Operating Systems Project

Muhammad Abdullah Aamir
Roll No: 22P-9235

November 26, 2024

FAST University Peshawar

Department of Computer Science
Course: **Operating Systems**
Lecturer: **Engr. M. Usman Malik**
Due Date: **26 November 2024**

Objective: To implement CPU scheduling algorithms and socket programming for both local and distributed systems.

Contents

1 Introduction

This project focuses on two essential areas of Operating Systems:

- **CPU Scheduling Algorithms:** Implementation and analysis of scheduling techniques like First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, Round Robin (RR), and Priority with Round Robin.
- **Socket Programming:** Establishing communication using sockets in local and distributed systems.

The goal is to deepen the understanding of process scheduling and inter-process communication through practical implementation.

2 CPU Scheduling Implementation

2.1 Task Description

The scheduling algorithms are implemented to process tasks defined in a file (`schedule.txt`) with the following details:

- Task Name
- Priority
- CPU Burst Time

Metrics calculated include:

- Completion Time
- Waiting Time
- Turnaround Time
- Average Waiting and Turnaround Times

2.2 Code Implementation

The program reads task details from a file and processes them using multiple CPU scheduling algorithms. Below is a snippet of the main code structure:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Task structure definition
typedef struct {
    char name[10];
    int priority;
    int cpuBurst;
    int completionTime;
```

```c
    int waitingTime;
    int turnaroundTime;
} Task;

// Function prototypes
void fcfs(Task tasks[], int n);
void sjf(Task tasks[], int n);
void priorityScheduling(Task tasks[], int n);
void roundRobin(Task tasks[], int n, int timeQuantum);
void priorityWithRoundRobin(Task tasks[], int n, int timeQuantum);

int main() {
    Task tasks[100];
    int n = 0, choice, timeQuantum = 4;

    // Reading tasks from file
    FILE *file = fopen("schedule.txt", "r");
    if (!file) {
        perror("Failed to open schedule.txt");
        return 1;
    }

    while (fscanf(file, "%[^,], %d, %d\n", tasks[n].name, &tasks[n].priority, &tasks[
        n++;
    }
    fclose(file);

    if (n == 0) {
        printf("No tasks found in schedule.txt.\n");
        return 1;
    }

    do {
        printf("\nCPU Scheduling Algorithms:\n");
        printf("1. FCFS\n2. SJF\n3. Priority Scheduling\n4. Round Robin\n5. Priority
        printf("Select an option: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                fcfs(tasks, n);
                break;
            case 2:
                sjf(tasks, n);
                break;
            case 3:
                priorityScheduling(tasks, n);
```

```c
                break;
            case 4:
                printf("Enter time quantum: ");
                scanf("%d", &timeQuantum);
                roundRobin(tasks, n, timeQuantum);
                break;
            case 5:
                printf("Enter time quantum: ");
                scanf("%d", &timeQuantum);
                priorityWithRoundRobin(tasks, n, timeQuantum);
                break;
            case 6:
                return 0;
            default:
                printf("Invalid option! Try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

Refer to the attached code for the full implementation of the scheduling functions.

2.3 Results and Secreenshots

The following are the outputs generated by the program:

```
Enter your choice: 1

Execution Order (Gantt Chart): T1 -> T2 -> T3 -> T4

        --- First-Come-First-Served (FCFS) Results ---
+------+-----------+------------+---------------+---------------+
| Task | CPU Burst | Completion | Waiting Time  | Turnaround Time |
+------+-----------+------------+---------------+---------------+
| T1   | 8         | 8          | 0             | 8               |
| T2   | 4         | 12         | 8             | 12              |
| T3   | 9         | 21         | 12            | 21              |
| T4   | 5         | 26         | 21            | 26              |
+------+-----------+------------+---------------+---------------+
Average Waiting Time: 10.25 ms
Average Turnaround Time: 16.75 ms
```

Figure 1: Sample Output of FCFS Scheduling

```
Enter your choice: 2

Execution Order (Gantt Chart): T2 -> T4 -> T1 -> T3

        --- Shortest Job First (SJF) Results ---
+------+-----------+------------+----------------+----------------+
| Task | CPU Burst | Completion | Waiting Time   | Turnaround Time |
+------+-----------+------------+----------------+----------------+
| T1   | 8         | 17         | 9              | 17              |
| T2   | 4         | 4          | 0              | 4               |
| T3   | 9         | 26         | 17             | 26              |
| T4   | 5         | 9          | 4              | 9               |
+------+-----------+------------+----------------+----------------+
Average Waiting Time: 7.50 ms
Average Turnaround Time: 14.00 ms
```

Figure 2: Sample Output of SJF Scheduling

```
Enter your choice: 3

Execution Order (Gantt Chart): T2 -> T3 -> T1 -> T4

        --- Priority Scheduling Results ---
+------+-----------+------------+----------------+----------------+
| Task | CPU Burst | Completion | Waiting Time   | Turnaround Time |
+------+-----------+------------+----------------+----------------+
| T1   | 8         | 21         | 13             | 21              |
| T2   | 4         | 4          | 0              | 4               |
| T3   | 9         | 13         | 4              | 13              |
| T4   | 5         | 26         | 21             | 26              |
+------+-----------+------------+----------------+----------------+
Average Waiting Time: 9.50 ms
Average Turnaround Time: 16.00 ms
```

Figure 3: Sample Output of Priority Scheduling

```
Enter your choice: 4
Enter time quantum: 2

Execution Order (Gantt Chart): T1 -> T2 -> T3 -> T4 -> T1 -> T2 -> T3 -> T4 -> T
1 -> T3 -> T4 -> T1 -> T3 -> T3

        --- Round Robin (RR) Results ---
+------+-----------+------------+----------------+----------------+
| Task | CPU Burst | Completion | Waiting Time   | Turnaround Time |
+------+-----------+------------+----------------+----------------+
| T1   | 8         | 23         | 15             | 23              |
| T2   | 4         | 12         | 8              | 12              |
| T3   | 9         | 26         | 17             | 26              |
| T4   | 5         | 21         | 16             | 21              |
+------+-----------+------------+----------------+----------------+
Average Waiting Time: 14.00 ms
Average Turnaround Time: 20.50 ms
```

Figure 4: Sample Output of Round Robin Scheduling

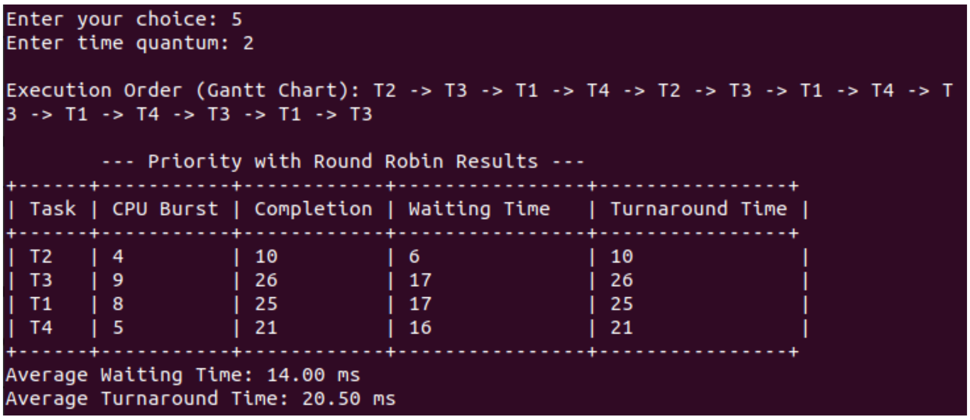

Figure 5: Sample Output of Priority Round Robin Scheduling

3 Socket Programming Implementation

3.1 Implementation

The implementation involves creating a client-server model using sockets. Below is the server code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 9090

void error(const char *msg) {
    perror(msg);
    exit(1);
}

int main() {
    int server_fd, new_socket, n;
    struct sockaddr_in address, cli_addr;
    socklen_t addrlen = sizeof(address);
    char buffer[1024] = {0};

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        error("ERROR: Unable to create socket.");
    }

    bzero((char *)&address, sizeof(address));
```

```c
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind the socket
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        error("ERROR: Binding failed.");
    }

    printf("Server is running on port %d...\n", PORT);

    // Listen
    if (listen(server_fd, 3) < 0) {
        error("ERROR: Listening failed.");
    }
    printf("Waiting for a connection...\n");

    // Accept a client connection
    new_socket = accept(server_fd, (struct sockaddr *)&cli_addr, &addrlen);
    if (new_socket < 0) {
        error("ERROR: Accepting connection failed.");
    }
    printf("Client connected.\n");

    // Communicate with the client
    while (1) {
        bzero(buffer, sizeof(buffer));
        n = read(new_socket, buffer, sizeof(buffer) - 1);
        if (n < 0) {
            error("ERROR: Reading from socket failed.");
        }

        printf("Client: %s\n", buffer);

        printf("You: ");
        bzero(buffer, sizeof(buffer));
        fgets(buffer, sizeof(buffer), stdin);

        // Remove the trailing newline character from `fgets`
        buffer[strcspn(buffer, "\n")] = '\0';

        n = write(new_socket, buffer, strlen(buffer));
        if (n < 0) {
            error("ERROR: Writing to socket failed.");
        }

        if (strncmp("Bye", buffer, 3) == 0) {
```

```
            printf("Exiting...\n");
            break;
        }
    }

    // Close the sockets
    close(new_socket);
    close(server_fd);

    return 0;
}
```

Below is the client code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>

void error(const char *msg) {
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[]) {
    int client_fd, n;
    struct sockaddr_in server_addr;
    struct hostent *server;

    char buffer[1024];

    if (argc < 3) {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
    }

    // Create socket
    client_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (client_fd < 0) {
        error("ERROR opening socket");
    }


    server = gethostbyname(argv[1]);
```

```
    if (server == NULL) {
        fprintf(stderr, "ERROR, no such host\n");
        exit(1);
    }


    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;

    bcopy((char *)server->h_addr, (char *)&server_addr.sin_addr.s_addr, server->h_len


    int port = atoi(argv[2]);
    if (port <= 0) {
        fprintf(stderr, "ERROR, invalid port number\n");
        exit(1);
    }
    server_addr.sin_port = htons(port);

    // Connect to the server
    if (connect(client_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
        error("ERROR connecting");
    }

    printf("Connected to the server...\n");

    // Communicate with server
    while (1) {
        bzero(buffer, sizeof(buffer));
        printf("You: ");
        fgets(buffer, sizeof(buffer), stdin);

        n = send(client_fd, buffer, strlen(buffer), 0);
        if (n < 0) {
            error("ERROR writing to socket");
        }

        if (strncmp(buffer, "Bye", 3) == 0) {
            printf("Exiting Client...\n");
            break;
        }

        bzero(buffer, sizeof(buffer));
        n = read(client_fd, buffer, sizeof(buffer) - 1);
        if (n < 0) {
            error("ERROR reading from socket");
        }
```

```c
        printf("Server: %s\n", buffer);

    }

    // Close the socket
    close(client_fd);

    return 0;
}
```

3.2 Sample Output

Screenshots:

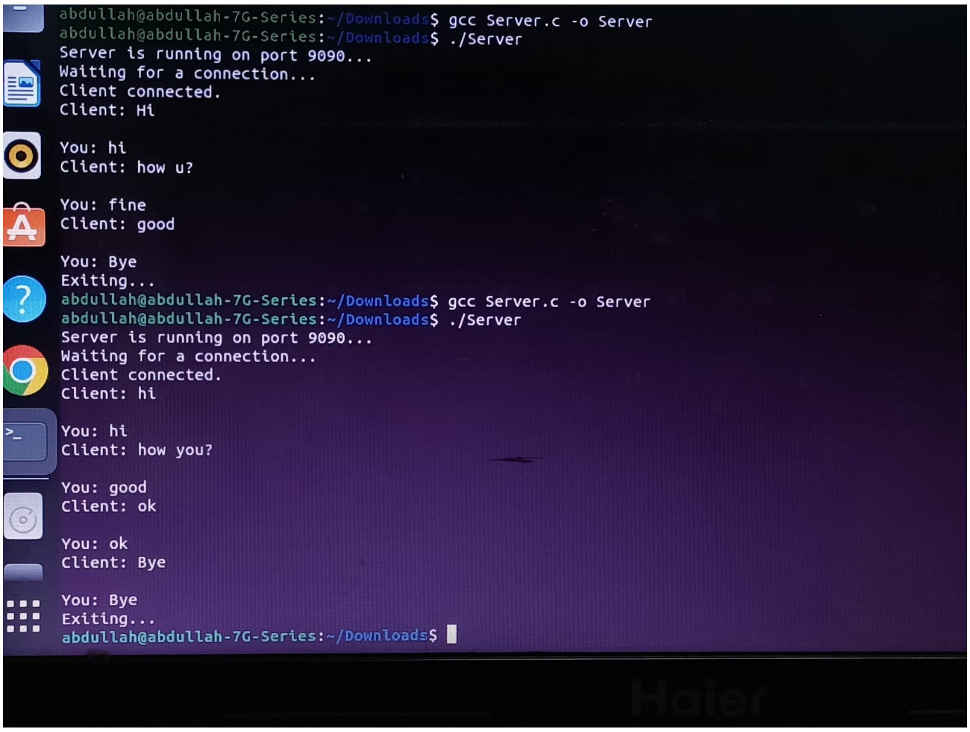

Figure 6: Sample Output of Server

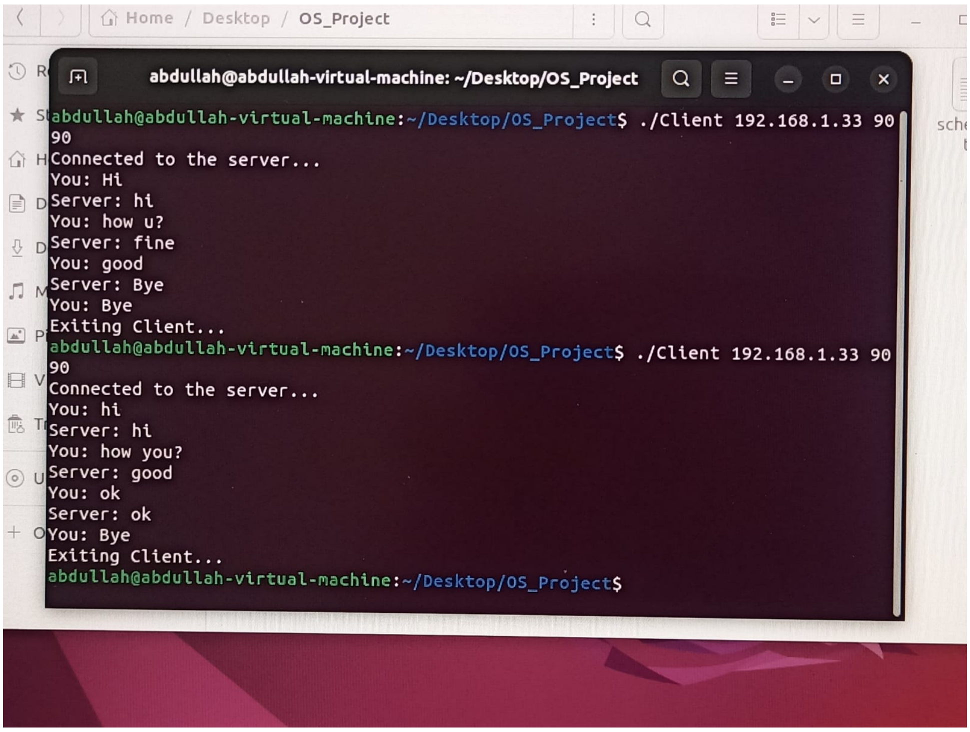

Figure 7: Sample Output of Client

4 Conclusion

This project provided hands-on experience with critical Operating Systems concepts, including scheduling algorithms and socket programming. The implementations demonstrated efficient process scheduling and successful inter-process communication in both local and distributed environments.

Bonus Task: Error Handling

To enhance robustness, error handling is implemented:

```
void error(const char *msg) {
    perror(msg);
    exit(1);
}
```

Figure 8: Error message function

Error Handling

- Handled socket creation, binding, connection, and data transmission errors with detailed messages.
- Improved user experience by gracefully managing unexpected issues.

```
if (server_fd < 0) {
    error("ERROR: Unable to create socket.");
}
```

Figure 9: Error message example 1

```
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    error("ERROR: Binding failed.");
}
```

Figure 10: Error message example 2