Secured Access And Manipulation Of Files in Client-Server Architecture Through Inter Process Communication Using Message Queues

SHREYAS KRISHNASWAMY(1RV22CS191), VEERESH(1RV22CS229)
Department of Computer Science and Engineering, R V College of Engineering, Mysore Rd, RV Vidyaniketan, Post, Bengaluru, Karnataka 560059

SUBMITTED TO:
Prof. Jyothi Shetty

ABSTRACT:

The project is a file transfer server implemented in Linux using message queues for inter-process communication. It serves as a secure platform for clients to request file transfers between directories. The server employs message queues to authenticate clients, manage communication, and facilitate file transfers. This project highlights the effectiveness of IPC mechanisms for ensuring secure client-server communication and efficient file manipulation operations within a distributed system.

PROBLEM STATEMENT:

Enforcing authentication and authorization mechanisms, and implementing secure communication channels, organizations is necessary to protect sensitive information from unauthorized access and tampering of data.

SOLUTION:

Securing access and manipulation of files in a client-server architecture using IPC via message queues is imperative for maintaining confidentiality, integrity, and trust.

RELEVANCE:

- **File Management**: Operating systems are responsible for managing files and providing mechanisms for processes to access and manipulate them.
- **Process Communication**: IPC mechanisms, such as message queues, are fundamental components of operating systems, enabling inter-process communication.
- **Resource Protection**: Operating systems must protect system resources, including files, from unauthorized access and manipulation.
- **System Security**: Security is a critical aspect of operating system design and implementation.

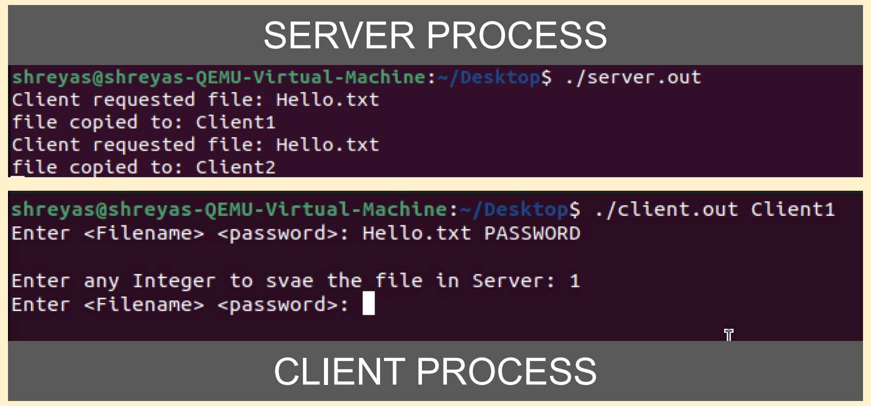
APPLICATIONS:

- **File System Security**: Operating systems manage file systems, including access control and permissions.
- **Networking and Distributed Systems**: Operating systems often support networked and distributed environments where multiple systems communicate over networks.
- **Kernel Security**: The kernel, the core component of an operating system, manages system resources, including file operations. Securing access and manipulation of files within the kernel is crucial for system security.
- **System Services and Daemons**: Operating systems often run system services and daemons responsible for various tasks, including file management. Employing secure IPC mechanisms ensures that communication between these components is secure, preventing unauthorized access to files or system resources.

SYSTEM CALLS:

- **msgget()**: This system call is used to create or access message queues.
- **msgrcv()**: This system call is used to receive messages from a message queue.
- **msgsnd()**: This system call is used to send messages to a message queue.
- **open()**: This system call is used to open files. It is used to open files for reading and writing.
- **read()**: This system call is used to read data from an open file descriptor.
- **write()**: This system call is used to write data to an open file descriptor.
- **unlink()**: This system call is used to remove a file. It is used to delete the temporary file created during the file copy operation.

IMPLEMENTATION:

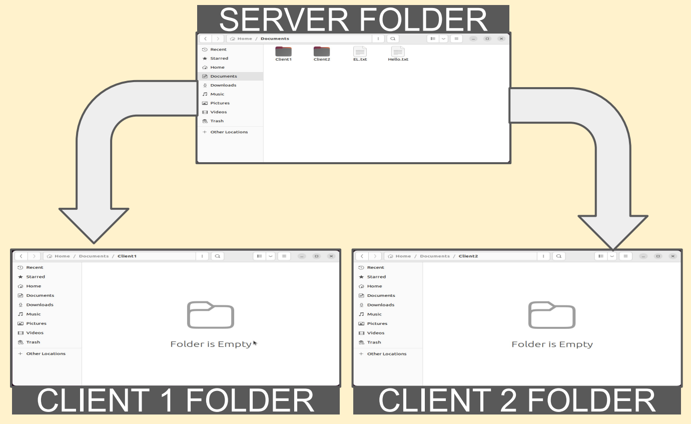

SERVER PROCESS

```
shreyas@shreyas-QEMU-Virtual-Machine:~/Desktop$ ./server.out
Client requested file: Hello.txt
file copied to: Client1
Client requested file: Hello.txt
file copied to: Client2
```

```
shreyas@shreyas-QEMU-Virtual-Machine:~/Desktop$ ./client.out Client1
Enter <Filename> <password>: Hello.txt PASSWORD

Enter any Integer to svae the file in Server: 1
Enter <Filename> <password>: 
```

CLIENT PROCESS



REQUIREMENTS
Identify the need for IPC and specify requirements such as data exchange frequency, types of data to be transferred, and system constraints.

IMPLEMENTATION:
Choose a suitable message queue implementation based on the operating system and programming language requirements, and determining the ways to authenticate file access.

MESSAGE STRUCTURE
Define the structure of messages to be exchanged between processes, including data fields, message types, and any necessary metadata.

COMMUNICATION
Develop code to establish communication between processes using the selected message queue implementation This involves sending files from the server process and receiving files in the receiver process and sending it back after editing.

TEST AND OPTIMIZE
Thoroughly test the project implementation to ensure correct functionality and performance under various conditions. Optimize the implementation for better performance by fine-tuning message queue settings and other parameters.