**Southern Luzon State University**

**College of Engineering**

CPE Department

**“OPERATING SYSTEM: LINUX & WINDOWS”**

in Partial Fulfillment of the Requirements

for CPE17L – Operating System Lab

By:

CORPUZ, Reimarc G.

DIMAYUGA, Lyka

BSCPE III – GE

Date of submission:

June \_\_\_, 2023

INTRODUCTION

This documentation paper explores the implementation of a cross-platform network communication system between two machines with different operating systems. The primary objective is to establish a connection that allows for the sharing and execution of files between the machines, all within the confines of a local area network (LAN). The setup can be configured as either peer-to-peer or client-server, providing flexibility to the users.

Additionally, this documentation covers the execution of a system call program and the utilization of a CPU scheduling technique on both machines. These functionalities will be implemented and tested to ensure seamless operation on each platform.

The primary focus of this documentation is to enable the sharing and execution of files between the two machines (LINUX and WINDOWS). To achieve this, a shared folder accessible by both machines will be established. The communication setup offers the flexibility to choose between a peer-to-peer or client-server architecture, allowing users to select the configuration that best suits their requirements.

METHODOLOGY

Methodology:

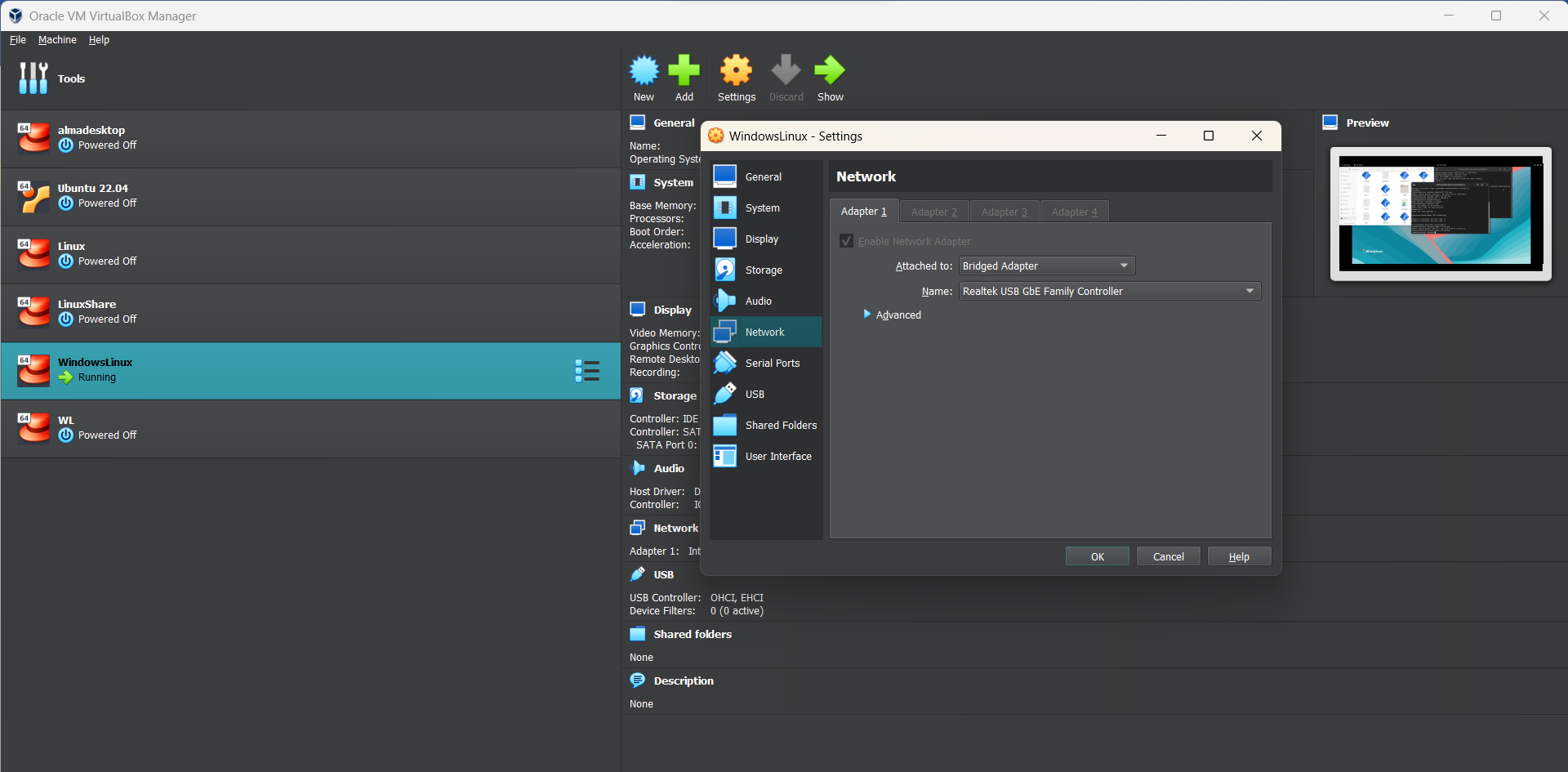
1. Setting up the Virtual Machines:

a. Install VirtualBox on a host machine capable of running virtualization software.

b. Create two virtual machines, one running Windows and the other running Linux, within the VirtualBox environment.

c. Configure the network settings of the virtual machines to ensure they are connected using the LAN cable to which the Network is attached in Bridge Adapter.

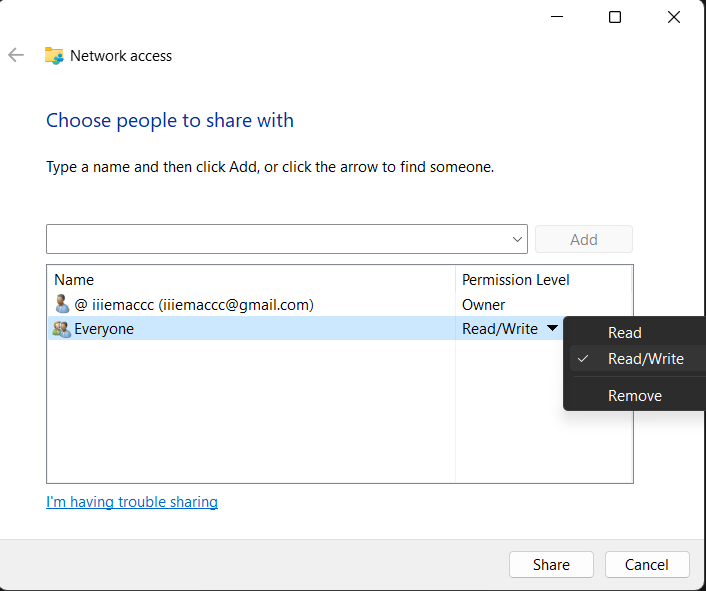
d. Install the respective operating systems (Windows and Linux) on their corresponding virtual machines.

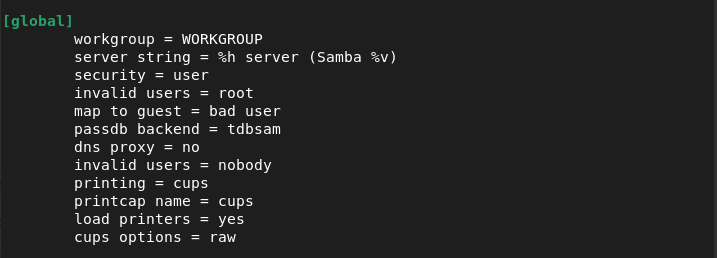


1. Configuring Shared Folder:

a. Identify a folder on the host machine that will serve as the shared folder.

b. In the VirtualBox settings for each virtual machine, enable shared folders and add the identified folder as a shared folder accessible by both virtual machines.

c. Within each operating system, configure the necessary settings to mount the shared folder, ensuring it is visible and accessible to both machines.





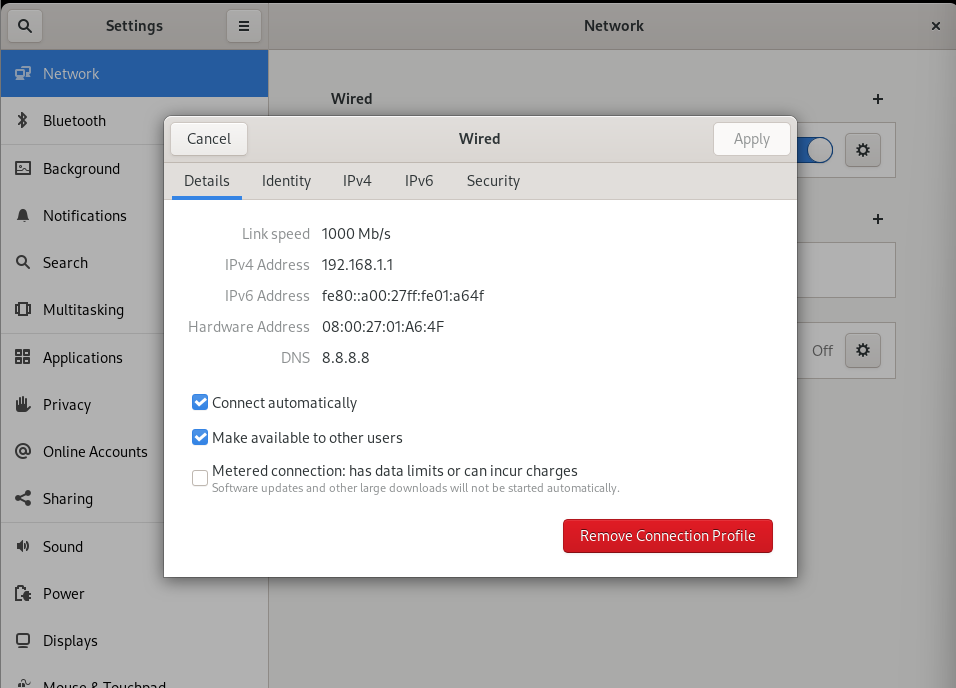
1. Establishing Network Communication:

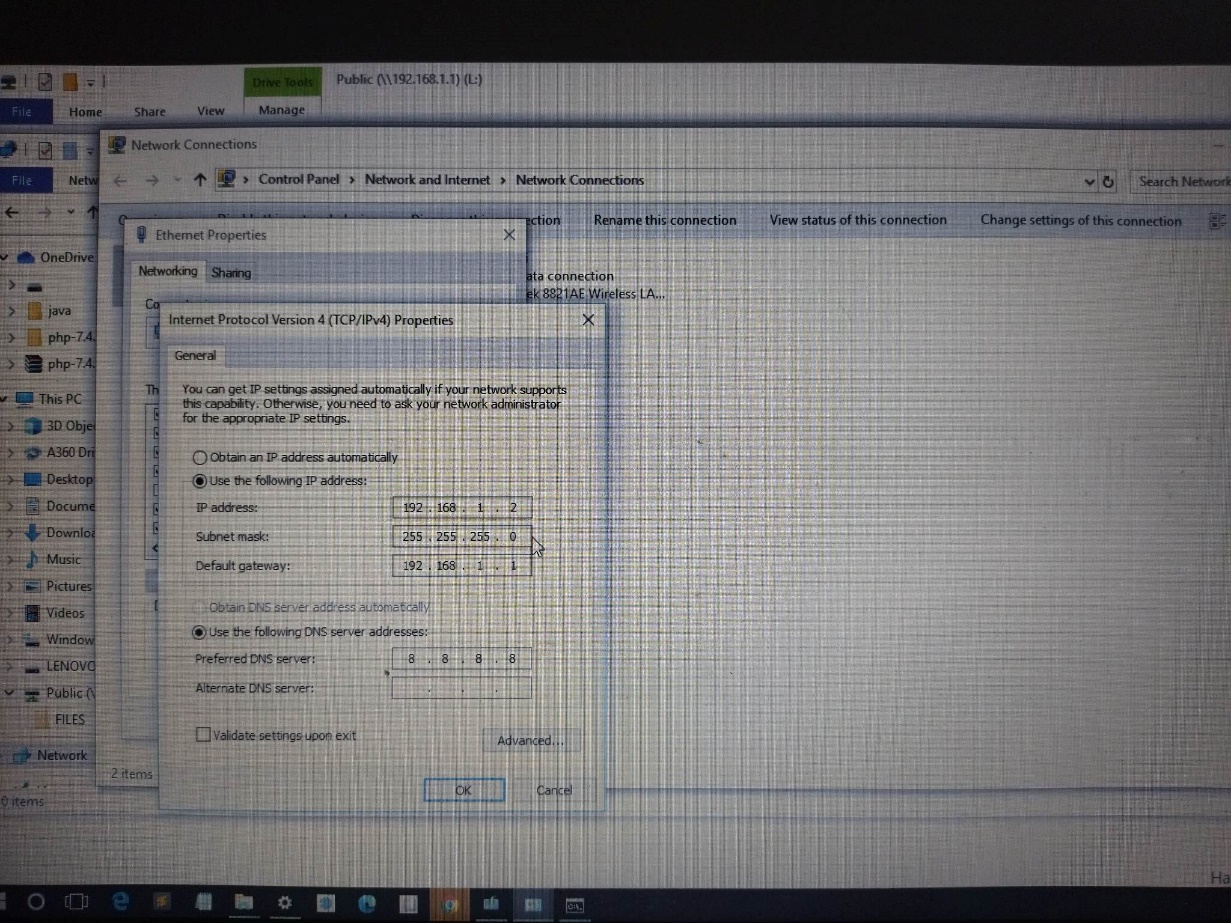
a. Determine the preferred communication architecture: peer-to-peer or client-server.

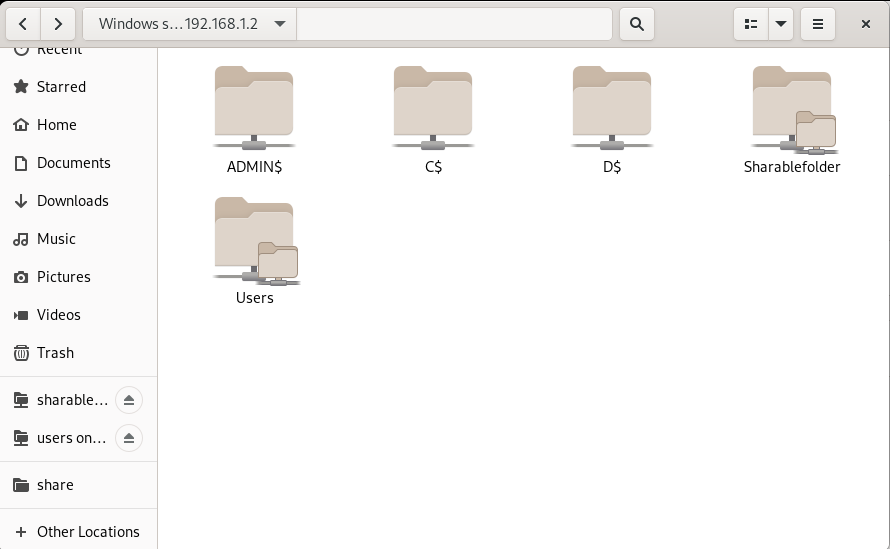
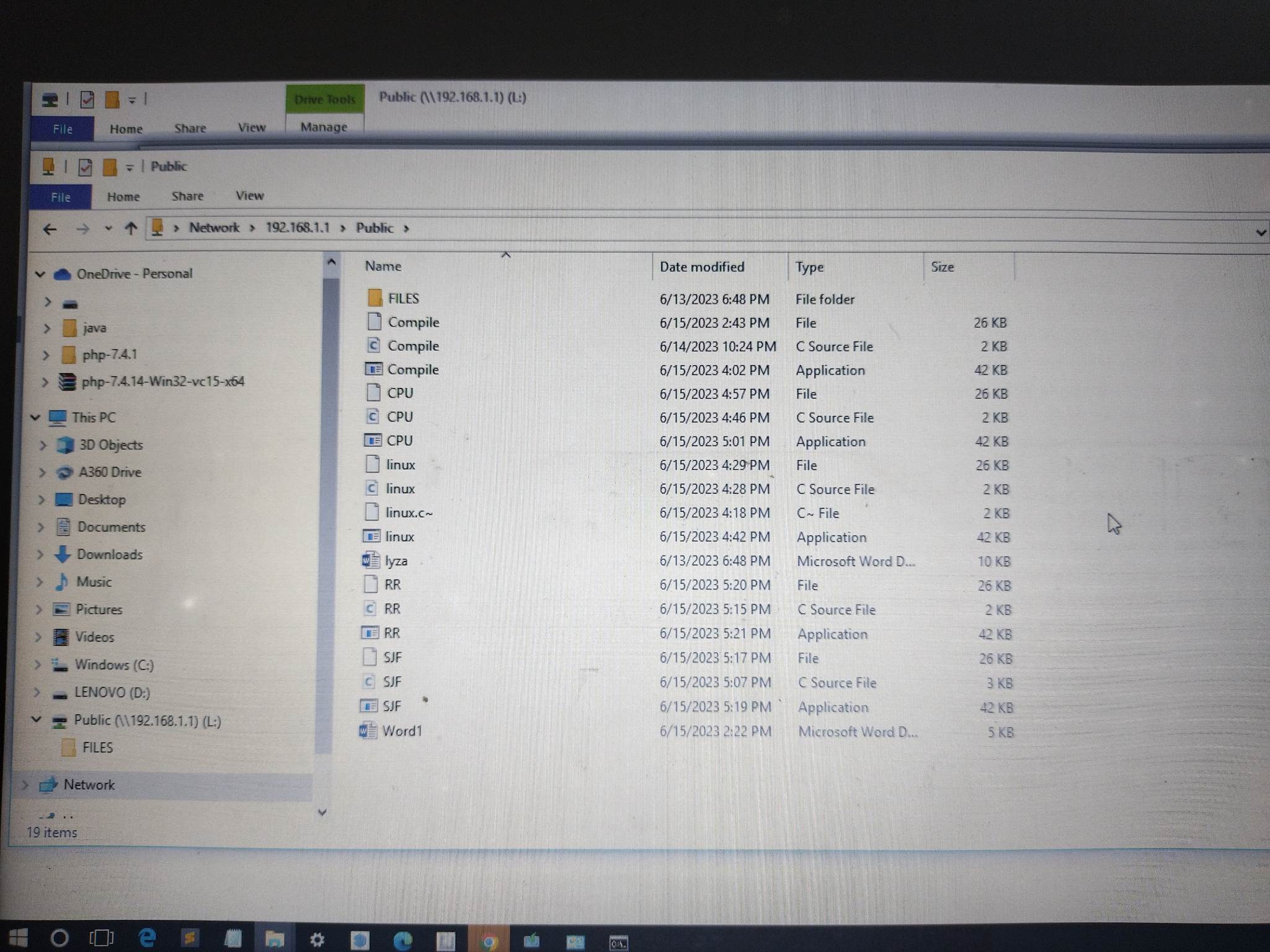
b. Configure the network settings on both virtual machines accordingly, ensuring they have unique IP addresses within the LAN.

c. Implement the chosen communication architecture, using appropriate protocols and libraries specific to each operating system. This may involve configuring network shares, setting up TCP/IP sockets, or utilizing network communication APIs.

d. Test the network communication between the virtual machines by creating, deleting, and editing files to ensure successful connectivity.





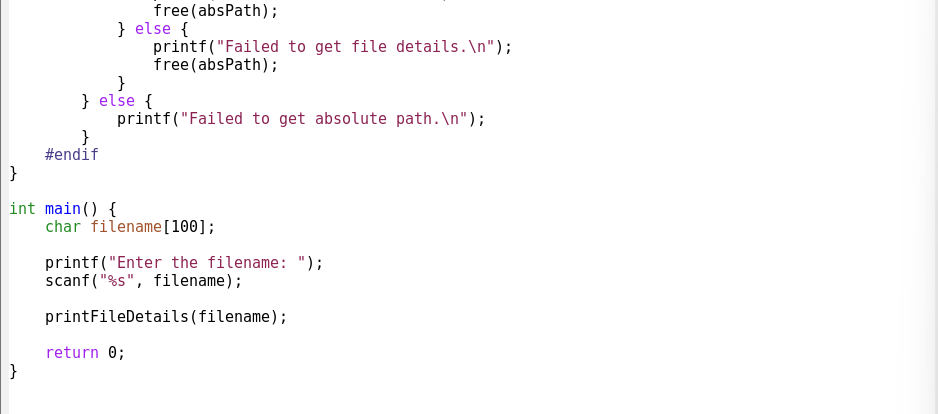
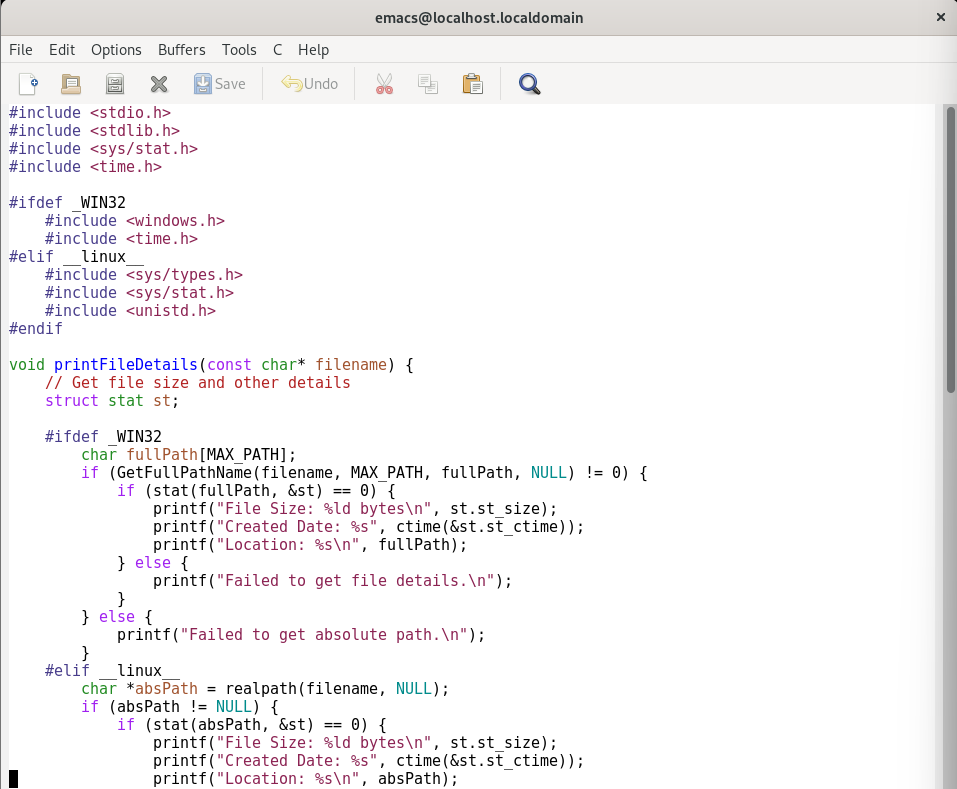
1. System Call Program Execution:

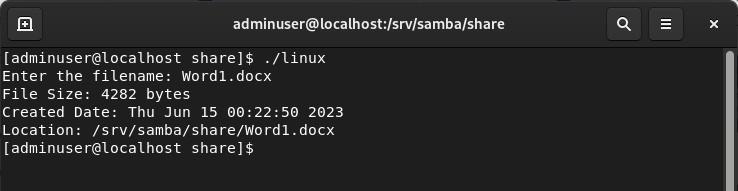
a. Develop a system call program compatible with both Windows and Linux operating systems. This may involve using programming languages like C.

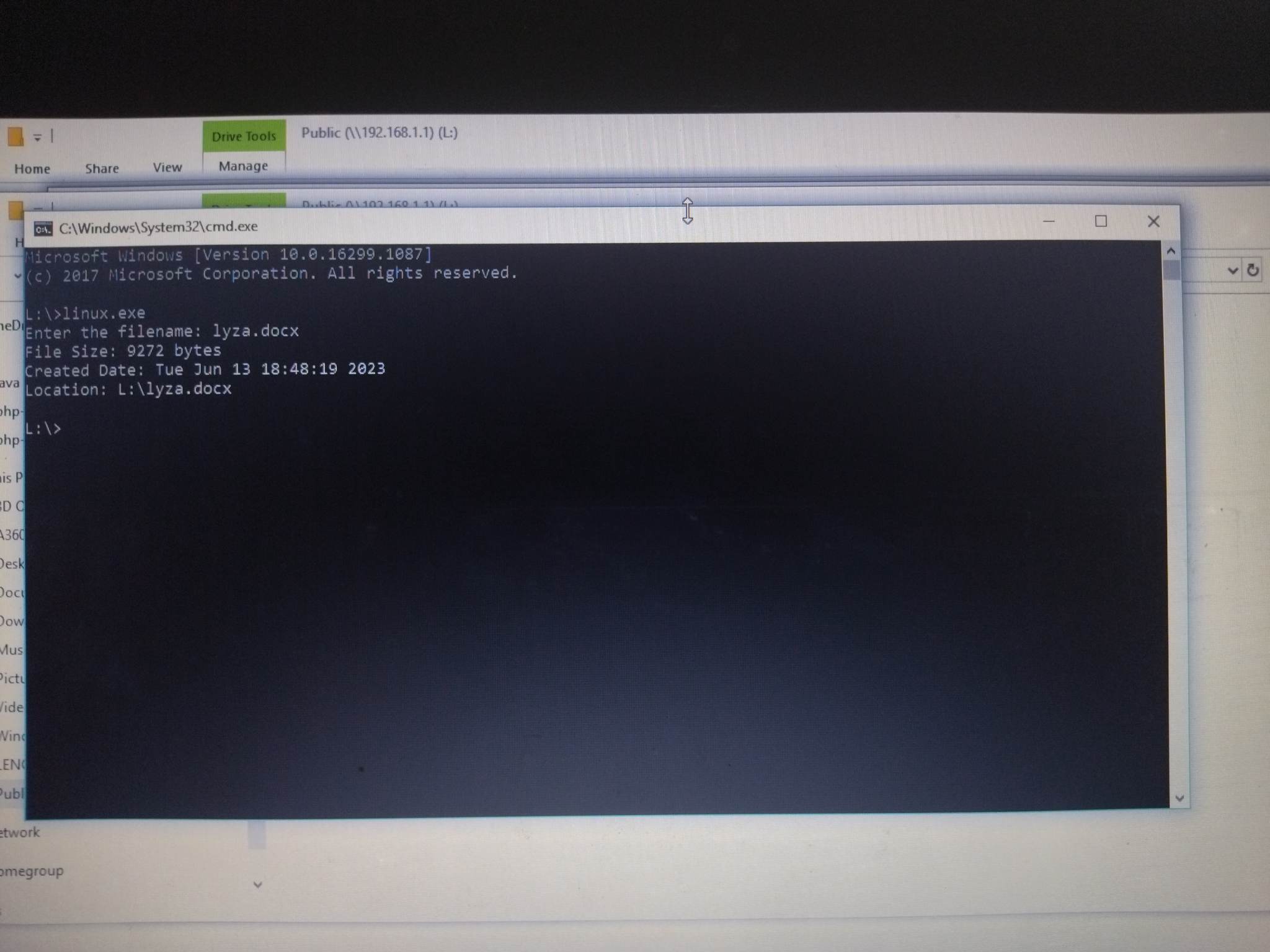
b. Compile the system call program separately for each operating system, taking into account any platform-specific dependencies.

c. Execute the system call program on both virtual machines, ensuring that the program operates as expected and provides the desired functionality.

d. Compare the results and behavior of the system call program on both operating systems to verify compatibility and consistency.







1. CPU Scheduling Technique Implementation:

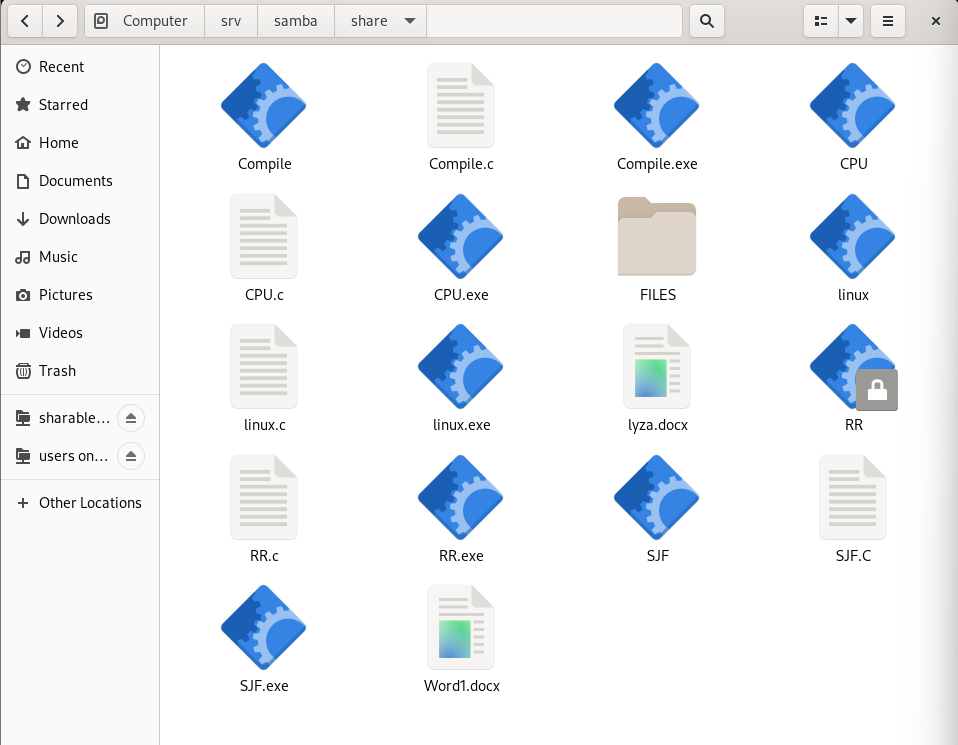
a. Choose a CPU scheduling technique to be implemented on the client machine.

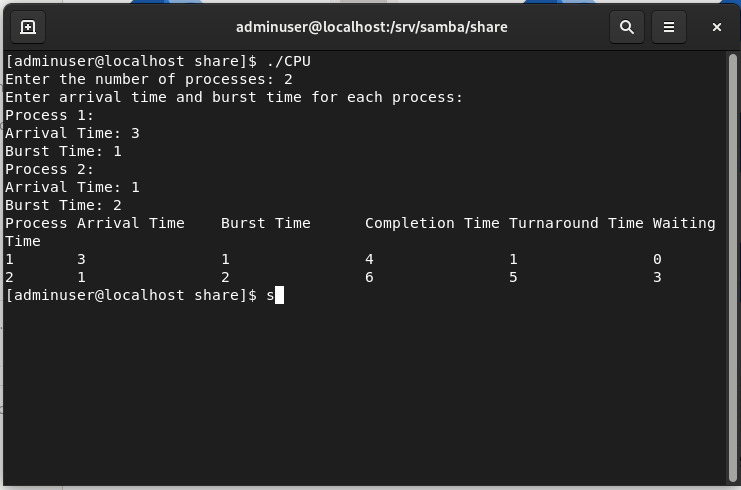
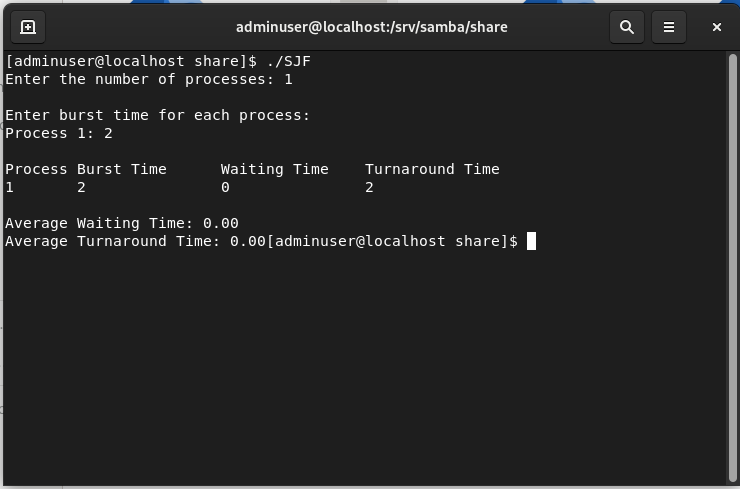
b. Develop a program or script that implements the chosen CPU scheduling technique, ensuring it is compatible with both Windows and Linux operating systems.

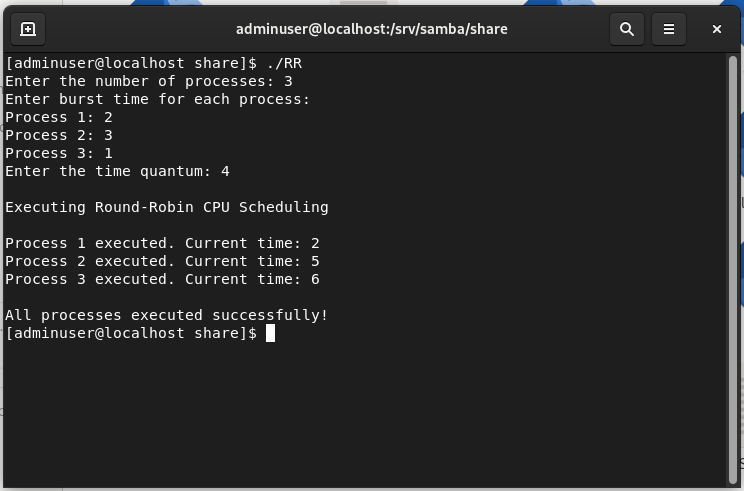
c. Test the CPU scheduling technique on the client machine, verifying its functionality and effectiveness in optimizing process scheduling.

d. Transmit the CPU scheduling technique program or script to the server machine, adjusting the file permissions to allow execution on the server side.

e. Execute the received CPU scheduling technique program or script on the server machine, observing its behavior and performance.







By following this methodology, users can effectively establish a cross-platform network communication system between a Windows and Linux virtual machine, while also executing system call programs and implementing CPU scheduling techniques that function seamlessly across both operating systems.

CONCLUSION

In conclusion, this documentation has addressed the challenges of establishing communication between two machines running different operating systems, specifically Linux and Windows, within a local area network (LAN) environment. The main objectives were to enable seamless sharing and execution of files between the machines and to implement a system call program and a CPU scheduling technique that functioned consistently on both platforms.

The setup allowed for the establishment of communication between the Linux and Windows machines by accessing a shared folder visible on both sides. This facilitated the seamless sharing of files between the machines, enabling efficient collaboration and data exchange. The chosen client-server architecture provided flexibility in configuring the communication setup to meet specific requirements.

A system call program was successfully executed on both machines. By developing the program to be compatible with both Linux and Windows, users were able to leverage system-level functionalities regardless of the underlying operating system. We create a c program in the system called that get the date and time, size, and location of files between Linux and Windows.

Furthermore, a CPU scheduling technique, determined by the client, was implemented and tested. The resulting file was securely sent to the server, with the necessary permission changes applied to ensure its successful execution on the server side. This allowed users to optimize their computing environments by customizing CPU scheduling techniques and effectively utilizing system resources.

REFLECTION

Throughout the process of documenting the establishment of communication between Linux and Windows machines, as well as the implementation of a system call program and CPU scheduling technique, I gained valuable insights and experienced several key learnings.

One of the significant achievements was successfully addressing the challenge of establishing communication between machines running different operating systems within a local area network. By configuring a shared folder accessible to both machines, I was able to enable seamless file sharing and collaboration. The flexibility offered by the client-server architecture allowed for customization based on specific requirements, providing an adaptable communication setup.

The implementation of a system call program was an interesting and enlightening experience. Developing the program to be compatible with both Linux and Windows required careful consideration of platform-specific dependencies and ensuring consistent behavior across operating systems. It was fulfilling to witness the program execute successfully on both machines, allowing users to leverage system-level functionalities regardless of the underlying operating system.

The utilization of a CPU scheduling technique further enhanced the optimization of computing environments. By allowing clients to create and transmit their preferred technique to the server. This experience has helped me understand the complexities of different operating systems better and the importance of adaptable solutions for smooth communication and collaboration of two operating systems.