README

Developed by Yatin Wadhawan
USC ID : 1139340247
Session # 2
EE450 : Introduction to Computer Networks
Socket Programming Project : Using Multithreading and Process Synchronization

/*NOTE : Socket programming part of code is referred from http://beej.us/guide/bgnet/output/print/bgnet_USLetter.pdf for the purpose of learning socket programming ONLY. The socket code snippet has been modified according to the specifications of the project. It describes the APIs for creating TCP and UDP packets in more detailed manner.*/

There are three folders present in the project folder which I have submitted. The name of the folders are **heathserver**, **patient** and **doctor**. I will describe contents of the each folder in subsequent sections.

HealthServer Folder

It contains files :

1. availabilityserver.h
2. healthcenterserver.c
3. users.txt
4. availabilities.txt
5. Makefile

healthcenterserver.c contains the logic of server. It contains a “struct” of users in order to encapsulate the information of patients. It creates the linked list and has two parameters username and password as char arrays. It has following functions in order to create the linked list of users :

1. void insertUser(char username[], char password[]) - Insert user in the linked list
2. int isPresent(char username[], char password[]) - to authenticate user
3. void displayUsersLinkedList() - to display list of username and passwords of users

I have created the linked list so that it becomes easy to increase the number of patients in future. This file contains two other functions

1. **void loadAvailabilityList(const char *file) -** to load the doctor availability list from the file
2. **void loadUsersDataFromFile(const char *file) -** to load the user data from the file in linked list.

In main function of this file, I have created the TCP socket for the server at port 21247 which is constant and will b used by the patients. After creating socket, infinite while is running in order to accept client connections at given IP address and port 21247. When client request for the connection, server creates a new child socket since it is a TCP socket and **fork** a new process

for that child. Once that child is created server is keep listening on the port 21247 and simultaneously child calls a function **void proccessing(int newSockId)** which contains the child socket as parameter. In processing, server receives the client authentication request and it checks in its linked list created to store the client records to validate the client. After validation, client asks for the availability lis of doctors. Server checks its availability list and send that to the client. Then server receives the request from the client regarding a particular doctor schedule and server checks its availability in its database. If it is available it confirms otherwise the rejects the connection and close the child socket. It contains two functions in order to authenticate the users and send time slots to the client:

1. char* authentication(char buf[], int newSockId,int *flag)
2. int sendTimeSlots(int newSockId, char *user)

To run server, you need to go to the directory where you have copied health server folder and run the make file in the folder. The following snapshot tells you to clean the make file : "**make clean"**, how to compile using make file : **"make command"** and how to run server : **./ healthcenterserver availabilities.txt users.txt.** Now Server is in passive listening mode.

```
Yatins-MacBook-Pro:healthserver yatinwadhawan$ make clean
rm -f *.o healthcenterserver
Yatins-MacBook-Pro:healthserver yatinwadhawan$ make
cc    -c -o healthcenterserver.o healthcenterserver.c
gcc -o healthcenterserver -g -lm healthcenterserver.o
Yatins-MacBook-Pro:healthserver yatinwadhawan$ ./healthcenterserver availabilities.txt users.txt
Phase 1: The Health Center Server has port number 21247 and IP  address 10.123.38.123
```

availabilityserver.h contains the logic of creation of linked list for availability list. I have created **"struct avail"** which has parameters index, date, time, port, docid, isavailable. In order to synchronize the availability list, since it is used by both the processes, I have used **pthread mutex** of **POSIX** library. Also in order to allocate memory to linked list I have used mmap which is used to create the shared memory for fork child processes. Using variable **isAvailable** it maintains the availability of the appointment. I have implemented the **signaling** in the code so that when user press **Ctrl + C** it will detect it and close all the ports. I have used **sigset** and a **hander** for this signal/

Points to be noted :

You need to add files availabilities.txt and users.txt in the folder health server so that to change your input to the server. And same syntax will be used to compile and run the program. If you change the file name only then you need to change the name of the file. See the Snapshots section in order to see the basic operation.

“Code works fine and performs all the phases accordingly. It prints the statement in stdout appropriately. If you want to stop server press Ctrl + C. It handle the closing of all ports.“

Patient Folder

It contains files :

1. patient.c
2. availability.h
3. patient1.txt
4. patient2.txt
5. patient1insurance.txt
6. patient2insurance.txt
7. Makefile

patient.c file has the logic of creating TCP and UDP in order to connect the server and doctor respectively. It contains the following functions :

1. struct users* loadUsernamePass(char *file); - to load patient username and password from the files

2. void loadUserNameAndStart(struct file *p); - it calls the above functions and function below it for both the patients.

3. void createTCPSocketAuthenticate(char *auth, struct users *pat, char *, int *,int,int *); - it is responsible for creating the TCP socket and interacting to the server so that to authenticate themselves, ask for availability list and fix the appointment from the server.

4. void createUDPSocketDoctor(struct users *pat,char doc[], int port, int, char []); - it create the UDP socket and connect to the doctor which patient has fixed in the phase 2. Once it checks for the appointment and receive the cost estimation from the doctor, it will close the socket and exit from the program.

5. void handler(int signo) - It is used when user press the Ctrl + C, it will called. Signalling is implemented so that to close all open sockets in the program.

This snapshots shows how to compile the code of patient. You need to go to patient folder and run corresponding commands :

make clean
make
./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt

```
Yatins-MacBook-Pro:patient yatinwadhawan$ make clean
rm -f *.o patient
Yatins-MacBook-Pro:patient yatinwadhawan$ make
cc    -c -o patient.o patient.c
gcc -o patient -g -lm patient.o
Yatins-MacBook-Pro:patient yatinwadhawan$ ./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt
Phase 1: Patient 2 has TCP port number 59842 and IP address 10.123.38.123
Phase 1: Patient 1 has TCP port number 59841 and IP address 10.123.38.123
```

Features Added:

1. I have used multithreading in order to create two patients threads and used POSIX library pthread mutex to synchronize the variables which are shared by the patient threads. In main, two threads are created and call the start function **loadUserNameAndStart.**

2. When all time slots are used by the patients, server will send **"no"** field so that patient knows the availability of list. If there are no slots, program displays **Not available timeslots.**

3. If patient is not able to authenticate properly, no further statements of phase 2 and phase 3 are shown for that patient.

4. Once the phase 3 of patient is finished, you need to restart the patient program in order to request more appointments.

Points to be noted :

Cases

1. There is a scenario when patient 1 has already fixed the appointment but still in the patient 2 list it shows that appointment. It is because patient 1 and patient 2 thread call for the list simultaneously, it depends on the timing that which asks for an input from the user first. Thats why it shows the selected appointment but if select it will say that appointment is already selected and close the patient socket.
2. Do not give input another than numbers in booking the appointments otherwise it will fail and go in infinite loop. It is because I am assuming users will give numbers as input.
3. User should give the same input of insurance as it is present in both patient#insurance.txt and in doctors insurance list. otherwise program may fail.

You need to add files patient1.txt, patient2.txt, patient1insurance.txt and patient2.insurance.txt in the folder patient server so that to change your input to the server. And same syntax will be used to compile and run the program. If you change the file name only then you need to change the name of the file. See the Snapshots section in order to see the basic operation.

"Code works fine and performs all the phases accordingly. It prints the statement in stdout appropriately. If you want to stop server press Ctrl + C. It handles the closing of all ports."

Doctor Folder

It contains files :

1. doctor.c
2. doc1.txt
3. doc2.txt
4. Makefile

doctor.c contains the logic that how doctor is listening on the designated UDP port. It creates the **struct insurance** so that to create the linked list of insurance. It contains two parameters name and price of the insurance. The following functions are used to create the insurance linked list :

1. struct insurance* insertInsurance(struct insurance *insu, char name[], long price) - it is used to add the new node of the linked list as an insurance
2. void display(struct insurance *insu) - It is used to display the list of insurance corresponding to the doctor.
3. long getPriceOfInsurance(struct insurance *insu, char name[]) - It retrieves the cost of the insurance by providing the name of the insurance.
4. struct insurance * loadInsuranceData(struct insurance *insu, char *file) - it creates the linked list by parsing the file containing all insurance and store them in linked list.

In main function, I have created pthread library to create two doctor threads running on **41427** and **42247. T**hese threads are listening to the patient request on these ports. They serve the request of patient by looking on the linked list of insurance they have created and send the cost of that insurance. And keep listening to the patient's request.
I have used signaling here also, so that when user breaks the program using Ctrl + C , the signal handler will close all the ports opened.
This snapshots shows how to compile the code of doctor. You need to go to patient folder and run corresponding commands :

```
Yatins-MacBook-Pro:doctor yatinwadhawan$ make clean
rm -f *.o doctor
Yatins-MacBook-Pro:doctor yatinwadhawan$ make
cc    -c -o doctor.o doctor.c
gcc -o doctor -g -lm doctor.o
Yatins-MacBook-Pro:doctor yatinwadhawan$ ./doctor doc1.txt doc2.txt
Phase 3: Doctor 2 has a static UDP port 42247 and IP address 10.123.38.123.
Phase 3: Doctor 1 has a static UDP port 41247 and IP address 10.123.38.123.
```

make clean
make
./doctor doc1.txt doc2.txt

Makefile has the commands to compile the files and clean them if required.

Points to be noted :

You need to add files doc1.txt and doc2.txt in the folder doctor server so that to change your input to the doctor server. And same syntax will be used to compile and run the program. If you change the file name only then you need to change the name of the file. See the Snapshots section in order to see the basic operation.

"Code works fine and performs all the phases accordingly. It prints the statement in stdout appropriately. If you want to stop server press Ctrl + C. It handle the closing of all ports. It will print multiple request of the patients and responses."

Others

I have taken modular approach to write code. It is flexible and scalable since I have used multithreading which makes my code clean and reusable. Synchronization is also implemented in order to avoid the race condition using pthread_mutex_lock/unlock. I have used
#include <pthread.h>
#include <signal.h>
libraries in order to achieve multi threading and signaling.

```
nunki.usc.edu(39): ./healthcenterserver availabilities.txt users.txt
Phase 1: The Health Center Server has port number 21247 and IP  address 68.181.201.3
Phase 1: The Health Center Server has reveived request from a patient with username patient2 and password password222.
Phase 1: The Health Center Server sends the response success to the patient with username patient2.
Phase 1: The Health Center Server has reveived request from a patient with username patient1 and password password111.
Phase 1: The Health Center Server sends the response success to the patient with username patient1.
Phase 2: The Health Center Server receives a request for available timeslots from patients with port 34782 and IP address 68.181.201.3
Phase 2: The Health Center Server sends available time slots to patient with username patient2
Phase 2: The Health Center Server receives a request for available timeslots from patients with port 34781 and IP address 68.181.201.3
Phase 2: The Health Center Server sends available time slots to patient with username patient1
Phase 2: The Health Center Server receives a request for appointment from patient with port number 34782 and username patient2
Phase 2: The Health Center Server confirms the following appointment 1 to patient with username patient2
Phase 2: The Health Center Server receives a request for appointment from patient with port number 34781 and username patient1
Phase 2: The Health Center Server confirms the following appointment 2 to patient with username patient1
```

Server Snapshot : Server is waiting for the further request.

```
nunki.usc.edu(45): ./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt
Phase 1: Patient 2 has TCP port number 34782 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 2 with username patient2 and password password222 has been sent to the Health Center Server.
Phase 1: Patient 1 has TCP port number 34781 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 1 with username patient1 and password password111 has been sent to the Health Center Server.
Phase 1: Patient 2 authentication result: success.
End of Phase 1 for Patient 2.
Phase 1: Patient 1 authentication result: success.
End of Phase 1 for Patient 1.
Phase 2: The following appointments are available for Patient 2:

1 Tue 01pm
2 Mon 03pm
3 Thu 02pm
4 Wed 10am
5 Sat 12pm

Please enter the preferred appointment index and press enter:
1
Phase 2: The requested appointment is available and reserved to Patient 2. The assigned doctor port number is 41247
Phase 2: The following appointments are available for Patient 1:

1 Tue 01pm
2 Mon 03pm
3 Thu 02pm
4 Wed 10am
5 Sat 12pm

Please enter the preferred appointment index and press enter:
2
Phase 3: Patient 2 has a dynamic UDP port number 62210 and IP address 68.181.201.3.
Phase 3: The cost estimation request from Patient 2 with insurance plan insurance1 has been sent to the doctor with port number 41247 and IP address 68.181.201.3.
Phase 2: The requested appointment is available and reserved to Patient 1. The assigned doctor port number is 42247
Phase 3: Patient 1 has a dynamic UDP port number 62212 and IP address 68.181.201.3.
Phase 3: The cost estimation request from Patient 1 with insurance plan insurance2 has been sent to the doctor with port number 42247 and IP address 68.181.201.3.
Phase 3: Patient 2 receives 30$ estimation cost from doctor with port number 41247 amd name Doctor 1
Phase 3: End of Phase 3 for Patient 2.
Phase 3: Patient 1 receives 50$ estimation cost from doctor with port number 42247 amd name Doctor 2
Phase 3: End of Phase 3 for Patient 1.
nunki.usc.edu(46):
```

Patient Snapshot : patients are finished with process.

```
nunki.usc.edu(21): ./doctor doc1.txt doc2.txt
Phase 3: Doctor 1 has a static UDP port 41247 and IP address 68.181.201.3.
Phase 3: Doctor 2 has a static UDP port 42247 and IP address 68.181.201.3.
Phase 3: Doctor 1 receives the request from the patient with the port number 62210 and name  with the insurance plan insurance1
Phase 3: Doctor 1 has sent estimated price 30$ to patient with port number 62210
Phase 3: Doctor 2 receives the request from the patient with the port number 62212 and name  with the insurance plan insurance2
Phase 3: Doctor 2 has sent estimated price 50$ to patient with port number 62212
```

Doctor Snapshot: doctor is waiting for the further request.

```
nunki.usc.edu(46): ./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt
Phase 1: Patient 1 has TCP port number 34818 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 1 with username patient1 and password password111 has been sent to the Health Center Server.
Phase 1: Patient 2 has TCP port number 34819 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 2 with username patient2 and password password222 has been sent to the Health Center Server.
Phase 1: Patient 1 authentication result: success.
End of Phase 1 for Patient 1.
Phase 1: Patient 2 authentication result: success.
End of Phase 1 for Patient 2.
Phase 2: The following appointments are available for Patient 2:

3 Thu 02pm
4 Wed 10am
5 Sat 12pm

Please enter the preferred appointment index and press enter:
3
Phase 2: The requested appointment is available and reserved to Patient 2. The assigned doctor port number is 41247
Phase 2: The following appointments are available for Patient 1:

3 Thu 02pm
4 Wed 10am
5 Sat 12pm

Please enter the preferred appointment index and press enter:
3Phase 3: Patient 2 has a dynamic UDP port number 62264 and IP address 68.181.201.3.
Phase 3: The cost estimation request from Patient 2 with insurance plan insurance1 has been sent to the doctor with port number 41247 and IP address 68.181.201.3.

Phase 2: The requested appointment from patient 1 is not available. Exiting...
Phase 3: Patient 2 receives 30$ estimation cost from doctor with port number 41247 amd name Doctor 1
Phase 3: End of Phase 3 for Patient 2.
nunki.usc.edu(47):
```

Patient Snapshot: Patient 2 has taken appointment 3 and when Patient 1 is asking for it, it is not available.

```
nunki.usc.edu(48): ./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt
Phase 1: Patient 2 has TCP port number 34877 and IP address 68.181.201.3
Phase 1: Patient 1 has TCP port number 34876 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 2 with username patient2 and password password222 has been sent to the Health Center Server.
Phase 1: Authentication request from Patient 1 with username patient1 and password password111 has been sent to the Health Center Server.
Phase 1: Patient 1 authentication result: success.
End of Phase 1 for Patient 1.
Phase 1: Patient 2 authentication result: success.
End of Phase 1 for Patient 2.
Not Available Time slots.
nunki.usc.edu(49):
```

Patient Snapshot: When no time slots are available for giving appointment to patients. We need to restart the server.

```
nunki.usc.edu(49): ./patient patient1.txt patient1insurance.txt patient2.txt patient2insurance.txt
Phase 1: Patient 1 has TCP port number 34912 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 1 with username patient1 and password password232 has been sent to the Health Center Server.
Phase 1: Patient 2 has TCP port number 34913 and IP address 68.181.201.3
Phase 1: Authentication request from Patient 2 with username patient2 and password password222 has been sent to the Health Center Server.
Phase 1: Patient 1 authentication result: failure.
End of Phase 1 for Patient 1.
Phase 1: Patient 2 authentication result: success.
End of Phase 1 for Patient 2.
Phase 2: The following appointments are available for Patient 2:

1 Tue 01pm
2 Mon 03pm
3 Thu 02pm
4 Wed 10am
5 Sat 12pm

Please enter the preferred appointment index and press enter:
1
Phase 2: The requested appointment is available and reserved to Patient 2. The assigned doctor port number is 41247
Phase 3: Patient 2 has a dynamic UDP port number 62344 and IP address 68.181.201.3.
Phase 3: The cost estimation request from Patient 2 with insurance plan insurance1 has been sent to the doctor with port number 41247 and IP address 68.181.201.3.
Phase 3: Patient 2 receives 30$ estimation cost from doctor with port number 41247 amd name Doctor 1
Phase 3: End of Phase 3 for Patient 2.
nunki.usc.edu(50):
```

Patient Snapshot: When patient 1 sends the wrong password to the server.

```
nunki.usc.edu(40): ./healthcenterserver availabilities.txt users.txt
Phase 1: The Health Center Server has port number 21247 and IP  address 68.181.201.3
Phase 1: The Health Center Server has reveived request from a patient with username patient1 and password password232.
Phase 1: The Health Center Server sends the response failure to the patient with username patient1.
Phase 1: The Health Center Server has reveived request from a patient with username patient2 and password password222.
Phase 1: The Health Center Server sends the response success to the patient with username patient2.
Phase 2: The Health Center Server receives a request for available timeslots from patients with port 34913 and IP address 68.181.201.3
Phase 2: The Health Center Server sends available time slots to patient with username patient2
Phase 2: The Health Center Server receives a request for appointment from patient with port number 34913 and username patient2
Phase 2: The Health Center Server confirms the following appointment 1 to patient with username patient2
```

Doctor Snapshot: When patient 1 sends the wrong password to the server.