



# UTHM

Universiti Tun Hussein Onn Malaysia

**UNIVERSITI TUN HUSSEIN ONN MALAYSIA**

**PROJECT REPORT:  
SCRIPTING PROGRAMMING & SIMPLE MESSAGING BETWEEN  
SERVER AND CLIENT(S) IN VIRTUAL MACHINE**

**SEM I 2023/2024**

COURSE NAME	:	OPERATING SYSTEM
COURSE CODE	:	BIC 20803
SECTION NO	:	4
LECTURER	:	TS. DR. MOHD AMIN BIN MOHD YUNUS
STUDENT NAME & MATRIC NO	:	1. NURKHAIRINA BALQIS BINTI MOHD JOE (AI220206) 2. NURUL JANNAH BINTI KAMARUL ZAMAN (AI220147) 3. TUAN KHALIDAH SYAZWANA BINTI TUAN MOHD KASMAWI (AI220118) 4. TUAN NUR RIFAQIAH BINTI TUAN HANIZI (AI220008)

**COGNITIVE: \_\_\_ /5%; PSYCHOMOTOR: \_\_\_ / 10%; AFFECTIVE: \_\_\_/5%**

# TABLE OF CONTENTS

<b>CHAPTER 1: BACKUP FILES.....</b>	<b>3</b>
1.1 Relevance Problem in the Given Task.....	3
1.2 Source Code.....	5
1.3 Code Functionality.....	7
1.4 Outputs and Screenshots.....	9
<b>CHAPTER 2: SIMPLE MESSAGING BETWEEN SERVER AND CLIENT.....</b>	<b>11</b>
2.1 SERVER.....	11
2.1.1 Initiating Server: Command Line for Opening Listening Port with Netcat.....	12
2.1.2 Responding to Client Messages: Handling Incoming Messages.....	13
2.2 CLIENT.....	14
2.2.1 Sending Messages to the Server: Client Interaction.....	15
2.2.2 Connecting to the Server: Command for Initiating Communication.....	16
2.2.3 Message Exchange: Sending and Receiving Messages.....	16
2.3 COMMUNICATION SHOWCASE.....	17
Evaluation Form.....	18

## CHAPTER 1: BACKUP FILES

Write a shell script program that prints a calendar of the current month, current date and time. Record the date the computer last rebooted into a log file named reboot.txt. Then, create backup files every Wednesday or Friday. Display the last modification time of the original file. Run the script to produce 15 backup files (show and list the backup files created). Then add a shell script program that will remove every backup file created in the above statement on Saturday. Make sure to prompt before any removal.

Note: For demo purposes, record the program execution assuming the script will run automatically based on the condition given. For convenience, make the command run automatically for you at the log-in time (save them to batch file to solve automation tasks)

### 1.1 Relevance Problem in the Given Task

During the execution of the provided task or script, several relevance problems or challenges might arise. These issues can impact the successful completion of the task and should be considered during script development and execution. Here are some potential relevance problems:-

- File Path Hardcoding

The possibility of running into issues when the script is run on a machine with a different directory structure is one potential difficulty with hardcoding the path in scripts. The reason for this is that fixed file paths require a certain directory structure that might not be universally applicable. Using variables for file paths inside the script or structuring the script to accommodate different directory structures are suggested solutions to this issue. This method makes the script more flexible and resilient, which lowers the possibility of directory discrepancy problems when the script is executed on multiple platforms.

- User Interaction on Automation

If a script is meant to run automatically without human participation, it may cause issues if it asks users for feedback during automation. The reason for this problem is that automated operations usually need to run well without requiring human intervention. A workable solution to this

issue is to modify the script so that it may respond to user requests in an automated setting. Alternatively, to ensure a more streamlined and dependable automated workflow that doesn't rely on human input, think about incorporating default choices or setting the script to run in a non-interactive mode. This method reduces the possibility of interruptions from user prompts while streamlining the automated process.

- Backup File Overwriting

The problem of backup file overwriting occurs when files with the same name or with insufficiently random naming standards run at risk of being overwritten. The integrity and retrieval of data during backup procedures are at risk because of this circumstance. Implementing a naming convention that ensures backup file uniqueness is a suggested method to lessen this problem. A good way to go about this is to include timestamps in the filenames so that every backup file has a unique number that corresponds to the moment it was created. By giving each backup instance a distinct and easily identifiable identity, this technique reduces the possibility of overwriting and improves the dependability of the backup system.

- Error Handling

Inadequate error handling is a prevalent problem in scripting, and it can lead to unexpected behavior. Problems that occur during script execution may be harder to find and fix if there isn't a strong error-handling system in place. It is imperative to incorporate extensive error-handling methods within the script in order to handle this risk. This entails foreseeing probable mistakes and creating protocols to manage them tactfully. Scripts that integrate efficient error handling can yield more lucid diagnostic data, streamline troubleshooting, and guarantee a more dependable and sturdy performance, especially in the face of unforeseen circumstances.

## 1.2 Source Code

Source code is the human-readable set of instructions written in a programming language, forming the basis of a computer program. It is translated by compilers or interpreters into machine code for computer execution, and its clarity and structure impact software functionality and maintainability.

```
#!/bin/bash

# Print calendar of the current month
echo "Calendar of the Current Month:"
cal

# Print current date and time
echo "Current Date and Time:"
date

# Record the date of the last reboot to reboot.txt
echo "Date of the Last Reboot: $(who -b)" >> repeat.txt

# Check if today is Wednesday or Friday
day=$(date +%A)
if [[ "$day" = "Wednesday" || "$day" = "Friday" ]]; then
    # Create backup files
    echo "Creating backup files for 15 backup files:-"
    for ((i=1; i≤15; i++)); do
        timestamp=$(date +%Y%m%d%H%M%S)
        filename="backup_${timestamp}.txt"
        cp repeat.txt "$filename"
        echo "$filename created"
    done
fi
```

Figure 1.2 (a): Shell script for repeat.txt part backup file

```

# Display the last modification time of the original file
echo "Last Modification Time of repeat.txt:"
stat -c "%y" repeat.txt

# Remove backup files on Saturday
if [[ "$day" = "Saturday" ]]; then
    echo "Today is Saturday. Removing backup files..."
    for file in backup_*.txt; do
        read -p "Are you sure you want to remove $file? (y/n): " choice
        if [[ "$choice" = "y" ]]; then
            rm "$file"
            echo "$file removed"
        fi
    done
fi

```

Figure 1.2 (b): Shell script for repeat.txt part remove file

```

#run the backup script every wednesday or friday at 0 am
0 0 * * 3,5 ./new.txt

#run the remove script every Saturday at 0 am
0 0 * * 6./remove.sh "y"

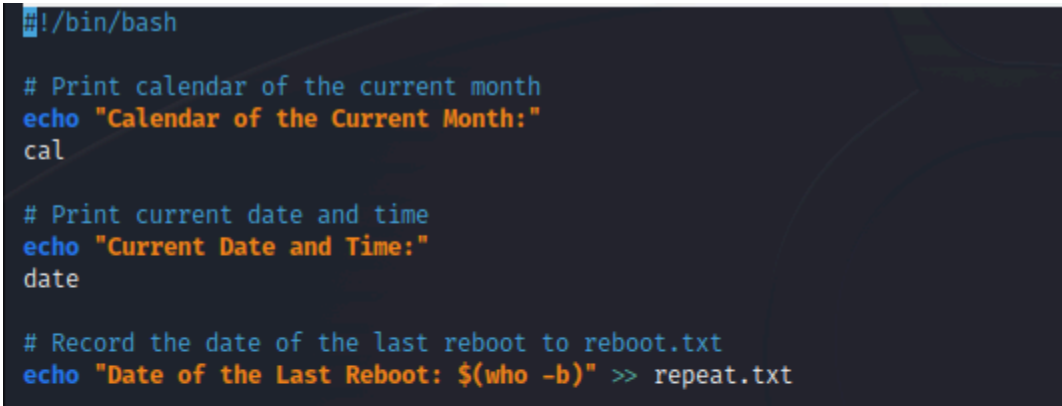
```

Figure 1.2 (c): Backup script and removal script scheduling using crontab -e command

### 1.3 Code Functionality

Code functionality refers to the operational capabilities and features defined within a computer program, specifying the tasks it can perform and the outcomes it can produce. It is a critical aspect of software development, determining the behavior and effectiveness of the resulting application.

#### a. Backup file script



```
#!/bin/bash

# Print calendar of the current month
echo "Calendar of the Current Month:"
cal

# Print current date and time
echo "Current Date and Time:"
date

# Record the date of the last reboot to reboot.txt
echo "Date of the Last Reboot: $(who -b)" >> repeat.txt
```

*Figure 1.3(a): Backup script for print and record.*

The first line `#!/bin/bash` in Figure 1.3(a), specifies the interpreter to be used for executing the script, which is the Bash shell. The script starts by printing the calendar of the current month using the `cal` command. The output will be displayed with the message "Calendar of the Current Month:". Next, the script will print the current date and time using the `date` command. The output will be displayed with the message "Current Date and Time:". The date of the last reboot is recorded using the `who -b` command, and the output is appended to the file "repeat.txt" with the message "Date of the Last Reboot:".

```

# Check if today is Wednesday or Friday
day=$(date +%A)
if [[ "$day" = "Wednesday" || "$day" = "Friday" ]]; then
    # Create backup files
    echo "Creating backup files for 15 backup files:-"
    for ((i=1; i≤15; i++)); do
        timestamp=$(date +%Y%m%d%H%M%S)
        filename="backup_${timestamp}.txt"
        cp repeat.txt "$filename"
        echo "$filename created"
    done
fi

```

Figure 1.3(b): Backup script for check day is and create backup files

As shown in Figure 1.3(b), the script checks if today is Wednesday or Friday by obtaining the day of the week using `date +%A` and storing it in the variable `day`. It then uses an `if` statement to execute the following block of code only if the day is "Wednesday" or "Friday". Inside the `if` block, a loop is used to create backup files. The loop runs 15 times, and in each iteration, a unique timestamp is generated using `date +%Y%m%d%H%M%S`. The timestamp is appended to the filename, and the `cp` command is used to copy the "repeat.txt" file to the backup file with the generated name. A message is displayed for each backup file created.

b. Remove file script

```

# Display the last modification time of the original file
echo "Last Modification Time of repeat.txt:"
stat -c "%y" repeat.txt

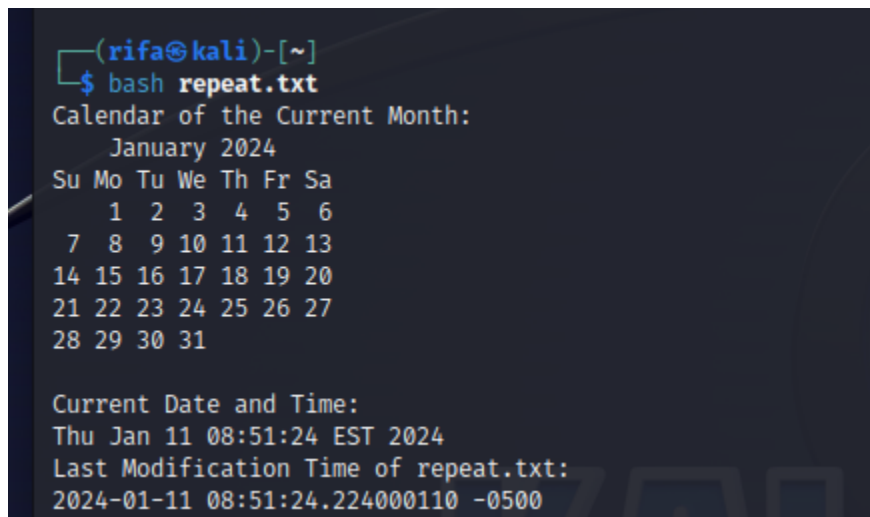
# Remove backup files on Saturday
if [[ "$day" = "Saturday" ]]; then
    echo "Today is Saturday. Removing backup files..."
    for file in backup_*.txt; do
        read -p "Are you sure you want to remove $file? (y/n): " choice
        if [[ "$choice" = "y" ]]; then
            rm "$file"
            echo "$file removed"
        fi
    done
fi

```

Figure 1.3(c): Display last modification and remove backup files script

After the if block, in Figure 1.3(b) the script uses the `stat -c` command to display the last modification time of the "repeat.txt" file. As Figure 1.3(c), if block checks if today is Saturday. If it is, the script proceeds to remove backup files. It uses a loop to iterate over all files in the current directory matching the pattern "backup\_\*.txt". For each file, it prompts the user with a confirmation message and waits for a response using the `read` command. If the user enters "y", the file is removed with the `rm` command, and a message is displayed.

## 1.4 Outputs and Screenshots



```
(rifa@kali)-[~]
$ bash repeat.txt
Calendar of the Current Month:
  January 2024
Su Mo Tu We Th Fr Sa
   1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29 30 31

Current Date and Time:
Thu Jan 11 08:51:24 EST 2024
Last Modification Time of repeat.txt:
2024-01-11 08:51:24.224000110 -0500
```

*Figure 1.4(a): The output if today is not Wednesday, Friday, or Saturday*

```

(rifa@kali)-[~]
$ bash repeat.txt
Calendar of the Current Month:
January 2024
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

Current Date and Time:
Wed Jan 10 08:50:49 EST 2024
Creating backup files for 15 backup files:-
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085049.txt created
backup_20240110085050.txt created
backup_20240110085050.txt created
backup_20240110085050.txt created
backup_20240110085050.txt created
Last Modification Time of repeat.txt:
2024-01-10 08:50:49.046000116 -0500

```

Figure 1.4(b): The output if today is Wednesday or Friday

```

(rifa@kali)-[~]
$ ./repeat.txt
Calendar of the Current Month:
January 2024
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

Current Date and Time:
Sat Jan 13 09:15:46 EST 2024
Last Modification Time of repeat.txt:
2024-01-13 09:15:46.408278506 -0500
Today is Saturday. Removing backup files...
Are you sure you want to remove backup_20240110090936.txt? (y/n): y
backup_20240110090936.txt removed
Are you sure you want to remove backup_20240110090937.txt? (y/n): y
backup_20240110090937.txt removed
Are you sure you want to remove backup_20240110090938.txt? (y/n): y
backup_20240110090938.txt removed
Are you sure you want to remove backup_20240110091426.txt? (y/n): n
Are you sure you want to remove backup_20240110091427.txt? (y/n): y
backup_20240110091427.txt removed

```

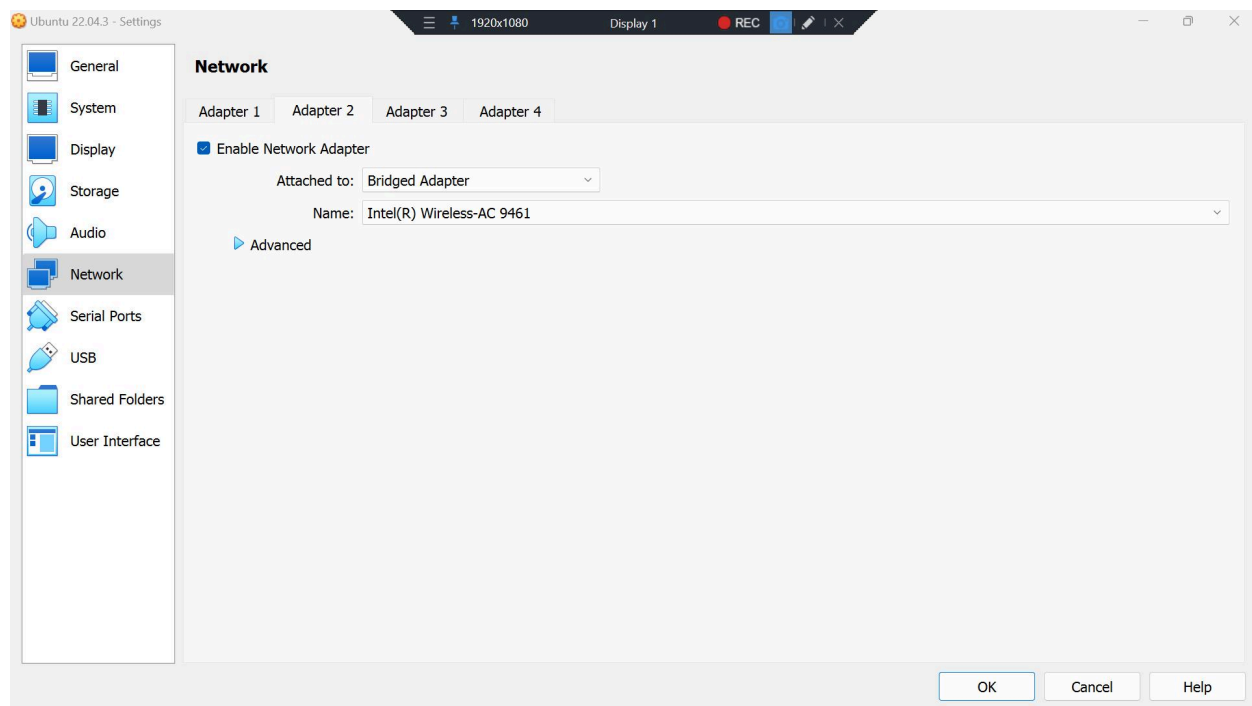
Figure 1.4(c): The output if today is Saturday

## CHAPTER 2: SIMPLE MESSAGING BETWEEN SERVER AND CLIENT

Simple messaging between a server and a client is the fundamental building block of communication in the field of computer networking. Concise messages are exchanged throughout this procedure, which makes it easier for data to be sent from a central server to linked clients and vice versa. This kind of simplicity not only increases system performance but also creates the foundation for a wide range of applications, from simple chat features to more complex data transfers.

### 2.1 SERVER

When it comes to server functionality, the first step in communicating is to use the command line—that is, Netcat—to open a listening port. The server may now receive incoming messages from connected clients thanks to this fundamental step. The virtual machine that we used as a server is Ubuntu. The virtual machine used as the server runs on Ubuntu. Firstly, ensure that both the server and client share the same Wi-Fi connection. Then, on Ubuntu, adjust the network settings to use a bridge adapter, especially when working with two laptops to complete this project.

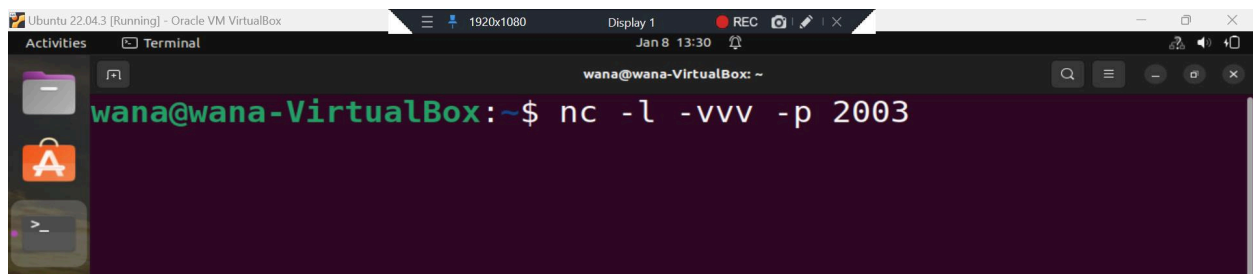


*Figure 2.1: The Settings of Bridged Adapter network on Ubuntu*

### 2.1.1 Initiating Server: Command Line for Opening Listening Port with Netcat

The first step is to start a server, which may be done via the command line by using Netcat to open a listening port. This command-driven procedure prepares the server to actively receive and handle incoming messages.

To initiate this project, open the terminal in Ubuntu and enter the command 'nc -l -vvv -p 2003'. Here, 'nc' is the command for netcat, a versatile networking utility facilitating reading from and writing to network connections. The '-l' option instructs netcat to operate in listening mode, awaiting incoming connections instead of initiating them. The '-vvv' option elevates the verbosity level, providing more detailed output, which, in this case, entails comprehensive information about connections and data transfer. The '-p 2003' option specifies the port number on which netcat will listen for incoming connections, set to port 2003 in this instance. Finally, press Enter to execute the command.

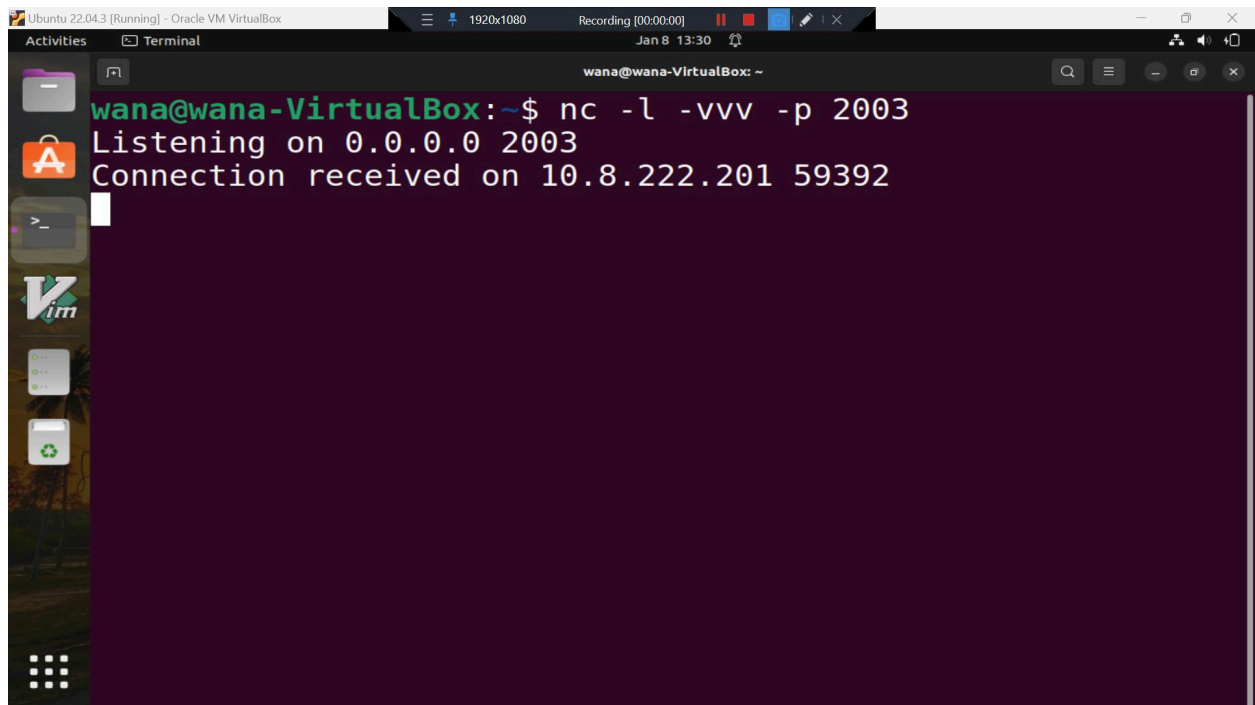


*Figure 2.1.1: Command Line for Opening Listening Port with Netcat*

### 2.1.2 Responding to Client Messages: Handling Incoming Messages

Reacting to client communications within the domain of the server requires efficient internal communication management. This entails putting in place systems to interpret messages from linked clients and respond accordingly.

After entering the command on both the client and server, the screen in the server terminal will be displayed as follows. This serves as proof that the server and client are already successfully connected to each other.

A screenshot of a terminal window within an Oracle VM VirtualBox environment. The window title is 'Ubuntu 22.04.3 [Running] - Oracle VM VirtualBox'. The terminal shows the command 'nc -l -vvv -p 2003' being executed. The output indicates the server is listening on 0.0.0.0 port 2003 and has received a connection from 10.8.222.201 on port 59392. The terminal background is dark purple with green and white text. The window includes standard Ubuntu desktop elements like a sidebar with application icons and a top status bar.

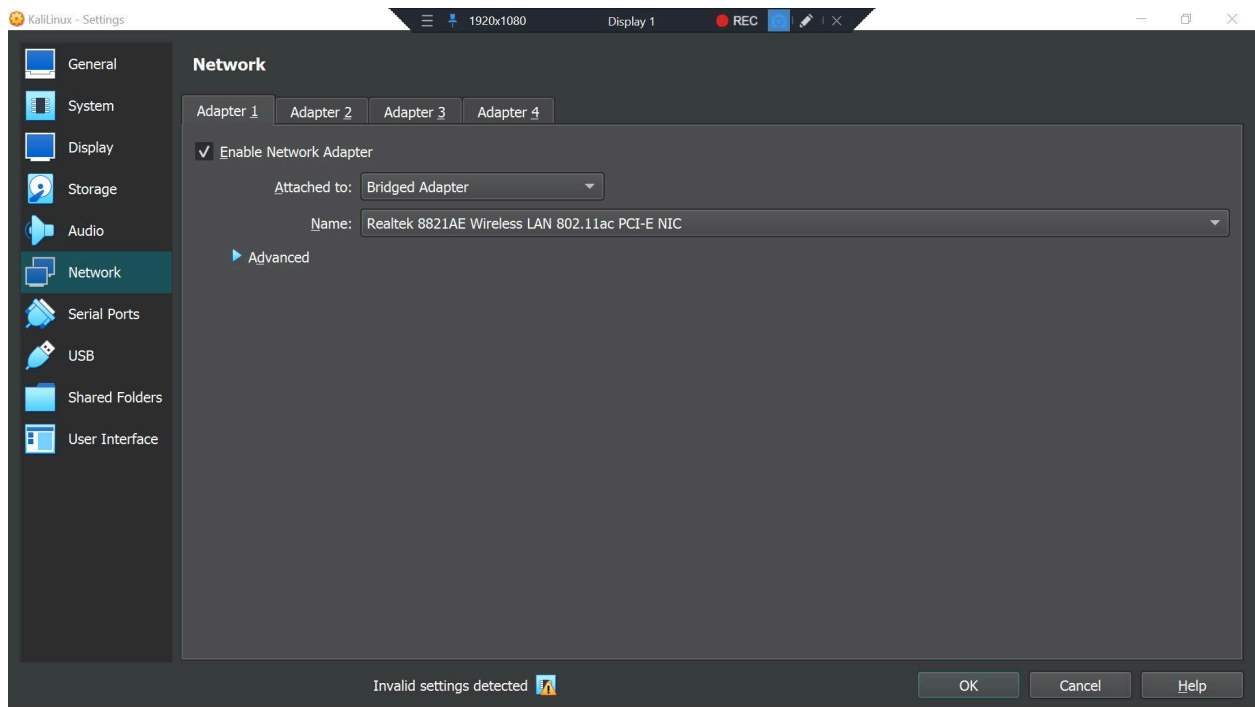
```
wana@wana-VirtualBox:~$ nc -l -vvv -p 2003
Listening on 0.0.0.0 2003
Connection received on 10.8.222.201 59392
```

*Figure 2.1.2: Handling Incoming Messages*

## 2.2 CLIENT

Messages are sent and conversations are established in order to promote client-server interaction. The instructions for starting a conversation, sending messages to the server, and understanding all aspects of message exchange are covered in this section.

On Kali Linux, it is necessary to modify the network settings to use a bridge adapter. Open the settings, navigate to the Network section, make the necessary network changes, and then click 'OK'.

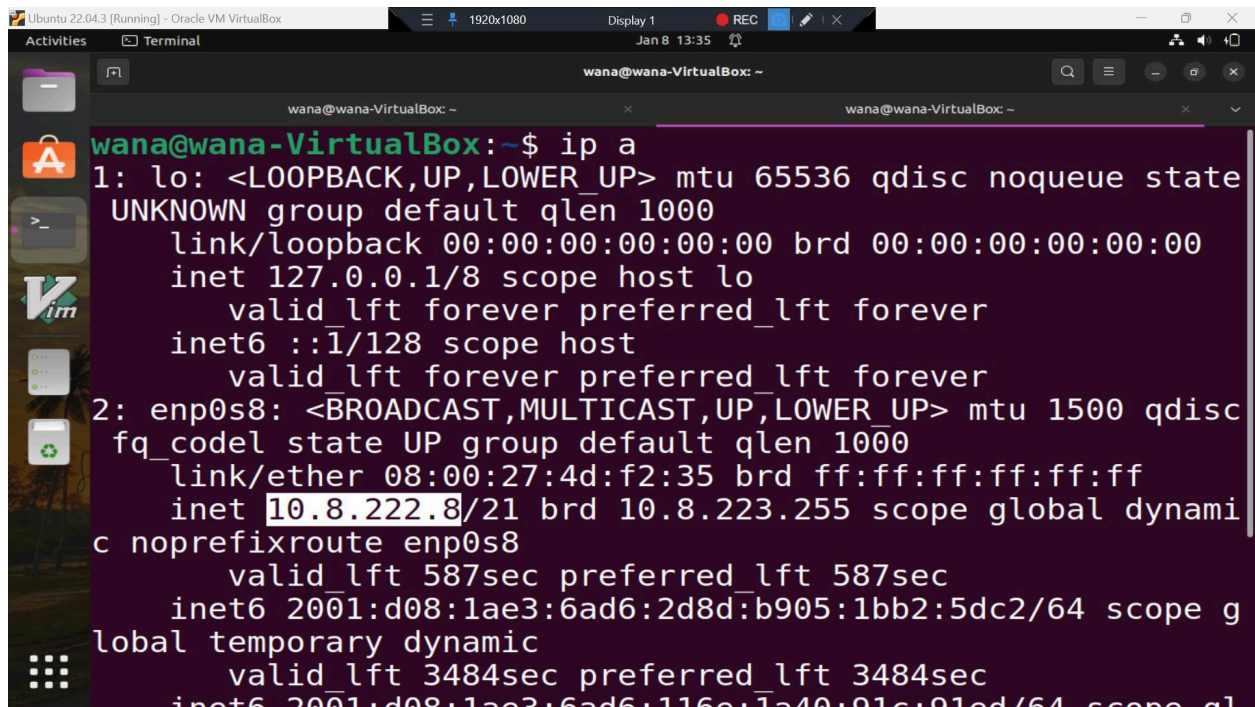


*Figure 2.2: The Settings of Bridged Adapter network on Kali Linux.*

### 2.2.1 Sending Messages to the Server: Client Interaction

Sending messages to the server is the main way that clients communicate with it. It is essential to understand the command structure for this interaction since it determines how clients connect with the server and exchange data.

To send a message to the client, it is essential to determine the server's IP address. To retrieve this information, enter the command 'ip a'. The IP address will then appear on the same line as 'inet'.



```
wana@wana-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
   UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
   fq_codel state UP group default qlen 1000
    link/ether 08:00:27:4d:f2:35 brd ff:ff:ff:ff:ff:ff
    inet 10.8.222.8/21 brd 10.8.223.255 scope global dynamic
        c noprefixroute enp0s8
            valid_lft 587sec preferred_lft 587sec
    inet6 2001:d08:1ae3:6ad6:2d8d:b905:1bb2:5dc2/64 scope g
        lobal temporary dynamic
            valid_lft 3484sec preferred_lft 3484sec
    inet6 2001:d08:1ae3:6ad6:116e:1a40:91c:91ed/64 scope gl
```

Figure 2.2.1: Server IP Address

### 2.2.2 Connecting to the Server: Command for Initiating Communication

Connecting to the server is an initial step in the communication process for the client. This requires following specific commands to connect and open a pathway for the client and server to exchange messages later on.

To connect to the server, the Kali Linux client needs to execute the command 'nc 10.8.222.8 2003'. Here, 'nc' represents netcat, '10.8.222.8' is the IP address, and '2003' is the port number configured for this project.



*Figure 2.2.2: Command for Initiating Communication*

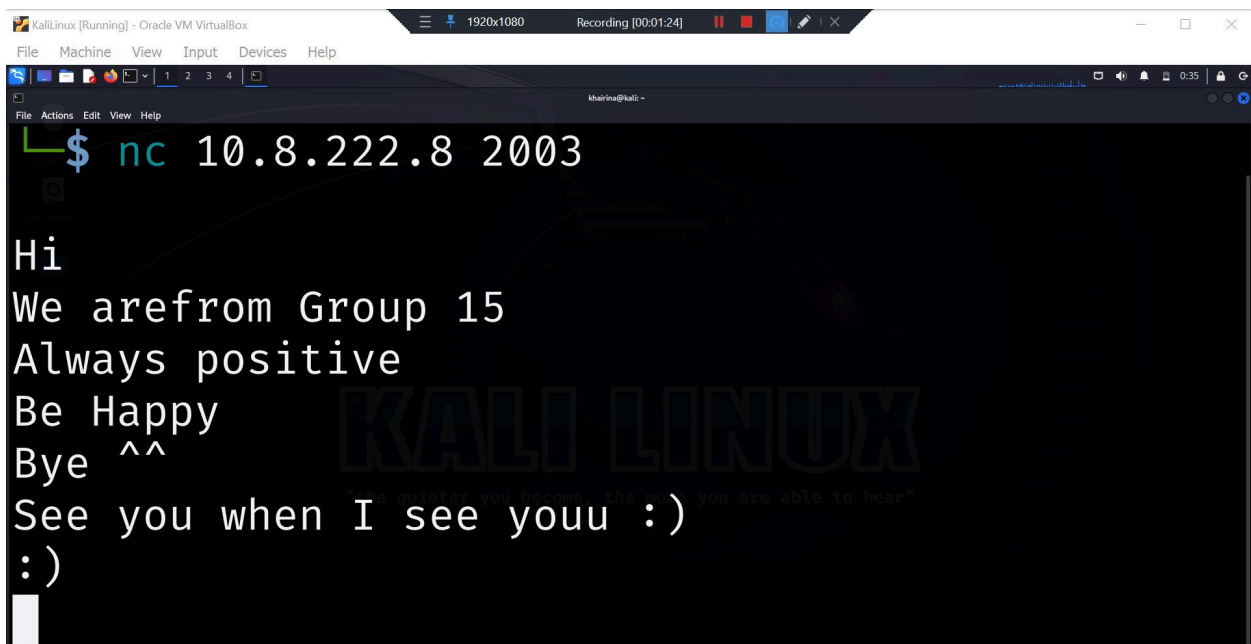
### 2.2.3 Message Exchange: Sending and Receiving Messages

Messages are the central component of client-server communication. In addition to sending messages to the server, clients also receive and handle incoming messages. The fundamental component of a dynamic client-server interaction is this exchange of information.

After executing all the commands in both Ubuntu and Kali Linux, message exchange becomes possible. Users can type messages in the client, and these messages will also appear on the server, and vice versa.

## 2.3 COMMUNICATION SHOWCASE

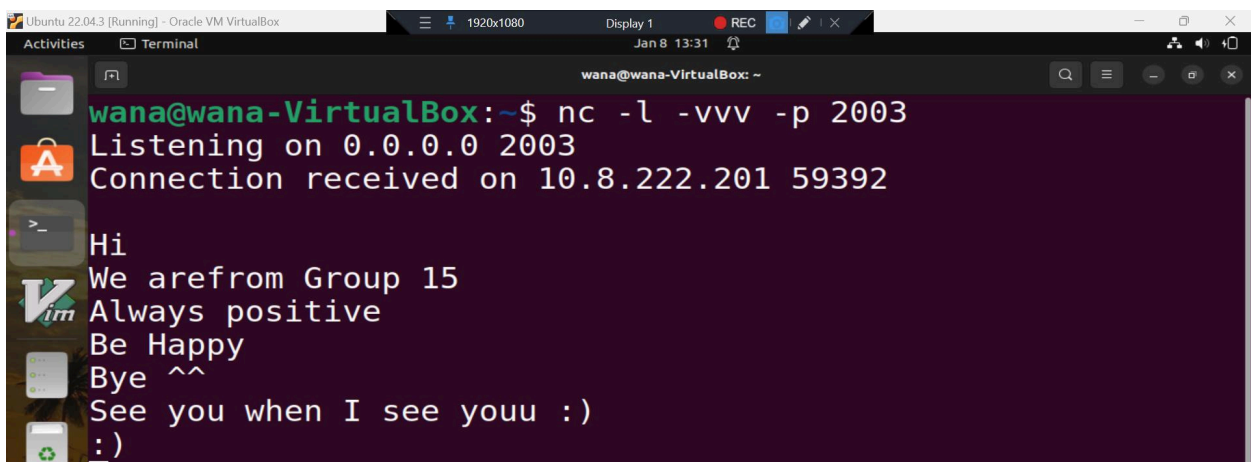
The Communication Showcase section offers an operational demonstration of the complete message process to show how server-client communication terminates. This demonstrates the easy exchange of messages between the clients and the server, highlighting the successful utilization of the defined communication standards. Below are screenshots from the server terminal and the client terminal, showcasing the successful exchange of messages from the server to the client.



The screenshot shows a Kali Linux terminal window titled "KaliLinux [Running] - Oracle VM VirtualBox". The terminal displays the following text:

```
$ nc 10.8.222.8 2003  
Hi  
We arefrom Group 15  
Always positive  
Be Happy  
Bye ^^  
See you when I see youu :)  
:)
```

Figure 2.3 (a): Messages Displayed in the Client, which is running on Kali Linux.



The screenshot shows an Ubuntu terminal window titled "Ubuntu 22.04.3 [Running] - Oracle VM VirtualBox". The terminal displays the following text:

```
wana@wana-VirtualBox:~$ nc -l -vvv -p 2003  
Listening on 0.0.0.0 2003  
Connection received on 10.8.222.201 59392  
Hi  
We arefrom Group 15  
Always positive  
Be Happy  
Bye ^^  
See you when I see youu :)  
:)
```

Figure 2.3(b): Messages inputted on the server, which is running on Ubuntu.

## APPENDIX

### Evaluation Form

NO.	NAME	MATRIC NUMBER	TOTAL MARKS (A + B + C)
1.	NURKHAIRINA BALQIS BINTI MOHD JOE	AI220206	
2.	NURUL JANNAH BINTI KAMARUL ZAMAN	AI220147	
3.	TUAN KHALIDAH SYAZWANA BINTI TUAN MOHD KASMAWI	AI220118	
4.	TUAN NUR RIFAQIAH BINTI TUAN HANIZI	AI220008	

#### A. Documentation – Project Report (COGNITIVE)

Criteria	Rating	Marks
Explain the relevance problem in the given task - Chapter 1 & 2 [C2]	<input type="checkbox"/> ①②③④⑤	
Clearly, confidently, and effectively produce such Input/Output redirection, if else, and basic statements - Chapter 1 [C4]	<input type="checkbox"/> ①②③④⑤	
Analyze important points - Chapter 2 [C4]	<input type="checkbox"/> ①②③④⑤	
Discover the critical and constructive solutions - Chapter 1 & 2 [C4]	<input type="checkbox"/> ①②③④⑤	
Outline the references & format adhere to UTHM thesis format. [C2]	<input type="checkbox"/> ①②③④⑤	
<b>Total</b>		[    /25    ] *5 =

#### B. Technical Skills (PSYCHOMOTOR)

Criteria	Rating	Marks
Execute - Successfully undertake Task 1 (hypervisor). [P3]	<input type="checkbox"/> ①②③④⑤	
Practice - Successfully undertake Task 2 (two guest OS). [P4]	<input type="checkbox"/> ①②③④⑤	
Practice- Successfully for: [P4] <ul style="list-style-type: none"> <li>- Task 3(a) – executing the scripts to backup files</li> <li>- Task 3(b-i) – executing the command to start the server to open the port for listening and replying to the client</li> <li>- Task 3(b-ii) – executing the command to start messaging the server and reply to the server</li> </ul>	<input type="checkbox"/> ①②③④⑤	
Demonstrate - flow of Task 3(a & b). [P3]	<input type="checkbox"/> ①②③④⑤	

<b>Total</b>	[    /20    ] *10 =
--------------	---------------------

**C. Presentation (AFFECTIVE)**

Criteria	Rating	Marks
Organize presentation in the given time. [A1]	<input type="checkbox"/> ①②③④⑤	
Demonstrate the ability to present clearly and confidently. [A2]	<input type="checkbox"/> ①②③④⑤	
Demonstrate importance information as specified in project question. [A4]	<input type="checkbox"/> ①②③④⑤	
Demonstrate the ability to handle Q n A session effectively and giving respond. [A3]	<input type="checkbox"/> ①②③④⑤	
Autonomy & Responsibility (Relationship building): Participation of group members (group commitment/ cooperation). [A3] <ol style="list-style-type: none"> <li>1. (Very week) – Not able to work in a team</li> <li>2. (Week) – Poor ability of : Teamwork; Collaboration in reaching consensus on an issue</li> <li>3. (Fair) – Satisfactory ability of : Teamwork; Collaboration in reaching consensus on an issue</li> <li>4. (Good) – Good ability of : Teamwork; Collaboration in reaching consensus on an issue</li> <li>5. (Very good) – Excellent ability of : Teamwork; Collaboration in reaching consensus on an issue</li> </ol>	<input type="checkbox"/> ①②③④⑤	
<b>Total</b>		[    /25    ] *5 =