National University of Computer & Emerging Sciences
Karachi Campus

Real Life Mail Server in C

Semester Project Report - Operating Systems (Spring 2025)
Section: 4C

Group Members:

23K-0628 M.Taha Nauman
23K-0044 Omaima Afaq
23K-0876 Aiman Nadeem Khan
23K-0678 Afaf Shahid

Table of Contents

1. Introduction
2. Project Objectives
3. Synchronization Objectives
4. System Design and Architecture
5. Development Tools and Environment
6. Test Plan and Test Cases
7. Project Artifacts
8. Deviations from Initial Plan
9. Known Improvements and Future Work
10. Conclusion

Introduction

This project aims to develop a fully functional Mail Server in C that integrates important Operating System (OS) concepts such as multi-processing, multi-threading, inter-process communication (IPC), process synchronization, and process scheduling. The server simulates email handling, user authentication, and concurrent client interactions, creating a practical and scalable communication system. Certain concepts like deadlock handling and some aspects of virtual memory management (mmap) were initially explored but later skipped to maintain project stability and focus on core functionalities. Demand paging concepts were considered in memory management techniques.

Project Objectives

- **Multi-Client Support:** Handle concurrent client connections using multi-processing and multi-threading.
- **Process Communication:** Use IPC mechanisms (message queues) for communication between processes.
- **Resource Synchronization:** Apply semaphores, mutexes, and reader-writer locks to ensure safe access to shared resources.
- **Deadlock Management:** *Skipped* — explained in Section 8.1 and 8.2.
- **Virtual Memory Management:** *Skipped* — explained in Section 8.1 and 8.2.
- **Process Scheduling:** Simulate mail processing fairness using sleep-based Round Robin scheduling.

Synchronization Objectives

Concept	Application in the Project
Inter-Process Communication (IPC)	Message queues used for inter-process communication.
Multi-Processing & Multi-Threading	fork() used for creating new processes; threads handle multiple operations per client.
Process Synchronization	Semaphores and mutexes prevent race conditions and coordinate shared resource access.
Reader-Writer Problem	Reader-writer locks optimize concurrent reading and exclusive writing to mailboxes.
Process Scheduling	Simulated with Round Robin scheduling using sleep() calls to introduce fairness between processes.

System Design and Architecture

4.1 Client-Server Model:

- Clients establish TCP socket connections.
- The server spawns processes or threads for each new client.

4.2 Process Handling:

- Parent server process waits for connections.
- New child processes (fork()) handle individual client sessions.

4.3 Data Flow:

- Client requests are parsed.
- IPC mechanisms (message queues) are used to pass information between processes.

4.4 Synchronization Mechanisms:

- Mutexes protect critical sections like inbox and sent folders.
- Semaphores regulate access to mail storage.
- Reader-writer locks are used for efficient email retrieval and updates.

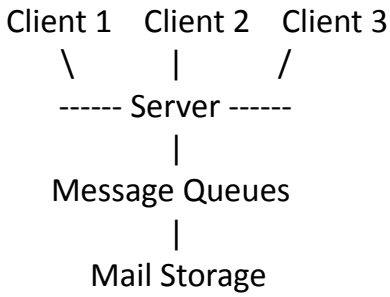
4.5 Scheduling:

- Simple Round Robin Scheduling simulated using sleep() to allow each process a fair turn.

4.6 Memory Management:

- **Note:** Virtual memory using mmap was explored but not implemented due to technical challenges and project stability concerns.

System Architecture Diagram:



Development Tools and Environment

Item	Tool/Version
Programming Language	C (GCC Compiler)
Libraries and Frameworks	POSIX Threads (pthreads)
Operating System	Linux/Unix-based OS

Test Plan and Test Cases

6.1 Unit Testing:

- Individual modules like login authentication, email sending, and inbox retrieval tested.

6.2 Integration Testing:

- Verified interaction between server and multiple clients.

6.3 Concurrency Testing:

- Multiple clients simultaneously logged in and sent emails without conflict.

6.4 Performance Testing:

- Server performance observed under simultaneous client loads.

6.5 Security Testing:

- Basic validation for user authentication, error handling for invalid usernames/passwords.

Project Artifacts

Artifact	Planned	Description/Notes
Source Code	Yes	Server and client-side logic, IPC, synchronization, and scheduling mechanisms implemented.
Requirements Document	Yes	Detailed project requirements and feature descriptions.
Design and Architecture Document	Yes	High-level diagrams and flow explanations.
Test Plan and Test Cases	Yes	Strategy for testing core functionalities and concurrency handling.

Deviations from Initial Plan

8.1 Deadlock Management: Deadlock handling was initially considered but later skipped. After careful analysis, it was concluded that the server's design (simple client-server model with short-lived critical sections) minimized the risk of deadlocks. Implementing deadlock detection would have unnecessarily complicated the project without tangible benefits.

8.2 Virtual Memory (mmap): The use of mmap was explored to optimize file access and memory mapping. However, during testing, it caused unexpected behavior and instability in file handling. To prioritize project functionality and stability, the traditional file I/O methods were retained.

Code Snippets

```
mtahanauman90@penguin:~$ make run-server
./socket_server
Server listening on port 2727...
New connection from 127.0.0.1:38960
Client Aiman@gmail.com connected
Aiman@gmail.com requested disconnection
```

```
mtahanauman90@penguin:~$ make run-writer
./email_writer
Writer started. Waiting for messages (PID: 7245)...
Processing email from Aiman@gmail.com to Taha@gmail.com...
Successfully stored email
```

```
mtahanauman90@penguin:~$ make run-client
./socket_client
1) Login
2) Create Account
Choose Option: 1
Enter username: Aiman@gmail.com
Enter password: Nadeem
Login successful!
Connected to Mail Server
Server: MAIL SERVER: Welcome Aiman@gmail.com!

1. Send Email
2. Fetch Inbox
3. Exit
> 2

--- Your Inbox (Page 1 of 4, showing 4 emails) ---

Email 1:
From: Taha@gmail.com
Subject: OS Project Works
Body: Hey I successfully added the email send and fetch inbox
--------------------------
Email 2:
From: Taha@gmail.com
Subject: Heyoooo
Body: Hi
--------------------------
Email 3:
From: Taha@gmail.com
Subject: Hi
Body: Testing Fetch Inbox
--------------------------
Email 4:
From: Taha@gmail.com
Subject: Hi
Body: Howre u
--------------------------

Options: [n]ext page, [p]revious page, [q]uit viewing: q
```

```
1. Send Email
2. Fetch Inbox
3. Exit
> 1
From: Aiman@gmail.com
To: Taha@gmail.com
Subject: OS Project
Body: Its Completed
Server: EMAIL_QUEUED

1. Send Email
2. Fetch Inbox
3. Exit
> 3
mtahanauman90@penguin:~$
```

Known Improvements and Future Work

While the core functionality was achieved successfully, a few minor improvements are identified for future development:

- Post-Exit Redirection: After user logout or exit, automatically return to the login page instead of terminating the session.
- Email Ordering: Display emails with the latest ones first, instead of the current oldest-first order.
- Better Error Handling: Show user-friendly messages for invalid usernames/passwords during login and email sending.
- UI/UX Enhancements: Improve command-line prompts and feedback for better user experience.

Conclusion

The Mail Server project successfully demonstrates the application of fundamental OS concepts like multi-processing, multi-threading, IPC (message queues), synchronization, and scheduling in building a concurrent and scalable server. While some advanced features like deadlock management and virtual memory optimization (mmap) were explored but ultimately skipped for practical reasons, the core objectives were fully achieved, resulting in a functional, stable, and educational project.