will easily solve the problem).

6. Reused code (from Beej’s Book)

(1) Blocks of codes of building TCP and UDP connections and exchanging information. (Including functions like getaddrinfo(), socket(), bind(), connect(), listen(), accept() and so on).

(2) Functions for getting IP and port number like getsockname(), getpeername(), gethostbyname().

(3) Function for reaping all dead progress.

7. Some important details

7.1. To exchange information about the appoint time used between different child progress.

Create a file named “availnum.txt”. The file at first is empty (maybe “ “? Or a number is not 0-5)(In the father progress, should first make this file into this format, maybe fprintf 7?). Child progress opens the file, and writes the number of appointment it takes into the file. (In this progress described, only 2 progresses are supported.)

Every child progress should read the file, read the number into a variable (maybe reservednum?), compare with the availabilities[i][0](0<=i<=5) during strcat, if it matches,

discard this line. Then send the form to the patient.

Patient receive the form, then read them into variables and print (since the number of variables are not constant, how to read and print?)

Patient then choose a number available send to server. At the patient side, the number should be checked first. If it is an available number (compare), then OK, send it to the server side. If not, ask the patient enter again.

Once the server side receives the number, print it into the file (means the spot is reserved).

Over.

NOTE: Every time after the all progresses from phase 1 to 3, healthcenterserver should be restarted to initial the availnum.txt.

7.2. Further improvement (use if…else…frame in server) :

The parent progress is listening to the socket.

And when a patient connects to this socket, the parent progress creates a child progress.

Then the child progress check the information the patient send to it, if:

(1) It is "authenticate username password", and then the phase 1 goes on. If authenticate failed, then terminate the connection and exit. If authenticate successfully, then the patient also terminate the connection, but continue to do phase 2. But at the server side, the IP and port addresses are recorded. For example in usersinuse[2][2][20].

(2) It is "available", and then the phase 2 goes on. The server first check the usersinuse[2][2][20] with the IP address and port number. If it is in it, then continue. If not, terminate. (Is this progress needed?)