Title:

Internet Engineering Course
Socket Programming Project

Instructor:

Dr. Mohammad Nassiri

Adapted from:
Mr. Raja Venkatraman P R’s course project at University of Southern California

Spring 2012
The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth 25% of your overall grade in this course. You have to work as a team of 2 students to do this programming assignment. Any cheating will result in an automatic F in the course (not just in the assignment).

The Problem:

In this project you will be simulating a race among 4 mobile units to find their way with the help of two checkpoints and reach a target before running out of battery or going out of range. All communications take place over TCP and UDP sockets in client-server architecture. The project has 3 major phases: Bootstrapping, Route discovery and Contacting Target.

Code and Input files:

You must write your programs either in C or C++ on UNIX. In fact, you will be writing (at least) 3 different pieces of code:

1- Checkpoint
   a. You must create 2 concurrent checkpoints.
      i. Either by using fork() or a similar Unix system call. In this case, you probably have only one piece of code for which you need to use one of these names: chp.c or chp.cc or chp.cpp (all small letters). Also you must call the corresponding header file (if any) chp.h (all small letters). You must follow this naming convention.
      ii. Or by running 2 instances of checkpoint code. However in this case, you probably have 2 pieces of code for which you need to use one of these sets of names: (chp1.c and chp2.c) or (chp1.cc and chp2.cc) or (chp1.cpp and chp2.cpp) (all small letters). Also you must call the corresponding header file (if any) chp.h (all small letters) or chp1.h and chp2.h (all small letters). You must follow this naming convention.

2- Mobile unit
   a. You must create 4 concurrent Mobile Units
      i. Either by using fork() or a similar Unix system call. In this case, you probably have only one piece of code for which you need to use one of these names: mu.c or mu.cc or mu.cpp (all small letters). Also you must call the corresponding header file (if any) mu.h (all small letters). You must follow this naming convention.
      ii. Or by running 4 instances of mobile unit code. However in this case, you probably have 7 pieces of code for which you need to use one of these sets of names: (mu1.c, mu2.c, ..., mu4.c) or (mu1.cc, mu2.cc, ..., mu4.cc) or (mu1.cpp, mu2.cpp, ..., mu4.cpp) (all small letters). Also you must call the corresponding header file (if any) mu.h (all small letters) or mu1.h, mu2.h, ..., mu4.h (all small letters). You must follow this naming convention.

3- Target
   a. You must create a Target and you need to use any of these names: target.c, target.cc, target.cpp and target.h (all small letters).
4- Input files: scenario.txt, strategy.txt
   a. Scenario.txt contains the x-y GPS coordinates (x is an integer between 0 and 300 meters and y is an integer between 0 and 400 meters) for target and mobile units. strategy.txt contains the velocity (in terms of meters/sec), movement direction (W for west, E for east, S for South and N for North) and battery life (a positive integer less than or equal to 100) for mobile units. The format of the sample input files are as follow:

<table>
<thead>
<tr>
<th>scenario.txt</th>
<th>strategy.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>target 150 20</td>
<td>mu1 60 S 90</td>
</tr>
<tr>
<td>mu1 10 350</td>
<td>mu2 80 W 100</td>
</tr>
<tr>
<td>mu2 280 10</td>
<td>mu3 100 E 80</td>
</tr>
<tr>
<td>mu3 150 300</td>
<td>mu4 50 N 70</td>
</tr>
<tr>
<td>mu4 100 200</td>
<td></td>
</tr>
</tbody>
</table>

As can be seen, the delimiter used between the columns is one space. You may use the above samples to test your project, however when your project is graded, TAs will use different input files but with the exact same format and same # of rows and columns to test your project.

This project is divided into 3 phases. It is not possible to proceed to one phase without completing the previous phase. In each phase you will have multiple concurrent processes that will communicate either over TCP or UDP sockets.

**Phase1: Bootstrapping**

In this phase, each entity opens one or both input files and reads the necessary info from the input files then define address structures and creates UDP and/or TCP sockets

**Phase2: Route Discovery**

In this phase mobile units start moving according to the provided strategies in strategy.txt and try to correct course with the help of checkpoints while communicating with checkpoints over UDP sockets. Only Mobile units that get to a certain proximity of the Target will receive TCP port number of the Target from Checkpoints and the rest of Mobile units will be terminated. This is the end of phase 2.

**Phase3: Contacting Target**

At this point Target is up and listening for the incoming connection requests from the mobile units that moved on to phase3, accepts the incoming connection requests in the FCFS manner and sends responses back to mobile units through its TCP socket.

**A more detailed explanation of the problem:**

Figure 1 illustrates Mobile units, Checkpoints and the Target in a square field of 300 by 400. Figure 2 shows the required format for a message sent through either UDP or TCP sockets in this project. For a detailed and step by step explanation of the tasks for each entity in this project, please refer to flow charts in Figure 3, Figure 4 and Figure 5. Using these flow charts you may easily design and implement each entity. Table1 provides a list of all static and dynamic ports required for each entity. Dynamic Ports are assigned by the OS and you are required to use getsockname() to retrieve them.
<table>
<thead>
<tr>
<th>Identifier String</th>
<th>Type</th>
<th>Option</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>target, mu1,mu2,mu3,mu4, chp1, chp2</td>
<td>Any of the following: PORT, ACK, NACK, XY</td>
<td>Any of the following: X and Y coordinates for the mobile unit, or the TCP port number for the target or empty</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: General format of a message
Figure 3: Flowchart for each mobile unit

1. Input file: scenario.dat
   - Read its own (x, y) coordinates

2. Input file: strategy.dat
   - Read its own velocity, initial direction, and current battery life

3. Create a dynamic UDP port # and a UDP socket

4. Randomly select one of the two checkpoints as its guide for the entire project and initialize a timer to 0

5. Move for 1 second at the given velocity in the current direction, i.e. \( \text{\textit{sleep}}(1) \), then update the (x, y) coordinates and increment the timer by 1

6. **Print** "Mobile unit # is moving towards \( \langle \text{direction} \rangle \) on the screen"

7. **Print** "Mobile unit # recharging battery" on the screen

   - **Sleep**(2) in order to recharge, then update battery life to 100 (i.e. Full)

8. **Update** battery life (i.e. deduct 10 from current battery life for every 50 meters of movement)

9. Are its (x, y) coordinates inside borders of the designated field?

   - **YES**
     - **Print** "Mobile unit # communicating with checkpoint #" on the screen
     - **Update** battery life (i.e. deduct 10 from current battery life)
     - **Sleep**(2) in order to recharge, then update battery life to 100 (i.e. Full)
     - **Send** its current (x, y) in a UDP message to the selected checkpoint and receives a reply from it
   - **NO**
     - **Close** UDP socket
     - **Print** the error message "Mobile unit # outside borders" on the screen and **Terminate** this Mobile unit process

10. **Close** UDP socket

11. **Close** UDP socket, Extract the TCP port number for Target from the payload of the reply message

12. Create a dynamic TCP port and a TCP socket

13. **Connect** to Target through this TCP socket and upon successful connection send current (x, y) coordinates to Target and receive an ACK

14. **Print** the error message "Mobile unit # is out of Target range" on the screen and **Terminate** this Mobile unit process

15. **Print** the message "Mobile unit # has made contact with the Target" on the screen and **Terminate** this Mobile unit process

16. Randomly select a new direction other than the past directions for movement
Read \((x,y)\) coordinates for all mobile units and the Target

Create a UDP socket for itself using the hardcoded UDP port

Expect to receive a message from a mobile unit

Does it receive a message from a mobile unit?

Yes

Compose a UDP reply message of Type PORT and send the TCP port number of the Target to this mobile unit

Extract the \((x,y)\) coordinates of that mobile unit from the message

Calculate the current distance between that mobile unit and the Target

Distance < 50 m

Compose a UDP reply message of Type ACK

Is the mobile unit closer to Target since last time?

No

Compose a UDP reply message of Type NACK

Yes

Figure 4: Flowchart for each checkpoint
Figure 5: Flowchart for Target

Table 1: A Summary of Static and Dynamic assignment of TCP and UDP Ports and IP addresses

<table>
<thead>
<tr>
<th>Process</th>
<th>Dynamic ports</th>
<th>Static Ports</th>
<th>IP address obtained from</th>
</tr>
</thead>
<tbody>
<tr>
<td>chp1</td>
<td>None</td>
<td>1 UDP Port, 4100 plus last 3 digits of your ID (e.g. 4100+751=4851)</td>
<td>localhost</td>
</tr>
<tr>
<td>chp2</td>
<td>None</td>
<td>1 UDP Port, 4200 plus last 3 digits of your ID (e.g. 4200+751=4951)</td>
<td>localhost</td>
</tr>
<tr>
<td>Target</td>
<td>None</td>
<td>1 TCP Port, 21100 plus last 3 digits of your ID (e.g. 21100+751=21851)</td>
<td>localhost</td>
</tr>
<tr>
<td>mu1 through mu4</td>
<td>1 UDP and 1 TCP Port</td>
<td>None</td>
<td>localhost</td>
</tr>
</tbody>
</table>
Assumptions:

1. The Processes are started in the order that you believe is necessary or appropriate. Please mention it in your README file.
2. You are allowed and required to insert delays into your code e.g. by using sleep().
3. The timer used in flowchart in Figure 3 is not a timer for the real time of the system. It only records 3 seconds of movement. In other words it just keeps track of the maximum possible time a mobile unit spends moving. It has nothing to do with the real time of the system and you may define and maintain extra timers to keep track of the real time, should you find that necessary.
4. TCP port number of the target is hardcoded in each checkpoint and in the target.
5. UDP port numbers for checkpoints are hardcoded in each mobile unit and the corresponding checkpoint.
6. Each message starts with the identifier string for the sender of that packet.
7. Messages of Type ACK or NACK have no payload.
8. Messages of type PORT carry the TCP port number of target as the payload.
9. Messages of type XY carry the x,y coordinates of mobile units in an arbitrary format as payload.
10. If you need to have more code files than the ones that are mentioned here, please use meaningful names and mention them all in your README file.
11. You are allowed to use blocks of code from Beej’s socket programming tutorial (Beej’s guide to network programming) in your project.
12. When you run your code, if you get the message “port already in use” or “address already in use”, please first check to see if you have a zombie process (from past logins or previous runs of code that are still not terminated and hold the port busy). If you do not have such zombie processes or if you still get this message after terminating all zombie processes, try changing the static UDP or TCP port number corresponding to this error message (all port numbers below 1024 are reserved and must not be used). If you have to change the port number, please do mention it in your README file.

Requirements:

1. Each entity must display its TCP and/or UDP port number and IP address on screen upon startup or later during the runtime of project.
2. Refer to Table1 to see which ports are statically defined and which ones are dynamically assigned. Use getsockname() function to retrieve the locally-bound port number wherever ports are assigned dynamically as shown below:

```c
//Retrieve the locally-bound name of the specified socket and store it in the sockaddr structure
getsock_check=getsockname(TCP_Connect_Sock,(struct sockaddr *)&my_addr, (socklen_t *)&addrlen) ;
//Error checking
if (getsock_check== -1) {
    perror("getsockname");
    exit(1);
}
```

3. Use gethostbyname() to obtain the IP address of the local host however the host name must be hardcoded localhost in all pieces of code.

4. You can either terminate all processes after completion of phase3 or terminate them as suggested by flow charts.

5. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.

6. All messages sent through sockets must start with an identifier string as instructed in the project description.
7. You are not allowed to pass any parameter or value or string or character as a command-line argument. No user interaction must be required (except for when the user runs the code obviously). Every thing is either hardcoded or dynamically generated as described before.

8. All the on-screen messages must conform exactly to the project description. You should not add anymore on-screen messages. If you need to do so for the debugging purposes, you must comment out all of the extra messages before you submit your project.

9. Using fork() or similar system calls are not mandatory if you do not feel comfortable using them to create concurrent processes.

10. Please do remember to close the socket and tear down the connection once you are done using that socket.

Programming languages and compilers:

You must use only C/C++ on UNIX as well as UNIX Socket programming commands and functions. Here are the pointers for Beej’s Guide to C Programming and Network Programming (socket programming):

http://www.beej.us/guide/bgnet/

(If you are new to socket programming please do study this tutorial carefully as soon as possible and before starting the project)
You can use a unix text editor like emacs (or vi/vim) to type your code and then use compilers such as g++ (for C++) and gcc (for C) to compile your code. You must use the following commands and switches to compile yourfile.c or yourfile.cpp. It will make an executable by the name of "yourfileoutput".

```
gcc -o yourfileoutput yourfile.c  -lsocket -lnsl -lresolv

g++ -o yourfileoutput yourfile.cpp -lsocket -lnsl -lresolv
```

Do NOT forget the mandatory naming conventions mentioned before!

Also inside your code you need to include these header files in addition to any other header file you think you may need:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

**Submission Rules:**

1. Along with your code files, include a **README** file which is plain text file (NOT a word document or a pdf file). In the README file please write
   a) Your **Full Name** as given in the class list
   b) Your **Student ID**
   c) What you have done in the assignment
   d) What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
   e) What the TA should do to run your programs. (Any specific order of events should be mentioned.)
   f) The format of all the messages exchanged (other than what is mentioned in the project description).
   g) Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
   h) Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)

Submissions **WITHOUT README** files **WILL NOT BE GRADED**.

2. Compress all your files including the README file into a single “tar ball” and call it:
IE902_username1_username2_univ.tar.gz (all small letters) e.g. my file name would be
IE902_nassiri_basu.tar.gz. Please make sure that your name matches the one in the class list. Here are
the instructions:

a. On your host, go to the directory which has all your project files. Remove all executable and other
unnecessary files. Only include the required source code files and the README file. Now run the following commands:

```
b. tar cvf IE902_username1_username2_univ.tar * - Now, you will find a file named
    “IE902_username1_username2_univ.tar” in the same directory.

c. gzip IE902_username1_username2_univ.tar – Now, you will find a file named
    “IE902_username1_username2_univ.tar.gz” in the same directory.
```

11. You have plenty of time to work on this project and submit it in time hence there is
    absolutely zero tolerance for late submissions! Do NOT assume that there will be a late
    submission penalty or a grace period. If you submit your project late (no matter for
    what reason or excuse or even technical issues), you simply receive a zero for the
    project.
Grading Criteria:

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, specially the communications through UDP and TCP sockets.

2. Inline comments in your code. This is important as this will help in understanding what you have done.

3. Whether your programs work as you say they would in the README file.

4. Whether your programs print out the appropriate messages and results.

5. If your submitted codes, do not even compile, you will receive 10 out of 100 for the project.

6. If your submitted codes, compile but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.

7. If your codes compile but when executed only perform phase1 correctly, you will receive 40 out of 100.

8. If your codes compile but when executed perform only phase1 and phase 2 correctly, you will receive 80 out of 100.

9. If your code compiles and performs all tasks in all 3 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.

10. If you forget to include any of the code files or the README file in the project tar-ball that you submitted, you will lose 5 points for each missing file (plus you need to send the file to the TA in order for your project to be graded.)

11. If your code does not correctly assign the TCP or UDP port numbers dynamically (in any phase), you will lose 20 points.

12. You will lose 5 points for each error or a task that is not done correctly.

13. The minimum grade for an on-time submitted project is 10 out of 100.

14. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend about 2 months on this project and it doesn’t even compile, you will receive only 10 out of 100.

15. Using fork() or similar system calls are not mandatory however if you do use fork() or similar system files in your codes to create concurrent processes (or threads) and they function correctly you will receive 10 bonus points.
16. If you submit a makefile or a script file along with your project that helps us compile your codes more easily, you will receive 5 bonus points.

17. The maximum points that you can receive for the project with the bonus points is 100. In other words the bonus points will only improve your grade if your grade is less than 100.

18. Your code will not be altered in any ways for grading purposes and however it will be tested with different input files. Your designated TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not.

**Cautionary Words:**

1. Start on this project early!!!

2. In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is a *linux-based host*. It is strongly recommended that students develop their code on a linux-based host like Ubuntu.

3. You may create zombie processes while testing your codes, please make sure you kill them every time you want to run your code. To see a list of all zombie processes even from your past logins to nunki, try this command: `ps –aux | grep <your_username>`

4. Identify the zombie processes and their process number and kill them by typing at the command-line:

5. Kill `-9 processnumber`

6. Please do remember to terminate all zombie or background processes, otherwise they hold the assigned port numbers and sockets busy and we will not be able to run your code in our account on nunki when we grade your project.

**Academic Integrity:**

All students are expected to write all their code on their own.

Copying code from friends is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use and you did not write, must be listed in your README file. All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. “I didn’t know” is not an excuse.