Assignment 6 Part 1

Kate Rupar and Mark Thekkethala

Introduction

In this part, we implemented a network communication layer for a distributed application, informed by our prototype from A5. The system consists of a single rendezvous server (registrar) and an arbitrary number of nodes. The goal of the network is to enable nodes to communicate directly with one another. The server allows new nodes to register themselves, and then sends an updated list of node addresses and ports to the nodes already in the network. Throughout our code, we primarily set the arrays to large sizes so we do not have to deal with memory reallocation, however a more efficient future solution would incorporate designing and utilizing a dynamically growing array class. In the following sections we will discuss our design as well as provide use cases for our implementation.

Message

The Message class represents the types of communication between the nodes and server. There are four types of Message we currently account for in our design: Ack, Status, Register, and Directory. For more information about Message and our serialization/deserialization process, see the part two MEMO.pdf.

Node

The Node class can be found in node.h. A Node has several fields within it. First, it keeps track of the status of the Network by keeping a number of nodes, list of ports, and list of addresses (a Directory). In our design, the first elements in the port and address list are the Server port and address. The Node class also contains the IP address and port of the Node. Lastly, the Node class has a socket, sock_send, which is able to send Messages and receive a response. Nodes can send Acks, Statuses, and Registrations. They cannot send Directory Messages, and cannot receive Registration Messages.

send_data

For our program, we utilize socket programming, heavily inspired by this GeeksForGeeks post. We use the library <sys/socket.h>.The send_data method resets the sock_send socket. It then attempts to connect to the destination IP and port and send a Message. After the message is sent, sock_send calls handle_packet, which handles the response Message- this will be discussed in the next section. To interact with this method, we have several helper methods like send_reg and send_status which build a Message and then call send_data.

handle_packet

In handle_packet, we read in the char* received by send_socket and parse it into a Message. This is done using the deserialization process described in the part 2 MEMO.pdf. The type of Message is identified by the first character in the string, which is a number indicating the MessageKind: 1-Register, 2-Ack, 3-Status, 4-Directory. We use helper functions like handle_directory to then interact with the Message. For example, handle_directory sets the Node’s directory (the nodes, ports and addresses fields) equal to the ones received in the Directory Message. Currently, for Ack we print, “Ack received in NODE”, and for Status we print the Status’s message. In our design, we utilize Status as a way to send messages between Network components.

Constructor

In our implementation, the constructor contains the functionality of the Node. In our example, every Node registers and then sends a Status. The init_client method is called before every task; it initializes and sets up sock_send to send. After the Node is done with its tasks, we have it repeatedly call handle_packet in a while loop so that we are able to see it recognize additional nodes joining the network and receive new Directory Messages from the Server.

Server

Our Server class in server.h is the brains of the Network. Servers contain the same fields as Node plus an additional socket, sock_listen, for receiving and an array of sockets which allow the Server to keep all of its connections with the multiple nodes. Servers can send Directory, Ack, and Status Messages. They cannot send Registration Messages or recieve Directory Messages.

handle_packet

handle_packet is extremely similar to Node’s method, except that the cases have different behavior. When a Register is received, Server updates its Directory and sends that updated directory to all of the nodes in the network using the list of sockets initialized in the constructor. Like in Node, when a Server receives an Ack or Status it simply prints.

Constructor

The constructor for Server is complex, since it sets up sock_listen and sock_send. There are steps to the process: creating the socket file descriptor, setting the socket options, binding the socket to the Server's port, listening for activity, accepting a connection and then reading and handling the received Message in handle_packet. These steps are all wrapped in a while loop so the sock_send socket is constantly being reconfigured and sock_listen repeatedly listens.

How to Run

To run our example, call 'make run' in the part1 folder.

Use Cases

The following are examples of functionality in our program:

```
//Creating a Status
Status* s = new Status(0, 1, new String("hello"));

//Creating a Register
Register* r = new Register(0, 1, 8080, new
String("127.0.0.1"));

//Creating an Ack
Ack* a = new Ack(0, 1);

//Creating a Directory
size_t* ports = new size_t[2];
ports[0] = 8080;
ports[1] = 8080;
String** add = new String*[3];
add[0] = new String("127.0.0.7");
add[1] = new String("127.0.0.1");
Directory* d = new Directory(0, 1, 2, ports, add);

//Creating a Node
Node* n1 = new Node(new String("127.0.0.7"),8080, new
String("127.0.0.1"), 8080);

//Creating a Server
Server* s1 = new Server(new String("127.0.0.7"), 8080);

//When a node is constructed, it automatically sends a Register
n1->send_data(r);
//And the Server will receive the Register and update its
//Directory
```

```
n1->nodes == 1 //This is # of systems in network including the
               //Server
s1->handle_packet();
s1->nodes == 2
//handle_packet sends the new Directory to the Node
//The Node receives this Directory and updates its Directory
n1->nodes == 1
n1->handle_packet(d);
n1->nodes == 2
//Now that Node is Registered, sends Status to Server
n1->send_data(s);
//Server receives Status and prints message
s1->handle_packet() -> “hi”
//handle_packet Sends Ack back to Node
//Node receives Ack and prints
n1->handle_packet() -> “Ack received in NODE”

//New Node Registers to Network
Node* n2 = new Node(new String("127.0.0.7"),8080, new
String("127.0.0.2"), 8080);
Register* r = new Register(0, 1, 8080, new
String("127.0.0.2"));
n2->send_data(r2);

//Server receives Register, and sends updated Directory to both
//Nodes
s1->handle_packet()
size_t* ports = new size_t[3];
ports[0] = 8080;
ports[1] = 8080;
ports[2] = 8080;
String** add = new String*[3];
add[0] = new String("127.0.0.7");
add[1] = new String("127.0.0.1");
add[2] = new String("127.0.0.2");
Directory* d2 = new Directory(0, 1, 3, ports, add);
//n1 receiving d2
n1->nodes == 2
n1->handle_packet();
n1->nodes == 3
//n2 receiving d2
n2->nodes == 1 \\not registered yet
n2->handle_packet();
```

```
n2->nodes == 3
```