CS 5523: Operating Systems

Homework 4

!!!!! No late HW will be accepted !!!!!!

(Check <u>BlackBoard Learn</u> for due date and submission)

Objectives

- Learn and practice client-server paradigm and peer-to-peer paradigm
- Implement a simple distributed system using socket programming
- Practice system calls and library functions in socket API in C/C++ **OR** Java

Description

You will design and implement a simplified tracking and chatting system, where a server keeps track of the registered clients and provide necessary information for clients to directly send each other short messages when they come close. You will implement a server program and a client program along with application layer protocol(s) to enable them to communicate. The same client program will be executed several times to simulate different users.

The **server** program will create a TCP socket and wait for client programs to connect. The server will create a thread to serve each client and maintain all the below mentioned information about each active client in a linked list.

Each **client** will connect to the server using TCP and **register** itself by reporting its initial (x,y) location along with other information such as name, sex, age etc. The server will assign a unique ID to each client and send this ID to the client. The client program will also create a UDP socket in a new thread to get the short messages that might be sent by other clients (peers) and print them on the screen. So the client program should also **register** its hostname and UDP port number to the server and server should maintain that information too.

After the registration step, the server program will wait for clients' requests and accordingly respond to them. A client's main thread can make the following requests:

1. **go -50 30** : client wants to go 50m South and 30m East, server updates the location and reports the new location (**go 40 -20** means go 40m North and 20m West)
2. **get location** : client wants to know its current (x,y) location, server reports the current location of this client
3. **list 30** : client wants to get the list of users within 30m, server determines such clients/users within 30m of requesting client and send their information (user-id, name, sex, age, host-name udp-port) to the client, client keeps that list in a linked list and prints their user-id, name, sex and age on the screen
4. **send user-id msg** : client sends msg to user-id (using the host-name and udp-port from the list mentioned above)
5. **quit** : client leaves the system, server takes it out from the list

Clearly, there will be some shared resource at the server and/or client so you need to protect them and implement necessary synchronization mechanisms.

Run the server program on hostA as follows:

```
hostA\> server -port num
```

Run each client on different hosts as follows:

```
anyhost\> client –SH hostA –SP num -N name -L x y -S [M|F] –A age
```

Grading: This is a 200-point homework.

First write a 2-3 page **report** (**20** points) to describe your design choices at the high level and particularly describe your protocols (i.e., describe message structures, in which order to exchange them, and what to do when a specific message is sent or received).

Then implement your server and client programs and make sure client can register and be able to interact with the server to update its locations, get list of other users etc. (**100** points)

Finally, enhance your client programs so that clients (peers) can send each other messages through udp. (This part will be **80** points)

Do all your work under a directory **lastname_hw4**, which should include your source codes, instructions to compile/execute, and some output files showing your test results etc...
Zip **lastname_hw4** and submit **lastname_hw4.zip** through BB Learn

----

Submission

You must submit your work using Blackboard Learn and respect the following rules:

1) All assignments must be submitted as either a zip or tar archive file unless it is a single pdf file.
2) Assignments must include all source code.
3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.