Group 15

Sun Yul Lee

Whitley Pittman

Moon Hyun Kim

Laboratory Assignment 2 Report

Introduction

Our names are Sun Yul Lee, Whit Pittman, and Moon Hyun Kim, and we are Labs Group 15. For this assignment, we set up a virtual ring of nodes over the Internet as covered in Phase 1 of the specifications. The ring is managed by our server (master node) and has a ring ID of 0. All the other nodes of the ring are our clients (slave nodes) with a ring ID assigned by the master. After these nodes are joined forming a ring, the objective is that the nodes on the ring must only use the ring to communicate. Again, we are setting up the ring for this assignment, and the communication between nodes will be addressed in the next assignment. The source code for the client and server that we created both compile and execute with components working as desired by specifications as we will further explain in this report.

Files

- Slave.py, Master.c

System Architecture

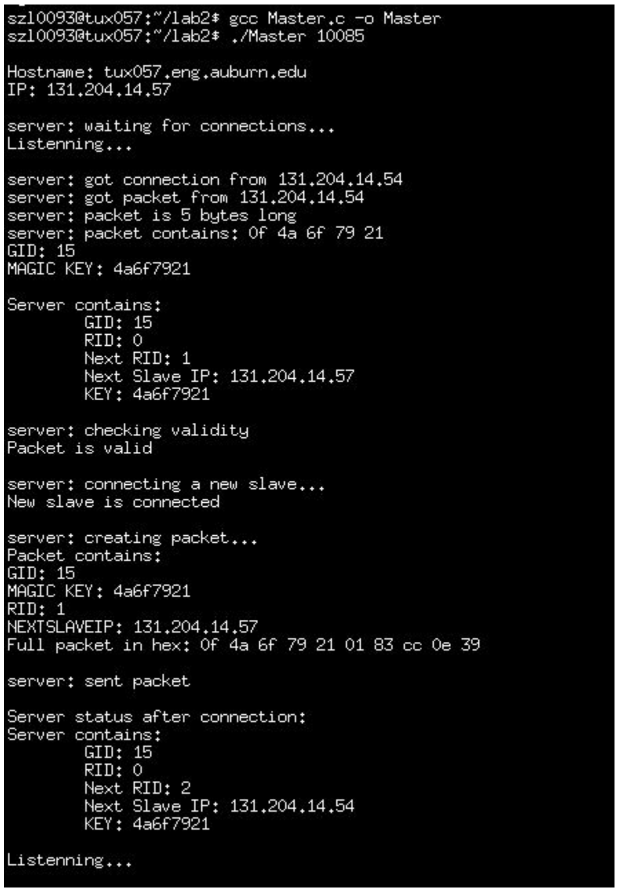
We will now specify the languages we used to write our source code. For our system, the client was written in Python, and the server was written in C. The distinct port number where our server works is 10085.

- Slave.py - First, this program gets the hostname and port number of the server from the user. It creates a socket using socket(), then shows the user a welcome message, hostname, and port number of the server. After that, the program connects to the server using connect() and shows the IP Address of the server. Then, the program sets the GID using the port number that the user provided and sets a key which is a magic number that the server will use for checking validity. With the values, the program creates a packet of size 5 bytes. Then, the program sends the packet using send(). The program receives a packet from the server using recv(). After the program receives a packet, it checks validity of the packet, and if it is invalid the program shows a message that the packet is invalid and closes the program. If the packet is valid, the program shows the content of received packet and gets data from the packet to set values of GID, key, RID, and next slave IP. After setting these values, the program closes the socket using close. Then, it shows the data that it receives from the server and a status of Slave.
- Master.c - First, this program gets the port number of the server from the user. Then, the server system creates the address information with the port number using getaddrinfo(). It creates a socket using socket() and binds the socket to the port using bind(). Then, the program is willing to listen from the client using listen(). If a client tries to connect to the server, the program accepts the connection using accept(). After that, using read(), the system gets the packet from the client and shows the content of the packet. The program checks validity of the packet. If the packet is invalid, it shows a message that the packet is invalid, sends fake packet, closes the connection, and listens the other client's connection. If the packet is valid, the program categorizes the data in the packet and

shows it to user. Then, the system shows the status of the server (GID, RID, next RID, next slave IP, and key) to the user. After that, the system sets attributes of the slave struct (GID, magic) with the data from the received packet and checks the validity of the packet. If the packet is valid, the system updates attributes of the master struct and slave struct. If the packet is invalid, the system creates a fake packet, sends the packet to the client, shows a message that the received packet is invalid, and closes the connection. After checking validity and updating values, the system creates a packet and sends the packet to client using send(). Then, the system closes the connection and shows new status of the server to user.

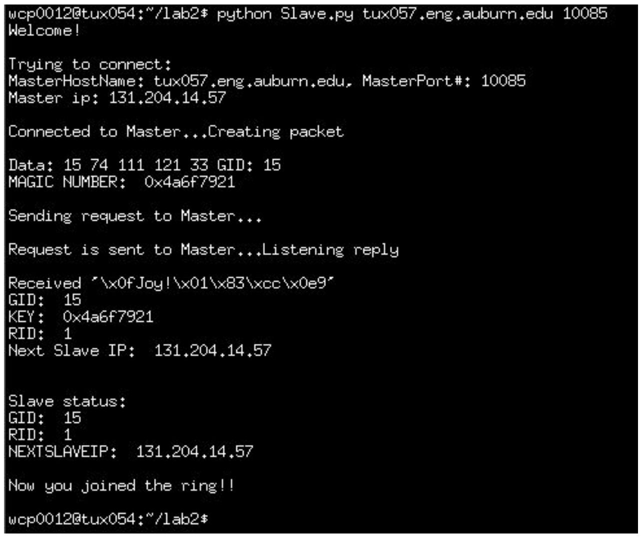
System Description

How to compile and run the Slave.py and Master.c is shown in Figure 1(a) and 1(b). For compiling and executing Slave.py, we type “python Slave.py <MasterHostName> <MasterPortNumber>” where <MasterHostName> is the master’s hostname and <MasterPortNumber> is the master’s port number. For compiling Master.c, we type “gcc Master.c -o Master” and for executing Master.c, we type “./Master <PortNumber>” where <PortNumber> is the master’s port number. We tested the programs by using a Tux machine and both programs were compiled and ran successfully. Figure 1(a) shows the flow of the client system, and Figure 1(b) shows the flow of the server system.



```
szl0093@tux057:~/lab2$ gcc Master.c -o Master
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...

server: got connection from 131.204.14.54
server: got packet from 131.204.14.54
server: packet is 5 bytes long
server: packet contains: 0f 4a 6f 79 21
GID: 15
MAGIC KEY: 4a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 1
        Next Slave IP: 131.204.14.57
        KEY: 4a6f7921

server: checking validity
Packet is valid

server: connecting a new slave...
New slave is connected

server: creating packet...
Packet contains:
GID: 15
MAGIC KEY: 4a6f7921
RID: 1
NEXTSLAVEIP: 131.204.14.57
Full packet in hex: 0f 4a 6f 79 21 01 83 cc 0e 39

server: sent packet

Server status after connection:
Server contains:
        GID: 15
        RID: 0
        Next RID: 2
        Next Slave IP: 131.204.14.54
        KEY: 4a6f7921

Listenning...
```

Figure1(a)



```
wcp0012@tux054:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Received '\x0fJoy!\x01\x83\xcc\x0e9'
GID:  15
KEY:  0x4a6f7921
RID:  1
Next Slave IP:  131.204.14.57


Slave status:
GID:  15
RID:  1
NEXTSLAVEIP:  131.204.14.57

Now you joined the ring!!

wcp0012@tux054:~/lab2$
```

Figure1(b)

Experiments

- Assumption

  - We assume that user put valid value for inputs. For example, we assume that user put a correct value for the hostname and the port number for Slave and the port number for Master when he or she executes the programs.

  - We assume that the server system, Master, is executed before the client system, Slave.

  - We assume that the server system never shuts down.

- We assume that group number in this class is no more than 30. Our systems set the GID using the port number from the user. For example, in Master.c, we set the GID by calculating (port - 10010) / 5 where port is the port number of the server.

- Notification
  - Our server system Master contains codes that sends a fake packet to client if the server gets an invalid packet. The fake packet contains 0xFF010203040506070809. So, the client system, Client, will show 0xFF as the GID, 0x01020304 as the magic number, 0x05 as the RID, and 6.7.8.9 as the next slave IP address if the system does not have a validity check algorithm.
  - Our client system, Slave, contains code that checks the validity of a packet received from the server.
- Error Condition

1. Size of received data in Master program is not 5 bytes.
2. Values in magic number is not valid.

   a) If magic number from Slave is not “4A6F7921”, it is invalid.
3. GID from Master and Slave are not matched.

   a) For our group, if GID from Slave or Master is not 15, it is invalid.

- Testing example

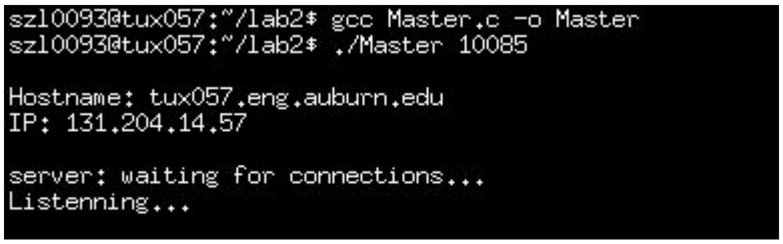
- Connecting 3 Slaves to Master

1. We compiled and executed our server system Master.c in tux057. The system showed messages as Figure2.

2. We executed our client system Slave.py in tux054. The server system showed messages as Figure3(a), and the client system showed messages as Figure3(b). The first slave was connected to the ring. Next, the RID of the server system was changed to 2, and the Next Slave IP Address of the server system was updated to the IP Address of the first slave. The RID of first slave was set as 1, and the Next Slave IP Address of the first slave was set as the IP Address of the server.

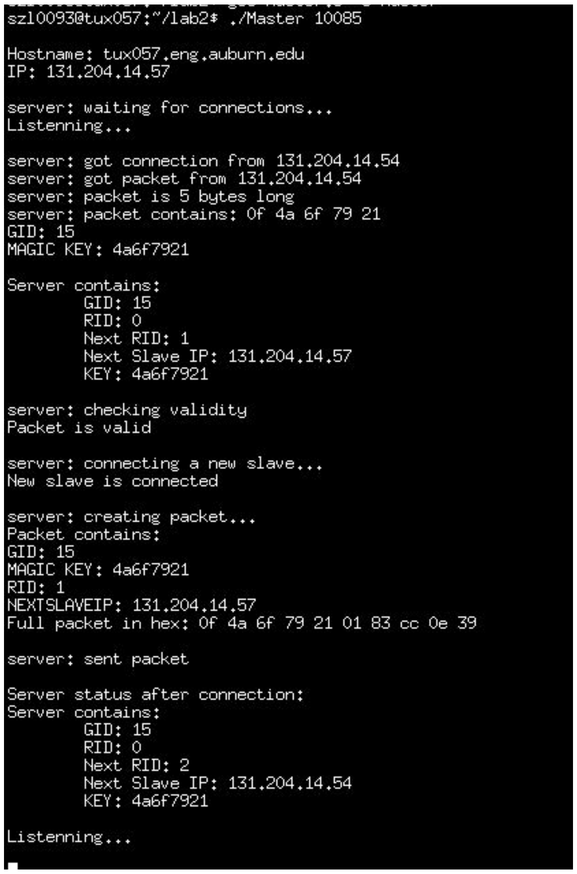
3. We executed the second client system in tux056. The system showed messages as Figure4(a), and the second client system showed messages as Figure4(b). The second slave was connected to the ring. The next RID of the server system was changed to 3, and the Next Slave IP Address of the server system was updated to the IP Address of the second slave. The RID of second slave was set as 2, and the Next Slave IP Address of the second slave was set as the IP Address of the first slave.

4. We executed third client system in tux061. The system showed messages as Figure5(a), and the third client system showed messages as Figure5(b). The third slave was connected to the ring. The next RID of the server system was changed to 4, and the Next Slave IP Address of the server system was updated to the IP Address of the third slave. The RID of third slave was set as 3, and the Next Slave IP Address of the third slave was set as the IP Address of the second slave.



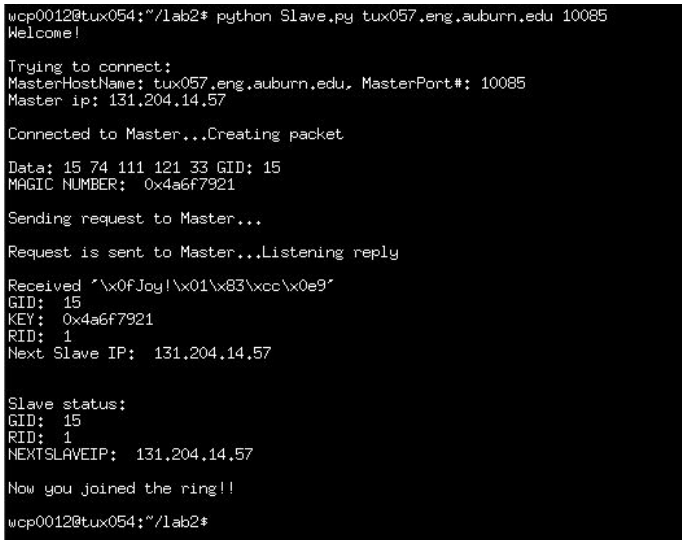
```
szl0093@tux057:~/lab2$ gcc Master.c -o Master
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...
```

Figure2



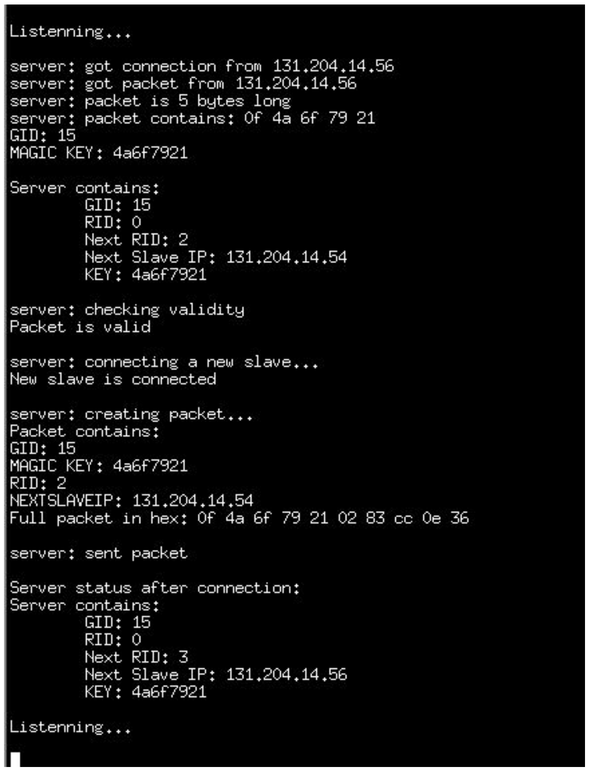
```
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...

server: got connection from 131.204.14.54
server: got packet from 131.204.14.54
server: packet is 5 bytes long
server: packet contains: 0f 4a 6f 79 21
GID: 15
MAGIC KEY: 4a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 1
        Next Slave IP: 131.204.14.57
        KEY: 4a6f7921

server: checking validity
Packet is valid

server: connecting a new slave...
New slave is connected

server: creating packet...
Packet contains:
GID: 15
MAGIC KEY: 4a6f7921
RID: 1
NEXTSLAVEIP: 131.204.14.57
Full packet in hex: 0f 4a 6f 79 21 01 83 cc 0e 39

server: sent packet

Server status after connection:
Server contains:
        GID: 15
        RID: 0
        Next RID: 2
        Next Slave IP: 131.204.14.54
        KEY: 4a6f7921

Listenning...
```

Figure3(a)



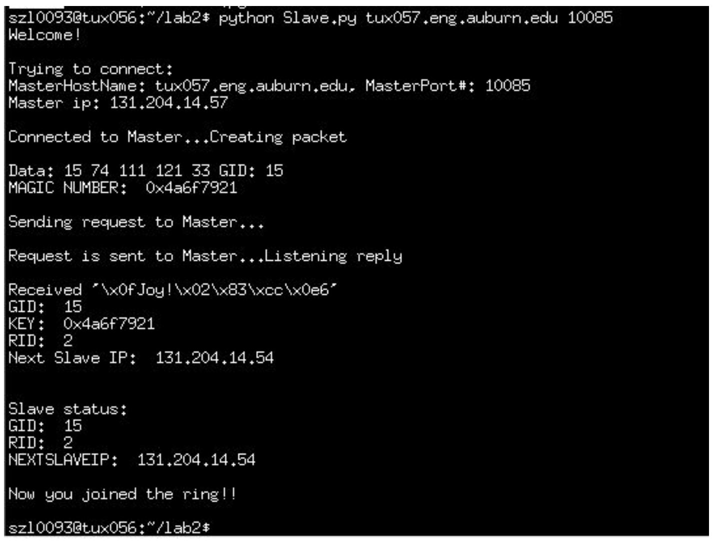
```
wcp0012@tux054:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Received '\x0fJoy!\x01\x83\xcc\x0e9'
GID:  15
KEY:  0x4a6f7921
RID:  1
Next Slave IP:  131.204.14.57


Slave status:
GID:  15
RID:  1
NEXTSLAVEIP:  131.204.14.57

Now you joined the ring!!

wcp0012@tux054:~/lab2$
```

Figure3(b)



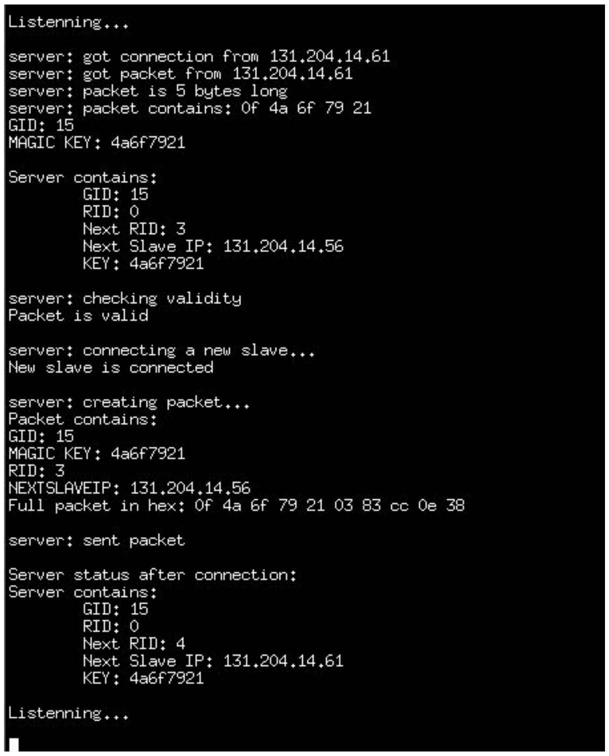
```
Listenning...

server: got connection from 131.204.14.56
server: got packet from 131.204.14.56
server: packet is 5 bytes long
server: packet contains: 0f 4a 6f 79 21
GID: 15
MAGIC KEY: 4a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 2
        Next Slave IP: 131.204.14.54
        KEY: 4a6f7921

server: checking validity
Packet is valid

server: connecting a new slave...
New slave is connected

server: creating packet...
Packet contains:
GID: 15
MAGIC KEY: 4a6f7921
RID: 2
NEXTSLAVEIP: 131.204.14.54
Full packet in hex: 0f 4a 6f 79 21 02 83 cc 0e 36

server: sent packet

Server status after connection:
Server contains:
        GID: 15
        RID: 0
        Next RID: 3
        Next Slave IP: 131.204.14.56
        KEY: 4a6f7921

Listenning...
```

Figure4(a)



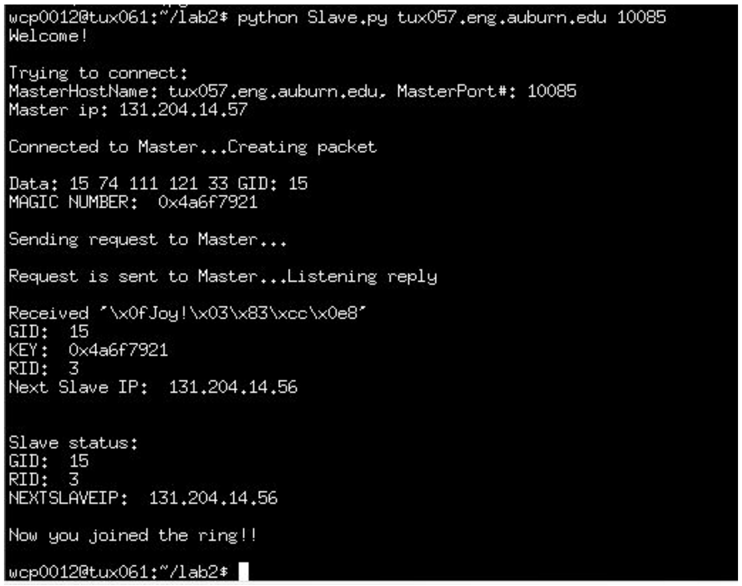
```
szl0093@tux056:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Received '\x0fJoy!\x02\x83\xcc\x0e6'
GID:  15
KEY:  0x4a6f7921
RID:  2
Next Slave IP:  131.204.14.54


Slave status:
GID:  15
RID:  2
NEXTSLAVEIP:  131.204.14.54

Now you joined the ring!!

szl0093@tux056:~/lab2$
```

Figure4(b)



```
Listenning...

server: got connection from 131.204.14.61
server: got packet from 131.204.14.61
server: packet is 5 bytes long
server: packet contains: 0f 4a 6f 79 21
GID: 15
MAGIC KEY: 4a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 3
        Next Slave IP: 131.204.14.56
        KEY: 4a6f7921

server: checking validity
Packet is valid

server: connecting a new slave...
New slave is connected

server: creating packet...
Packet contains:
GID: 15
MAGIC KEY: 4a6f7921
RID: 3
NEXTSLAVEIP: 131.204.14.56
Full packet in hex: 0f 4a 6f 79 21 03 83 cc 0e 38

server: sent packet

Server status after connection:
Server contains:
        GID: 15
        RID: 0
        Next RID: 4
        Next Slave IP: 131.204.14.61
        KEY: 4a6f7921

Listenning...
```

Figure5(a)

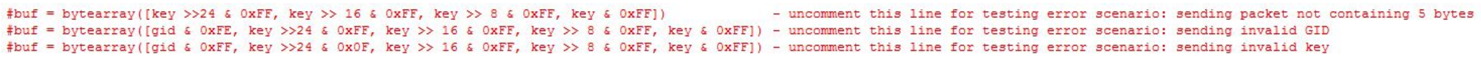


```
wcp0012@tux061:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Received '\x0fJoy!\x03\x83\xcc\x0e8'
GID:  15
KEY:  0x4a6f7921
RID:  3
Next Slave IP:  131.204.14.56


Slave status:
GID:  15
RID:  3
NEXTSLAVEIP:  131.204.14.56

Now you joined the ring!!

wcp0012@tux061:~/lab2$
```

Figure5(b)

- Error examples

  For error testing, we added some lines in our program and commented them. In Slave.py, line 42, line 43, and line 44 are for error testing. To do error testing, uncomment the corresponding line except for the comment portion of the line. Figure6 shows codes for error testing in Slave.py.



```
#buf = bytearray([key >>24 & 0xFF, key >> 16 & 0xFF, key >> 8 & 0xFF, key & 0xFF])           - uncomment this line for testing error scenario: sending packet not containing 5 bytes
#buf = bytearray([gid & 0xFE, key >>24 & 0xFF, key >> 16 & 0xFF, key >> 8 & 0xFF, key & 0xFF]) - uncomment this line for testing error scenario: sending invalid GID
#buf = bytearray([gid & 0xFF, key >>24 & 0x0F, key >> 16 & 0xFF, key >> 8 & 0xFF, key & 0xFF]) - uncomment this line for testing error scenario: sending invalid key
```

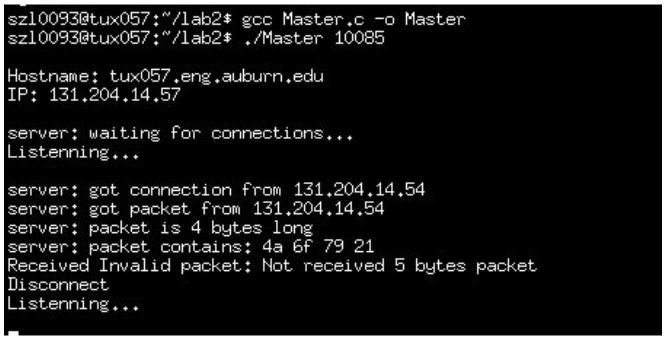
Figure6

- Size of received data in Master program is not 5 bytes.

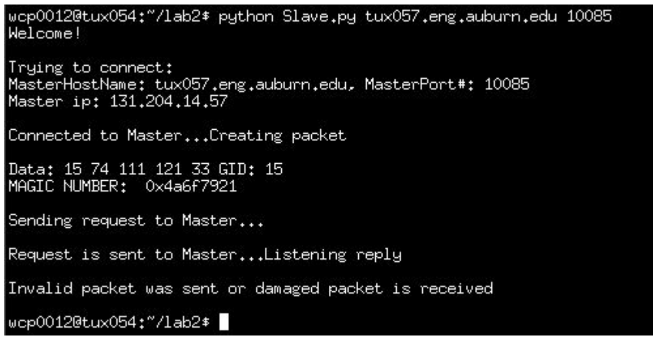
1. First, we uncommented line 42 in Slave.py, which changes size of sending packet to 4 bytes.

2. We compiled and executed Master.c first.

3. We executed Slave.py. Our server system showed messages as Figure7(a), and our client system showed messages as Figure7(b).



```
szl0093@tux057:~/lab2$ gcc Master.c -o Master
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...

server: got connection from 131.204.14.54
server: got packet from 131.204.14.54
server: packet is 4 bytes long
server: packet contains: 4a 6f 79 21
Received Invalid packet: Not received 5 bytes packet
Disconnect
Listenning...
```



```
wcp0012@tux054:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Invalid packet was sent or damaged packet is received

wcp0012@tux054:~/lab2$
```

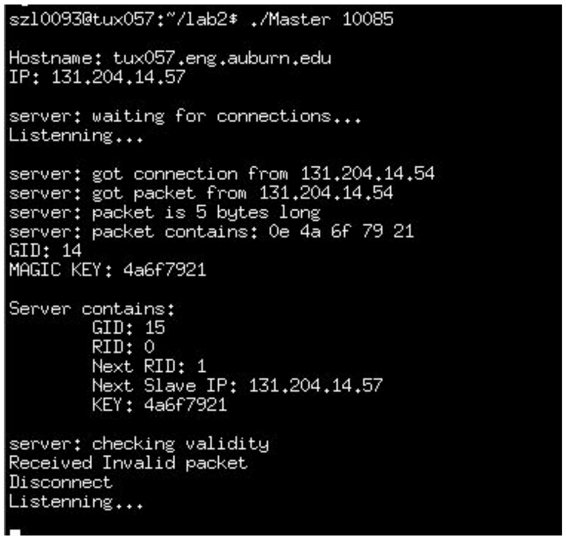
Figure7(a) **Figure7(b)**

- GID from Master and Slave are not matched.

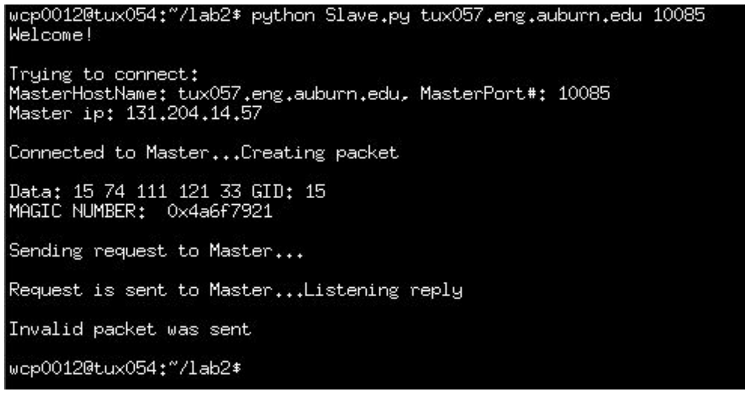
1. First, we uncommented line 43 in Slave.py, which changes GID from 15 to 14.

2.We compiled and executed Master.c first.

3.We executed Slave.py. Our server system showed messages as Figure8(a), and our client system showed messages as Figure8(b).



```
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...

server: got connection from 131.204.14.54
server: got packet from 131.204.14.54
server: packet is 5 bytes long
server: packet contains: 0e 4a 6f 79 21
GID: 14
MAGIC KEY: 4a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 1
        Next Slave IP: 131.204.14.57
        KEY: 4a6f7921

server: checking validity
Received Invalid packet
Disconnect
Listenning...
```



```
wcp0012@tux054:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Invalid packet was sent

wcp0012@tux054:~/lab2$
```

Figure8(a) **Figure8(b)**

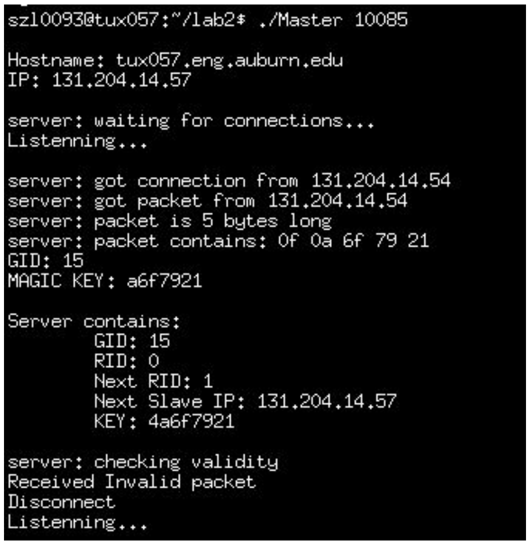
-

- Values in magic number are not valid.

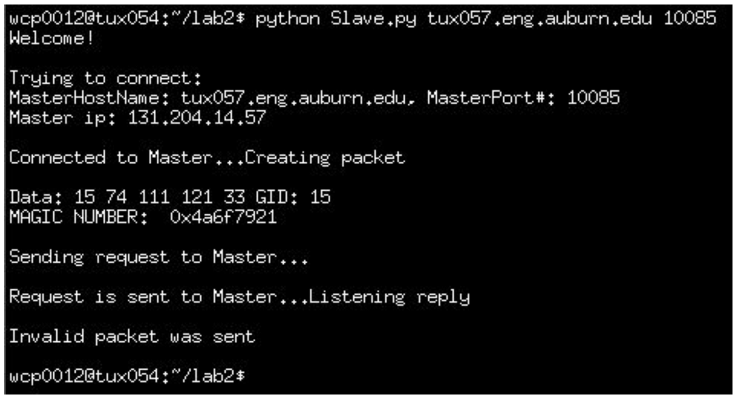
1. First, we uncommented line 44 in Slave.py, which changes magic number from 4A6F7921 to 0A6F7921.

2. We compiled and executed Master.c first.

3. We executed Slave.py. Our server system showed messages as Figure9(a), and our client system showed messages as Figure9(b).



```
szl0093@tux057:~/lab2$ ./Master 10085

Hostname: tux057.eng.auburn.edu
IP: 131.204.14.57

server: waiting for connections...
Listenning...

server: got connection from 131.204.14.54
server: got packet from 131.204.14.54
server: packet is 5 bytes long
server: packet contains: 0f 0a 6f 79 21
GID: 15
MAGIC KEY: a6f7921

Server contains:
        GID: 15
        RID: 0
        Next RID: 1
        Next Slave IP: 131.204.14.57
        KEY: 4a6f7921

server: checking validity
Received Invalid packet
Disconnect
Listenning...
```



```
wcp0012@tux054:~/lab2$ python Slave.py tux057.eng.auburn.edu 10085
Welcome!

Trying to connect:
MasterHostName: tux057.eng.auburn.edu, MasterPort#: 10085
Master ip: 131.204.14.57

Connected to Master...Creating packet

Data: 15 74 111 121 33 GID: 15
MAGIC NUMBER:  0x4a6f7921

Sending request to Master...

Request is sent to Master...Listening reply

Invalid packet was sent

wcp0012@tux054:~/lab2$
```

Figure9(a) **Figure9(b)**

Conclusion

In conclusion, after analyzing a testing our code as described above, we have a working communication between our client and server and a functional setup of a virtual ring made up of the master node and slave nodes in compliance with the lab specifications. Our programs work between different tux machines as we tested many times. Also, we have learned a lot more about the functionality of communications between clients and servers through this lab in addition to the first lab. We were able to simulate the structure of ring made up of slaves (client system) and a master (server system) and get hands on experience with what we learned in class about communications within networks.

References

The main resources we used to guide us in this lab were the assignment specifications, piazza, and the sample source code file that Dr. Biaz provided for us. More specifically, for our server system, Master, we expanded on TCP-server.c sample file.