Technical Report – Robin Hood

Tudor Cezar

Universitatea Alexandru Ioan Cuza, Bulevardul Carol I nr. 11, Iași

tudor.cezar01@gmail.com

1. Introduction

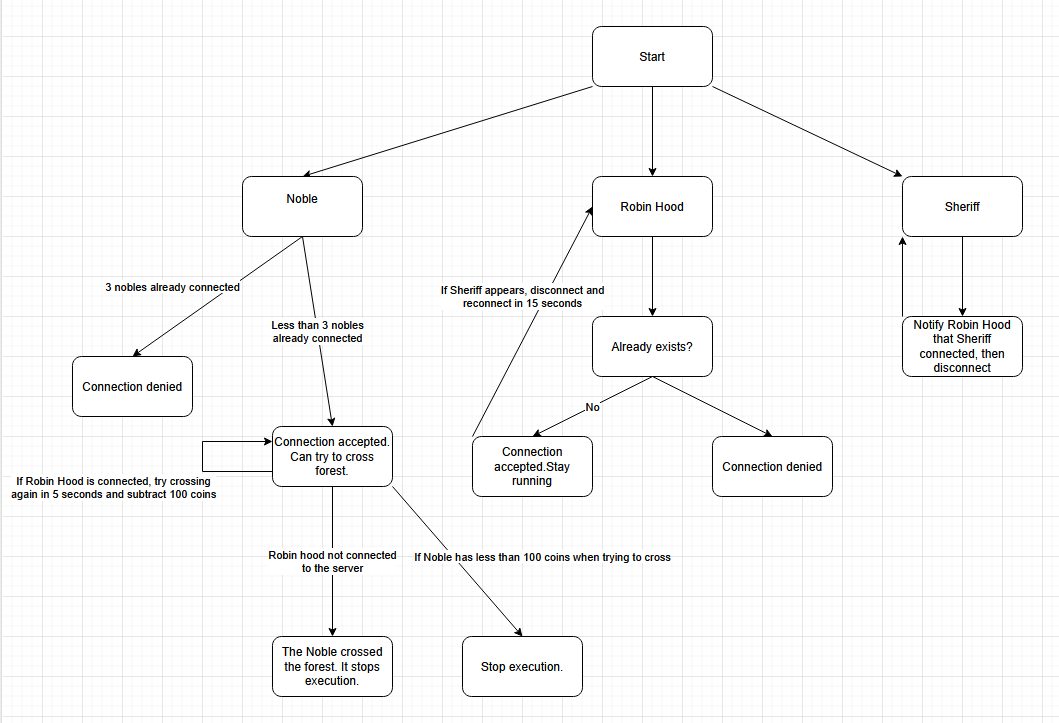
The project consists of a concurrent TCP server that simulates the following situation: at first, a Robin Hood client will connect to the server. The following clients that connect will either be of type Noble (a maximum of 3 connected at the same time) or of type Sheriff (a maximum of 1 connected at any given time). The appearance of the Sheriff will temporarily disconnect Robin Hood from the server, so that clients of type Noble will be able to pass through the forest. Otherwise, if Robin Hood is present, a total of 100 coins will be deducted from the Noble client, and it will have to try to pass again shortly. If the Noble type client reaches 0 coins, it will exit. A mention: Noble type clients correctly connect to the server with the following type of message: nobil <id> <coins>, where id is a number between 0 and 100.

1. Used technologies

The project uses the TCP. TCP is more reliable guarantees the delivery of data to the destination. This is important for this project, as for example we want to completely avoid the situation in which a Noble client passes the forest while Robin Hood is there.

Threads were used to obtain concurrency and to ease the communication between instances of different clients connected to the server (since threads share global memory)

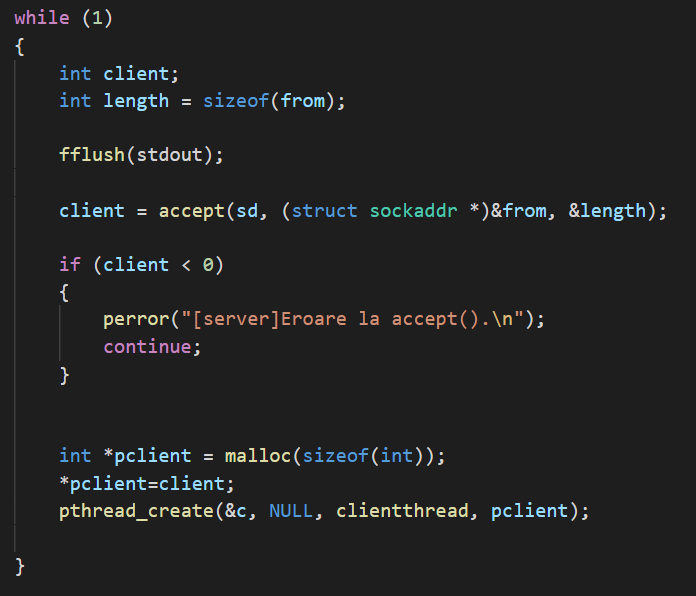
1. Application Architecture



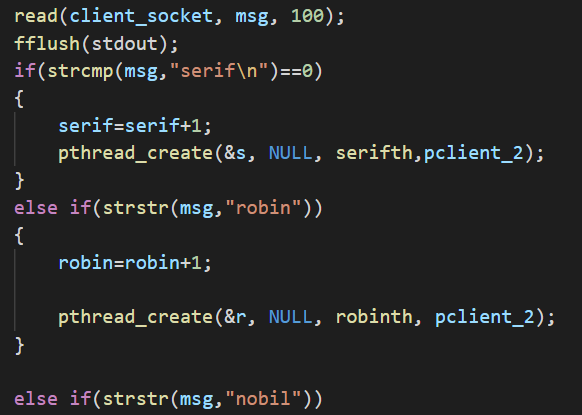
1. Implementation details



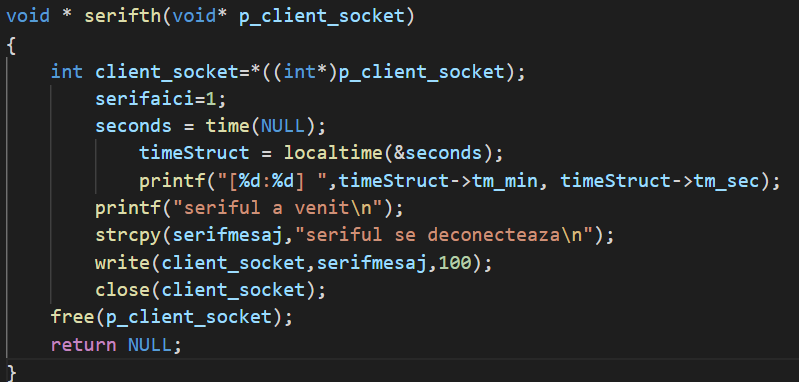
The client attempts to connect to the server



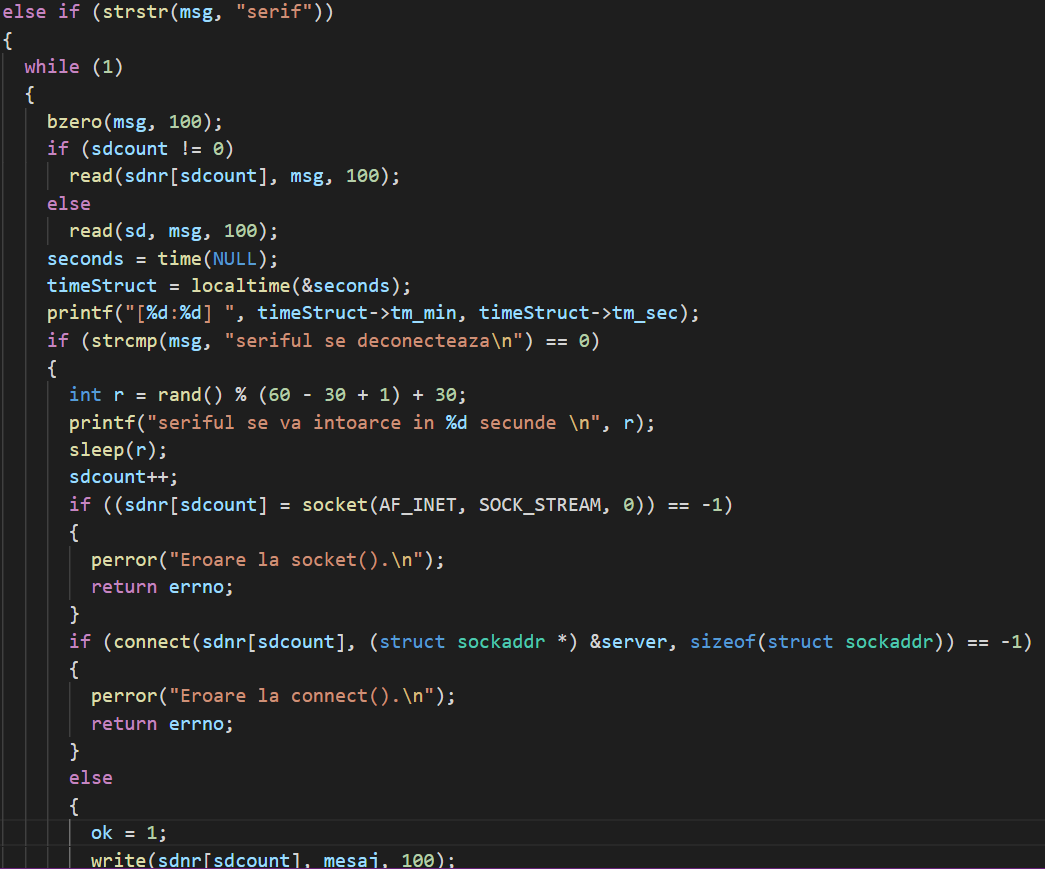
In a while loop, the server waits for clients to connect. One thread per client.



This is inside the client thread. After receiving a message from the client, the client thread then creates another thread, depending on the message received: the client is either a robin client, a noble client or a sheriff client. We’ll take a look at each of them.



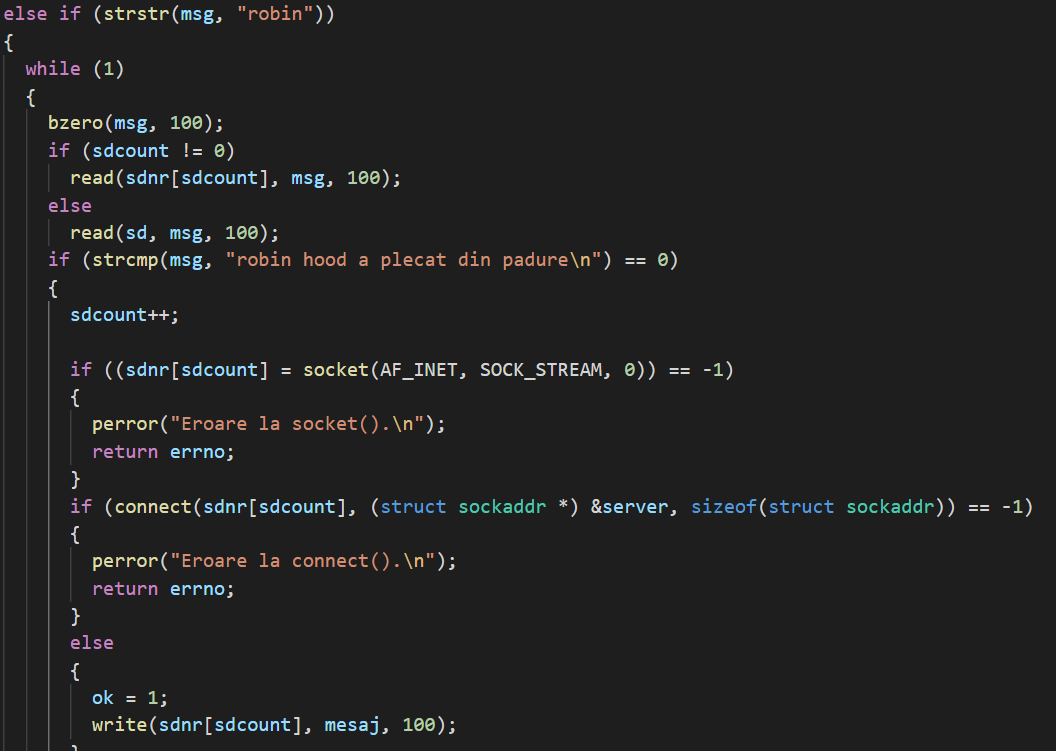
This is the sheriff thread. When a sheriff thread is created, a variable that notifies Robin Hood of the sheriff’s arrival is modified. Right after its arrival, the sheriff thread will send a message to the sheriff client. The client socket closes. Let’s see what happens in the sheriff client.



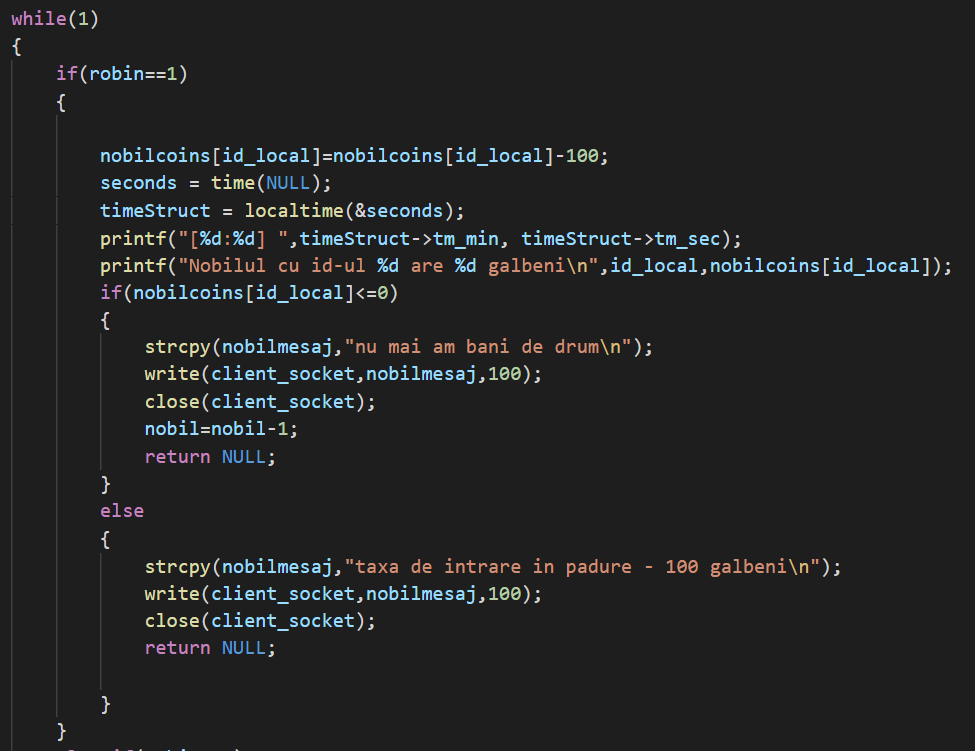
After receiving the message from the sheriff thread, a random variable is created. After that amount of time, the sheriff client will attempt to reconnect to the server like it did the first time, but with a new socket. Any connection requests to the server will be served instantly, it being a concurrent server. So the sheriff client then sends a message to the server containing its role, again. The process repeats itself.



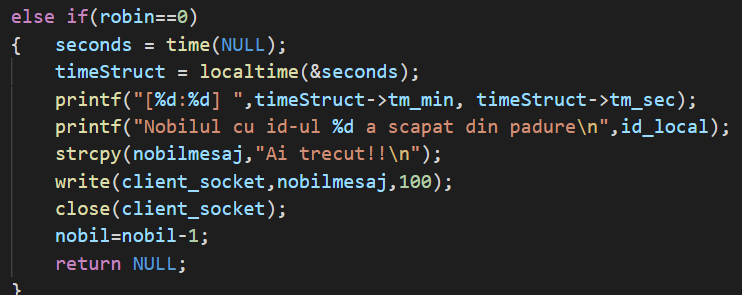
This is the Robin Hood’s thread function. It modifies a global variable that will be used by the noble clients to know whether to pass or not, and then enters a while loop, waiting for the sheriff’s arrival. Once the sheriff arrives the mentioned variable changes to 0, it closes the client socket, sends a message to the client and closes the client socket. After 15 seconds, the Robin Hood client will try to reconnect to the server.



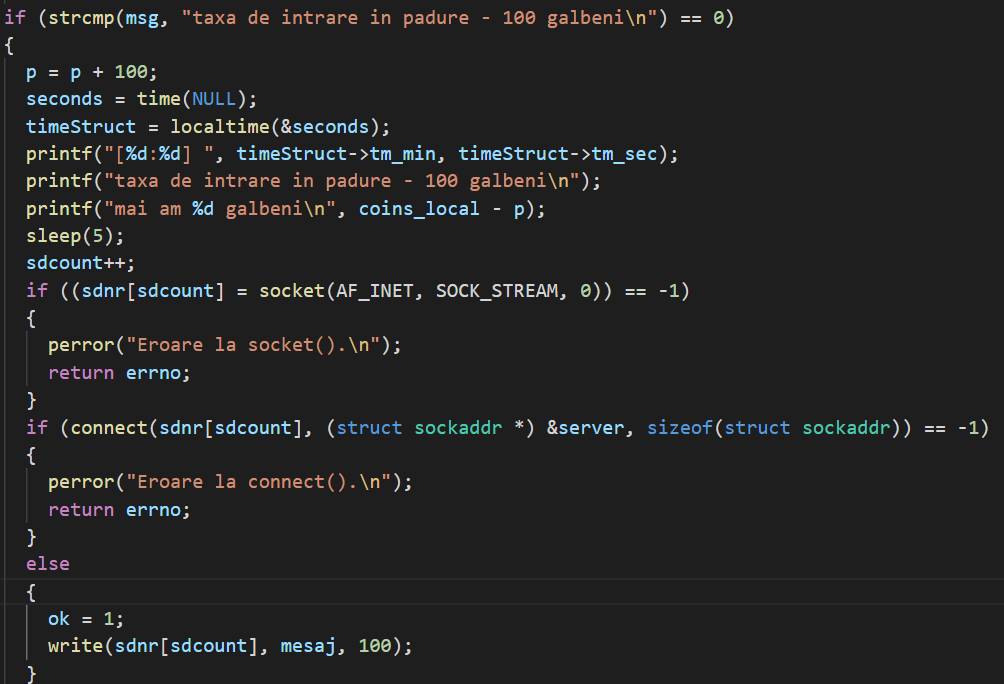
In the Robin Hood client, after receiving the message from the server, similarly to the sheriff client, it will try reconnecting to the server through a different socket.



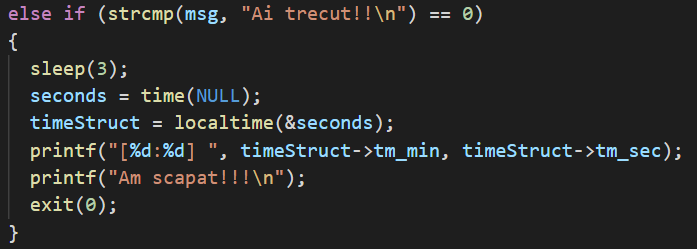
The first part of the noble thread function. In case Robin Hood is in the forest, the thread will subtract 100 coins from the noble client. It then sends a message to the noble client depending on whether the noble has 0 coins left or more. In both cases, the socket closes and the thread exits.



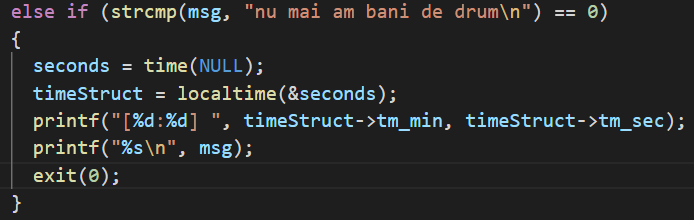
If instead Robin Hood is not there because of the sheriff, the thread sends a message to the client notifying that the noble passed the forest, then closes the socket.



In the noble client, if the received message is that of the subtraction of 100 coins, the client will wait 5 seconds then connect to the server again.

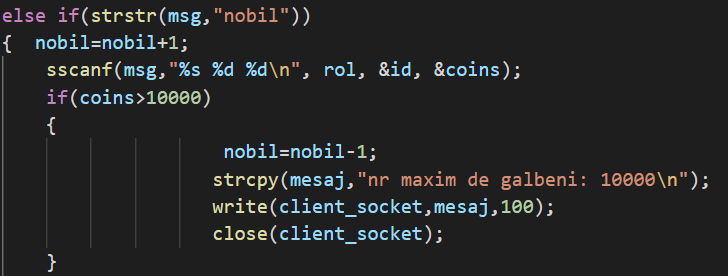


If Robin Hood wasn’t there and the corresponding message was received by the client, it will print out a message in the console then stop execution.

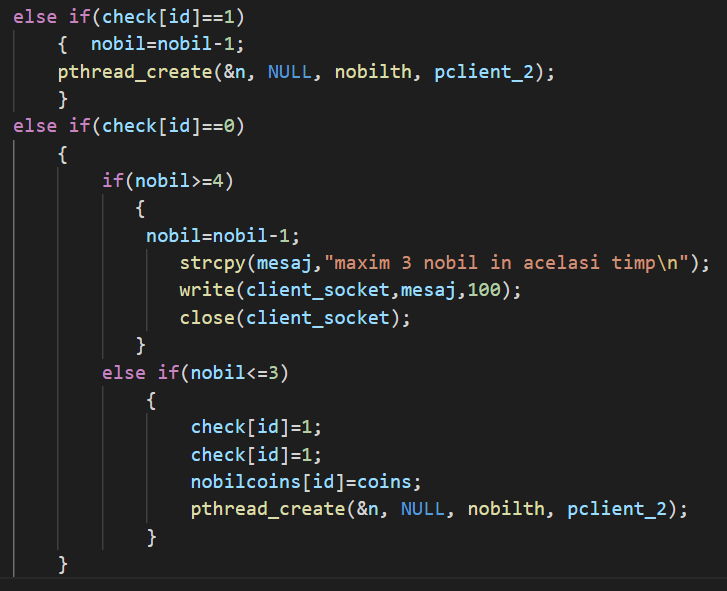


If the sheriff failed to save the noble and Robin Hood got all of their money after repeated tries, the noble client will print out a message then stop execution.

**Two restrictions:**



Before creating a noble type thread, the client type thread checks two things: that the noble starts with no more than 10000 coins (in the picture above), and that there are no more than 3 nobles trying to cross the forest at the same time (in the picture below).



The noble thread is only created if: The noble client is a returning one (it has already tried to cross the forest, and is now trying again) or if it’s a new noble client and there are less than 3 current nobles trying to cross the forest.

1. Conclusions

A few improvements/additions that can be made:

* A graphic interface
* More types of clients with reference to Robin Hood that can interact with the existing ones.
* Cleaning up redundant code

References/Bibliography

1. <https://profs.info.uaic.ro/~computernetworks/files/NetEx/S9/cliTcp.c>
2. <https://profs.info.uaic.ro/~computernetworks/files/NetEx/S9/servTcpCSel.c>
3. <https://www.codevscolor.com/c-print-current-hour-minute-second>
4. <https://youtu.be/Pg_4Jz8ZIH4>
5. <https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c>
6. https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c