RV Educational Institutions ®

RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

TOPIC : Secured Access And Manipulation Of Files in Client-Server Architecture (Using Message Queues)

Submitted by

NAME	USN
VEERESH	1RV22CS229
SHREYAS KRISHNASWAMY	1RV22CS191

Under the Guidance of

DR. JYOTHI SHETTY

PROF. APPORVA UDAY KIRAN CHATE

DEPARTMENT OF COMPUTER SCIENCE

INTRODUCTION

The exchange and manipulation of files in client-server architectures represent core functionalities of countless applications across diverse domains. However, ensuring the security and integrity of these operations amidst the evolving landscape of cyber threats remains a paramount concern. Traditional approaches often face challenges related to authentication, authorization, and data protection, necessitating innovative solutions to fortify file handling processes.This paper introduces a novel approach to address these challenges by leveraging message queues within a client-server architecture. Message queues provide a robust communication mechanism that enhances security, scalability, and reliability in file transactions between clients and servers. By decoupling communication channels and implementing advanced security measures, our solution aims to establish a secure framework for accessing and manipulating files in distributed environments.The integration of message queues offers several key advantages. Firstly, it introduces a layer of abstraction that isolates clients from direct interactions with server resources, reducing the surface area for potential attacks. Secondly, message queues facilitate asynchronous communication, enabling seamless scalability and improved system responsiveness. Moreover, they enable the implementation of sophisticated security protocols, including authentication, authorization, and encryption, to safeguard file transactions from unauthorized access and malicious activities.In this paper, we delve into the design principles and architecture of our proposed system, elucidating how message queues enhance the security and efficiency of file handling processes. We explore the role of authentication mechanisms in verifying the identity of clients and servers, as well as authorization mechanisms in regulating access to files based on predefined policies.Through empirical evaluations and performance analyses, we aim to demonstrate the effectiveness of our approach in real-world scenarios. By comparing our solution to conventional file transfer methods, we seek to showcase its superior security, reliability, and performance metrics.

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.

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. Ensuring secure access to files is essential to prevent unauthorized users or processes from compromising system integrity and confidentiality.
- **Process Communication**: IPC mechanisms, such as message queues, are fundamental components of operating systems, enabling inter-process communication. Secure IPC ensures that sensitive data exchanged between processes is protected from unauthorized access and tampering.
- **Resource Protection**: Operating systems must protect system resources, including files, from unauthorized access and manipulation. By implementing robust security measures, operating systems can enforce access control policies, authenticate users and processes, and safeguard sensitive data.
- **System Security**: Security is a critical aspect of operating system design and implementation. Secure file access and IPC mechanisms help prevent security vulnerabilities, such as privilege escalation, data breaches, and denial-of-service attacks, which can compromise system security and stability.

METHODOLOGY

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.

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.

COMMUNICATION:

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

APPLICATIONS

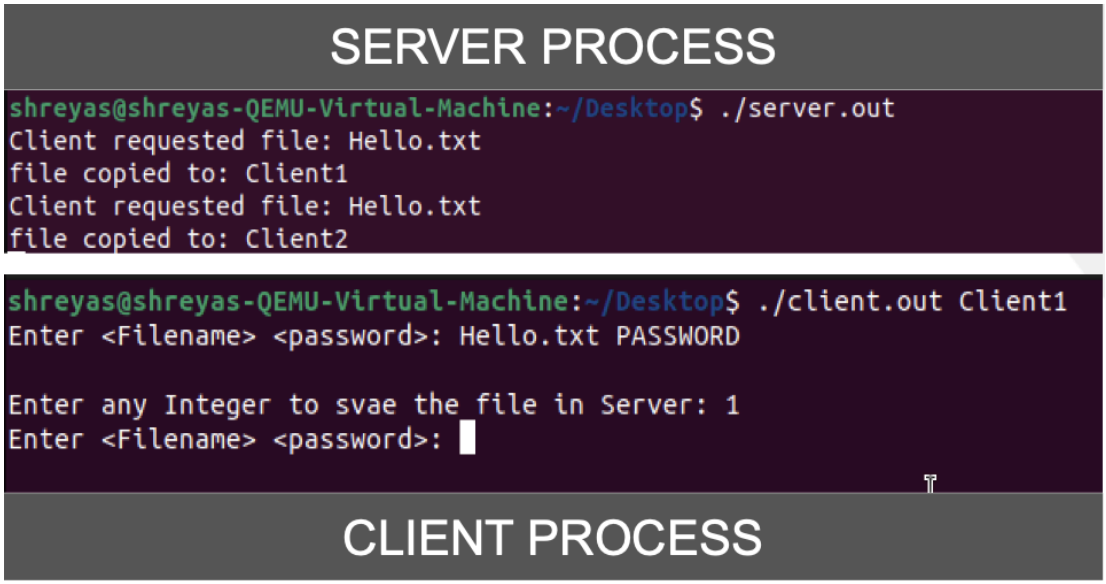
- **File System Security**: Operating systems manage file systems, including access control and permissions. Implementing secure IPC mechanisms like message queues can enhance the security of file system operations.
- **Networking and Distributed Systems**: Operating systems often support networked and distributed environments where multiple systems communicate over networks. Securely transferring files between systems in such environments requires robust security mechanisms.
- **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. Using secure IPC mechanisms like message queues within the kernel ensures that file-related operations are performed securely, mitigating the risk of security vulnerabilities or exploits.
- **System Services and Daemons**: Operating systems often run system services and daemons responsible for various tasks, including file management. These services may communicate with each other or with user applications to perform file-related operations. Employing secure IPC mechanisms such as message queues ensures that communication between these components is secure, preventing unauthorized access to files or system resources.

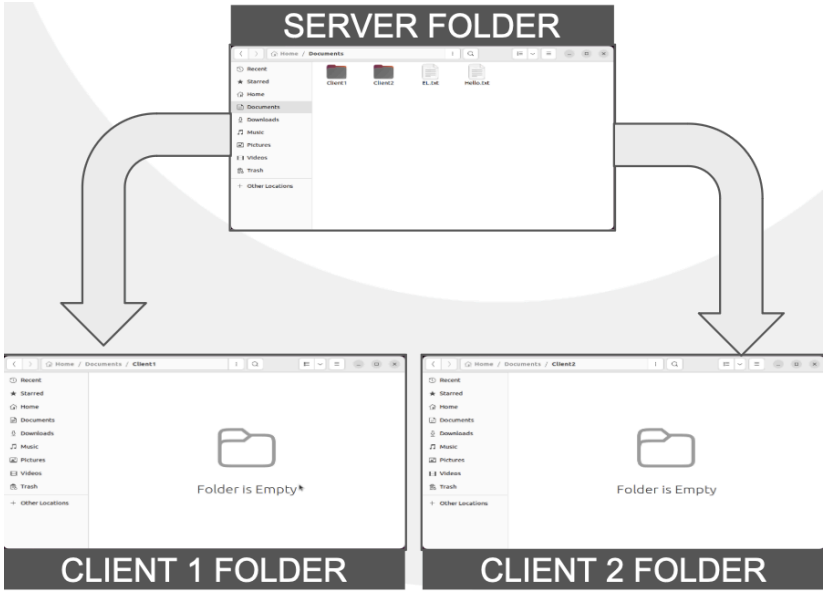
SYSTEMS CALLS USED

- **msgget()**: This system call is used to create or access message queues. It is called twice in the code to create two message queues: request_qid and response_qid.
- **msgrcv()**: This system call is used to receive messages from a message queue. It is used multiple times in the code to receive messages from the response_qid message queue.
- **msgsnd()**: This system call is used to send messages to a message queue. It is used to send messages to the response_qid message queue to communicate the success or failure status of the password verification process.
- **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. It is used to read file contents from the input file descriptor.
- **write()**: This system call is used to write data to an open file descriptor. It is used to write file contents to the output 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 CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define BUF_SIZE 8192
#define MSG_TYPE_REQUEST 1
#define MSG_TYPE_RESPONSE 2
#define MSG_QUEUE_KEY 2000

typedef struct {
    long mtype;
    char filename[100];
    char content[1024];
} message_t;

int main() {
        int input_fd, output_fd;
        ssize_t ret_in, ret_out;
        char buffer[BUF_SIZE];
        char a[150] = "/home/shreyas/Documents/" , c[150] = "/home/shreyas/Documents/", pass[50] = " ";
        int msgqid, request_qid, response_qid;
        message_t msg;



while (1) {
    char b[20];
    strcpy(a, "/home/shreyas/Documents/") ;strcpy(c,"/home/shreyas/Documents/"); strcpy(pass, " ");
    request_qid = msgget(MSG_QUEUE_KEY, IPC_CREAT | 0666);
    if (request_qid == -1) {
 perror("msgget (request)");
        exit(1);
    }
    response_qid = msgget(MSG_QUEUE_KEY + 1, IPC_CREAT | 0666);
    if (response_qid == -1) {
        perror("msgget (response)");
        exit(1);
    }
        msg.mtype = MSG_TYPE_RESPONSE;
        if (msgrcv(response_qid, &msg, sizeof(msg.filename), MSG_TYPE_RESPONSE, 0) == -1) {
                perror("msgrcv");
                exit(1);
        }
        strcpy(pass,msg.filename);
        while(strcmp("PASSWORD", pass)){
                strcpy(msg.filename, "Failure");
                if (msgsnd(response_qid, &msg, sizeof(msg.filename), 0) == -1) {
                    perror("msgsnd");
                    exit(1);
                }
                if (msgrcv(response_qid, &msg, sizeof(msg.filename), MSG_TYPE_RESPONSE, 0) == -1) {
                        perror("msgrcv");
                        exit(1);
                }
                strcpy(pass,msg.filename);
        }
        strcpy(msg.filename, "Success");
        if (msgsnd(response_qid, &msg, sizeof(msg.filename), 0) == -1) {
            perror("msgsnd");
            exit(1);
 }
    msg.mtype = MSG_TYPE_REQUEST;
    if (msgrcv(request_qid, &msg, sizeof(msg.filename), MSG_TYPE_REQUEST, 0) == -1) {
        perror("msgrcv");
        exit(1);
```

```
        }
        printf("Client requested file: %s\n", msg.filename);
        strcat(a,msg.filename);
        strcpy(b,msg.filename);
        if (msgrcv(request_qid, &msg, sizeof(msg.filename), MSG_TYPE_REQUEST, 0) == -1) {
           perror("msgrcv");
           exit(1);
        }
        printf("file copied to: %s\n", msg.filename);
        strcat(c,msg.filename);
        input_fd = open (a, O_RDONLY);
             if (input_fd == -1) {
                        perror ("open");
                        return 2;
             }
             strcat(c,"/");
             strcat(c,b);
             output_fd = open(c, O_WRONLY | O_CREAT, 0644);
             if(output_fd == -1){
                        perror("open");
                        return 3;

}
             while((ret_in = read (input_fd, &buffer, BUF_SIZE)) > 0){
                        ret_out = write (output_fd, &buffer, (ssize_t) ret_in);
                        if(ret_out != ret_in){
             perror("write");
                                   return 4;
                        }
             }
             close (input_fd);
             close (output_fd);
             if (msgrcv(request_qid, &msg, sizeof(msg.filename), MSG_TYPE_REQUEST, 0) == -1) {
           perror("msgrcv");
           exit(1);
        }
        input_fd = open (c, O_RDONLY);
             if (input_fd == -1) {
                        perror ("open");
                        return 2;
             }
             output_fd = open(a, O_WRONLY | O_CREAT, 0644);
             if(output_fd == -1){
                        perror("open");
                        return 3;
             }
             while((ret_in = read (input_fd, &buffer, BUF_SIZE)) > 0){
                        ret_out = write (output_fd, &buffer, (ssize_t) ret_in);
                        if(ret_out != ret_in){
                                   perror("write");
                                   return 4;
                        }
             }
             close (input_fd);
             close (output_fd);
             int output_fd = unlink(c);
if(output_fd == -1){
                        perror("unlink error");
                        return 3;
             }
   }
   return 0;
}
```

CLIENT CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>

typedef struct {
    long mtype;
    char filename[100];
    char content[1024];
} message_t;

#define MSG_TYPE_REQUEST 1
#define MSG_TYPE_RESPONSE 2
#define MSG_QUEUE_KEY 2000
#define MSG_QUEUE_KEY_RESPONSE (MSG_QUEUE_KEY + 1)

int main(int argc, char* argv[]) {
char a[20],b[5] = "quit", pass[30] , j[30];
int m;
 int msgqid, response_qid;
                message_t msg;
              msgqid = msgget(MSG_QUEUE_KEY, 0);
                if (msgqid == -1) {
                        perror("msgget (request)");
                        exit(1);
                }
                response_qid = msgget(MSG_QUEUE_KEY_RESPONSE, 0);
                if (response_qid == -1) {
                        perror("msgget (response)");

exit(1);
                }
                while(1){
                        printf("Enter <Filename> <password>: ");
                        scanf("%s %s",a,pass);
                        int i =0;
                        if(!strcmp(a,b)){
                                return 0;
                        }
                        msg.mtype = MSG_TYPE_RESPONSE;
                        strcpy(msg.filename,pass);
                        if (msgsnd(response_qid, &msg, sizeof(msg.filename), 0) == -1) {
                           perror("msgsnd");
                           }
                        if (msgrcv(response_qid, &msg, sizeof(msg.filename), MSG_TYPE_RESPONSE, 0) == -1) {
                                perror("msgrcv");
                                exit(1);
                        }
                        strcpy(j,msg.filename);
                        while(strcmp(j,"Success")){
                                printf("Wrong Password!\nEnter right Password : ");
                                scanf("%s",pass);
                                strcpy(msg.filename,pass);
                        if (msgsnd(response_qid, &msg, sizeof(msg.filename), 0) == -1) {
                           perror("msgsnd");
                           }
                        if (msgrcv(response_qid, &msg, sizeof(msg.filename), MSG_TYPE_RESPONSE, 0) == -1) {
                                perror("msgrcv");
                                exit(1);
                        }

strcpy(j,msg.filename);
                        }
                msg.mtype = MSG_TYPE_REQUEST;
                strcpy(msg.filename, a);
                if (msgsnd(msgqid, &msg, sizeof(msg.filename), 0) == -1) {
```

```
                perror("msgsnd");
                exit(1);
        }
        strcpy(msg.filename, argv[1]);
        if (msgsnd(msgqid, &msg, sizeof(msg.filename), 0) == -1) {
                perror("msgsnd");
                exit(1);
        }


 printf("\nEnter any Integer to svae the file in Server: ");
        scanf("%d", &m);
        if (msgsnd(msgqid, &msg, sizeof(msg.filename), 0) == -1) {
                perror("msgsnd");
                exit(1);
        }
    }
   return 0;
}
```

CONCLUSION

Implementing secured access and manipulation of files in a client-server architecture using message queues offers several advantages. By leveraging message queues, such as RabbitMQ or Kafka, the system can ensure asynchronous communication, scalability, and fault tolerance. Securing access to files through authentication, authorization, and encryption enhances data integrity and confidentiality. Moreover, utilizing message queues facilitates efficient file manipulation, enabling reliable file transfers, processing, and management across distributed systems. Overall, this approach provides a robust foundation for building secure and efficient file management systems in client-server architectures.Furthermore, the use of message queues enables decoupling between components, allowing for better maintainability and flexibility in the system's architecture. This separation of concerns simplifies updates and upgrades to individual components without disrupting the overall system functionality. Additionally, incorporating logging and monitoring mechanisms into the architecture enhances visibility into file operations, aiding in troubleshooting and ensuring compliance with regulatory requirements. Overall, by combining secure access controls with message queue-based file manipulation, organizations can establish a resilient and adaptable foundation for managing files in client-server architectures, meeting both security and operational needs effectively.