

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	2
2	CPU Scheduling Implementation	2
2.1	Task Description	2
2.2	Code Implementation	2
2.3	Results and Screenshots	4
3	Socket Programming Implementation	6
3.1	Implementation	6
3.2	Sample Output	10
4	Conclusion	11

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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // Task structure definition
6  typedef struct {
7      char name[10];
8      int priority;
9      int cpuBurst;
10     int completionTime;
```

```

11     int waitingTime;
12     int turnaroundTime;
13 } Task;
14
15 // Function prototypes
16 void fcfs(Task tasks[], int n);
17 void sjf(Task tasks[], int n);
18 void priorityScheduling(Task tasks[], int n);
19 void roundRobin(Task tasks[], int n, int timeQuantum);
20 void priorityWithRoundRobin(Task tasks[], int n, int timeQuantum);
21
22 int main() {
23     Task tasks[100];
24     int n = 0, choice, timeQuantum = 4;
25
26     // Reading tasks from file
27     FILE *file = fopen("schedule.txt", "r");
28     if (!file) {
29         perror("Failed to open schedule.txt");
30         return 1;
31     }
32
33     while (fscanf(file, "%[^,], %d, %d\n", tasks[n].name, &tasks[n].priority, &tasks[n].timeQuantum) == 3)
34         n++;
35     }
36     fclose(file);
37
38     if (n == 0) {
39         printf("No tasks found in schedule.txt.\n");
40         return 1;
41     }
42
43     do {
44         printf("\nCPU Scheduling Algorithms:\n");
45         printf("1. FCFS\n2. SJF\n3. Priority Scheduling\n4. Round Robin\n5. Priority With Round Robin\n");
46         printf("Select an option: ");
47         scanf("%d", &choice);
48
49         switch (choice) {
50             case 1:
51                 fcfs(tasks, n);
52                 break;
53             case 2:
54                 sjf(tasks, n);
55                 break;
56             case 3:
57                 priorityScheduling(tasks, n);

```

```

58         break;
59     case 4:
60         printf("Enter time quantum: ");
61         scanf("%d", &timeQuantum);
62         roundRobin(tasks, n, timeQuantum);
63         break;
64     case 5:
65         printf("Enter time quantum: ");
66         scanf("%d", &timeQuantum);
67         priorityWithRoundRobin(tasks, n, timeQuantum);
68         break;
69     case 6:
70         return 0;
71     default:
72         printf("Invalid option! Try again.\n");
73     }
74     while (choice != 6);
75
76     return 0;
77 }

```

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

2.3 Results and Screenshots

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

```

Enter your choice: 5
Enter time quantum: 2

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

--- Priority with Round Robin Results ---
+-----+-----+-----+-----+-----+
| Task | CPU Burst | Completion | Waiting Time | Turnaround Time |
+-----+-----+-----+-----+-----+
| T2 | 4 | 10 | 6 | 10 |
| T3 | 9 | 26 | 17 | 26 |
| T1 | 8 | 25 | 17 | 25 |
| T4 | 5 | 21 | 16 | 21 |
+-----+-----+-----+-----+-----+
Average Waiting Time: 14.00 ms
Average Turnaround Time: 20.50 ms

```

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:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/socket.h>
5  #include <arpa/inet.h>
6  #include <unistd.h>
7
8  #define PORT 9090
9
10 void error(const char *msg) {
11     perror(msg);
12     exit(1);
13 }
14
15 int main() {
16     int server_fd, new_socket, n;
17     struct sockaddr_in address, cli_addr;
18     socklen_t addrlen = sizeof(address);
19     char buffer[1024] = {0};
20
21     // Create socket
22     server_fd = socket(AF_INET, SOCK_STREAM, 0);
23     if (server_fd < 0) {
24         error("ERROR: Unable to create socket.");
25     }
26
27     bzero((char *)&address, sizeof(address));

```

```

28 address.sin_family = AF_INET;
29 address.sin_addr.s_addr = INADDR_ANY;
30 address.sin_port = htons(PORT);
31
32 // Bind the socket
33 if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
34     error("ERROR: Binding failed.");
35 }
36
37 printf("Server is running on port %d...\n", PORT);
38
39 // Listen
40 if (listen(server_fd, 3) < 0) {
41     error("ERROR: Listening failed.");
42 }
43 printf("Waiting for a connection...\n");
44
45 // Accept a client connection
46 new_socket = accept(server_fd, (struct sockaddr *)&cli_addr, &addrlen);
47 if (new_socket < 0) {
48     error("ERROR: Accepting connection failed.");
49 }
50 printf("Client connected.\n");
51
52 // Communicate with the client
53 while (1) {
54     bzero(buffer, sizeof(buffer));
55     n = read(new_socket, buffer, sizeof(buffer) - 1);
56     if (n < 0) {
57         error("ERROR: Reading from socket failed.");
58     }
59
60     printf("Client: %s\n", buffer);
61
62     printf("You: ");
63     bzero(buffer, sizeof(buffer));
64     fgets(buffer, sizeof(buffer), stdin);
65
66     // Remove the trailing newline character from `fgets`
67     buffer[strcspn(buffer, "\n")] = '\0';
68
69     n = write(new_socket, buffer, strlen(buffer));
70     if (n < 0) {
71         error("ERROR: Writing to socket failed.");
72     }
73
74     if (strncmp("Bye", buffer, 3) == 0) {

```

```

75         printf("Exiting...\n");
76         break;
77     }
78 }
79
80 // Close the sockets
81 close(new_socket);
82 close(server_fd);
83
84 return 0;
85 }

```

Below is the client code:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/socket.h>
5  #include <netdb.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8
9  void error(const char *msg) {
10     perror(msg);
11     exit(1);
12 }
13
14 int main(int argc, char *argv[]) {
15     int client_fd, n;
16     struct sockaddr_in server_addr;
17     struct hostent *server;
18
19     char buffer[1024];
20
21     if (argc < 3) {
22         fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
23         exit(1);
24     }
25
26     // Create socket
27     client_fd = socket(AF_INET, SOCK_STREAM, 0);
28     if (client_fd < 0) {
29         error("ERROR opening socket");
30     }
31
32
33     server = gethostbyname(argv[1]);

```



```

34     if (server == NULL) {
35         fprintf(stderr, "ERROR, no such host\n");
36         exit(1);
37     }
38
39
40     bzero((char *)&server_addr, sizeof(server_addr));
41     server_addr.sin_family = AF_INET;
42
43     bcopy((char *)server->h_addr, (char *)&server_addr.sin_addr.s_addr, server->h_len);
44
45
46     int port = atoi(argv[2]);
47     if (port <= 0) {
48         fprintf(stderr, "ERROR, invalid port number\n");
49         exit(1);
50     }
51     server_addr.sin_port = htons(port);
52
53     // Connect to the server
54     if (connect(client_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
55         error("ERROR connecting");
56 }
57
58 printf("Connected to the server...\n");
59
60 // Communicate with server
61 while (1) {
62     bzero(buffer, sizeof(buffer));
63     printf("You: ");
64     fgets(buffer, sizeof(buffer), stdin);
65
66     n = send(client_fd, buffer, strlen(buffer), 0);
67     if (n < 0) {
68         error("ERROR writing to socket");
69     }
70
71     if (strncmp(buffer, "Bye", 3) == 0) {
72         printf("Exiting Client...\n");
73         break;
74     }
75
76     bzero(buffer, sizeof(buffer));
77     n = read(client_fd, buffer, sizeof(buffer) - 1);
78     if (n < 0) {
79         error("ERROR reading from socket");
80     }

```

```

81     printf("Server: %s\n", buffer);
82
83 }
84
85 // Close the socket
86 close(client_fd);
87
88 return 0;
89 }

```

3.2 Sample Output

Screenshots:

```

abdullah@abdullah-7G-Series:~/Downloads$ gcc Server.c -o Server
abdullah@abdullah-7G-Series:~/Downloads$ ./Server
Server is running on port 9090...
Waiting for a connection...
Client connected.
Client: Hi
You: hi
Client: how u?
You: fine
Client: good
You: Bye
Exiting...
abdullah@abdullah-7G-Series:~/Downloads$ gcc Server.c -o Server
abdullah@abdullah-7G-Series:~/Downloads$ ./Server
Server is running on port 9090...
Waiting for a connection...
Client connected.
Client: hi
You: hi
Client: how you?
You: good
Client: ok
You: ok
Client: Bye
You: Bye
Exiting...
abdullah@abdullah-7G-Series:~/Downloads$

```

Figure 6: Sample Output of Server

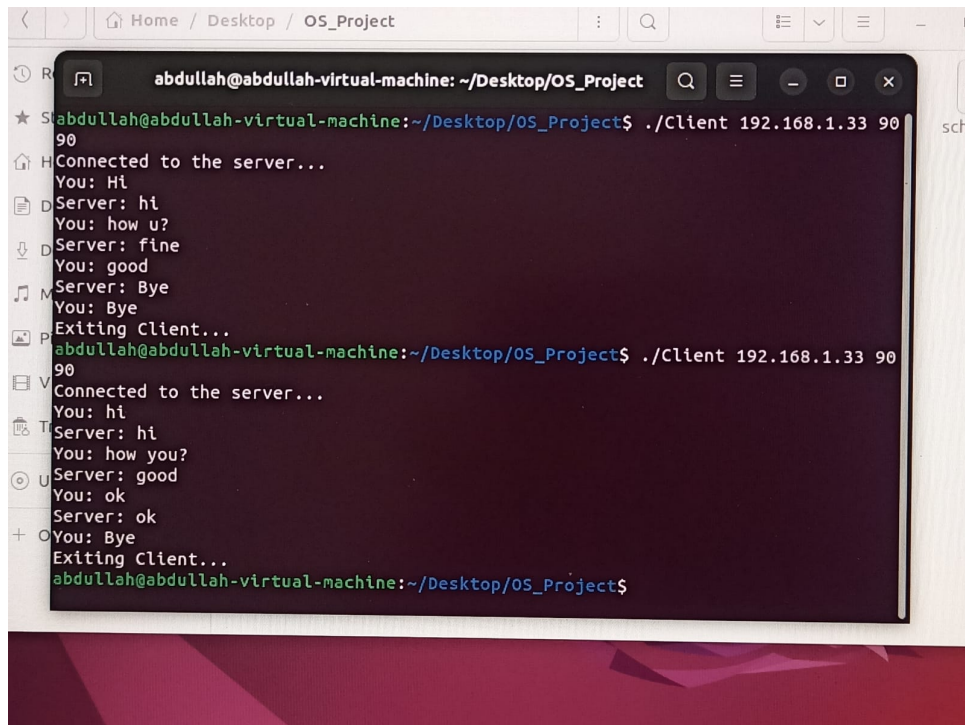
A screenshot of a terminal window titled 'abdullah@abdullah-virtual-machine: ~/Desktop/OS_Project'. The terminal shows two runs of a client program. In the first run, the user enters '90', the client connects to the server at 192.168.1.33, and they exchange messages: 'Hi', 'hi', 'how u?', 'fine', 'good', 'Bye', and 'Bye'. The client then exits. In the second run, the user enters '90' again, the client connects, and they exchange: 'hi', 'hi', 'how you?', 'good', 'ok', 'ok', 'Bye', and 'Bye'. The client exits again. The terminal background is dark with light-colored text.

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