

Operating Systems Project

Amber Khurshid

November 26, 2024

1 Task 1: Scheduling Algorithms

Task 1: CPU Scheduling Implementation In this task, you are required to implement several CPU scheduling algorithms in C. The algorithms include:

- First-Come-First-Served (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling
- Round Robin (RR)
- Priority with Round Robin

Each scheduling algorithm will handle a predefined set of tasks based on the given task's priority and CPU burst time. The task list will be provided via a `schedule.txt` file, and the program should read this file to create the schedule.

1.1 Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // Define task structure
6  typedef struct
7  {
8      char name[10];
9      int priority;
10     int cpuBurst;
11 } Task;
12
13 // Function prototypes
14 void loadTasks(Task tasks[], int *n);
15 void fcfs(Task tasks[], int n);
16 void sjf(Task tasks[], int n);
17 void priorityScheduling(Task tasks[], int n);
```

```

18 void roundRobin(Task tasks[], int n, int timeQuantum);
19 int compareBurst(const void *a, const void *b);
20 int comparePriority(const void *a, const void *b);
21
22 int main()
23 {
24     Task tasks[100];
25     int n;
26
27     // Load tasks from file
28     loadTasks(tasks, &n);
29
30     // Call each scheduling algorithm
31     printf("Task List:\n");
32     for (int i = 0; i < n; i++)
33     {
34         printf("%s (Priority: %d, CPU Burst: %d)\n", tasks[i]
35             .name, tasks[i].priority, tasks[i].cpuBurst);
36     }
37     printf("\n");
38
39     fcfs(tasks, n);
40     sjf(tasks, n);
41     priorityScheduling(tasks, n);
42     roundRobin(tasks, n, 5); // Example time quantum
43
44     return 0;
45 }
46
47 // Function to load tasks from schedule.txt
48 void loadTasks(Task tasks[], int *n)
49 {
50     FILE *file = fopen("schedule.txt", "r");
51     if (file == NULL)
52     {
53         printf("File does not exist");
54         exit(1);
55     }
56     *n = 0;
57     while (fscanf(file, "[%^,], %d, %d\n", tasks[*n].name, &
58         tasks[*n].priority, &tasks[*n].cpuBurst) != EOF)
59     {
60         (*n)++;
61     }
62     fclose(file);

```

```

61 }
62
63 // First-Come-First-Served Scheduling
64 void fcfs(Task tasks[], int n)
65 {
66     printf("FCFS Scheduling:\n");
67     int waitTime = 0;
68     int totalWait = 0;
69     for (int i = 0; i < n; i++)
70     {
71         printf("Task %s (Priority: %d, CPU Burst: %d)\n",
72             tasks[i].name, tasks[i].priority, tasks[i].
73             cpuBurst);
74         printf("Wait Time: %d\n", waitTime);
75         totalWait += waitTime;
76         waitTime += tasks[i].cpuBurst;
77     }
78     int average_wait_time = 0;
79     average_wait_time = totalWait / n;
80     printf("Average Wait Time: %.2d\n\n", average_wait_time);
81 }
82
83 // Shortest Job First Scheduling
84 void sjf(Task tasks[], int n)
85 {
86     // Sort tasks by burst time (ascending order)
87     for (int i = 0; i < n - 1; i++)
88     {
89         for (int j = 0; j < n - i - 1; j++)
90         {
91             if (tasks[j].cpuBurst > tasks[j + 1].cpuBurst)
92             {
93                 Task temp = tasks[j];
94                 tasks[j] = tasks[j + 1];
95                 tasks[j + 1] = temp;
96             }
97         }
98     }
99     printf("SJF Scheduling:\n");
100     int waitTime = 0;
101     int totalWait = 0;
102     for (int i = 0; i < n; i++)
103     {
104         printf("Task %s (Priority: %d, CPU Burst: %d)\n",
105             tasks[i].name, tasks[i].priority, tasks[i].

```

```

        cpuBurst);
103     printf("Wait Time: %d\n", waitTime);
104     totalWait += waitTime;
105     waitTime += tasks[i].cpuBurst;
106 }
107 int average_wait_time = 0;
108 average_wait_time = totalWait / n;
109 printf("Average Wait Time: %.2d\n\n", average_wait_time);
110 }
111
112 // Priority Scheduling
113
114 void priorityScheduling(Task tasks[], int n)
115 {
116     printf("Priority Scheduling:\n");
117
118     for (int i = 0; i < n - 1; i++) {
119         for (int j = 0; j < n - i - 1; j++) {
120             if (tasks[j].priority < tasks[j + 1].priority ||
121                 (tasks[j].priority == tasks[j + 1].priority &&
122                  strcmp(tasks[j].name, tasks[j + 1].name) >
123                     0)) {
124                 Task temp = tasks[j];
125                 tasks[j] = tasks[j + 1];
126                 tasks[j + 1] = temp;
127             }
128         }
129     }
130
131     int waitTime = 0;
132     int totalWait = 0;
133     for (int i = 0; i < n; i++)
134     {
135         printf("Task %s (Priority: %d, CPU Burst: %d)\n",
136             tasks[i].name, tasks[i].priority, tasks[i].
137             cpuBurst);
138         printf("Wait Time: %d\n", waitTime);
139         totalWait += waitTime;
140         waitTime += tasks[i].cpuBurst;
141     }
142
143     int average_wait_time = 0;
144     average_wait_time = totalWait / n;
145     printf("Average Wait Time: %.2d\n\n", average_wait_time);
146 }

```

```

143 // Round Robin Scheduling
144 void roundRobin(Task tasks[], int n, int timeQuantum)
145 {
146     printf("Round Robin Scheduling (Time Quantum: %d):\n",
147           timeQuantum);
148     int remaining[n];
149     for (int i = 0; i < n; i++)
150         remaining[i] = tasks[i].cpuBurst;
151
152     int time = 0;
153     int done = 0;
154     while (done < n)
155     {
156         for (int i = 0; i < n; i++)
157         {
158             if (remaining[i] > 0)
159             {
160                 if (remaining[i] <= timeQuantum)
161                 {
162                     time += remaining[i];
163                     printf("Task %s completed at time %d\n",
164                           tasks[i].name, time);
165                     remaining[i] = 0;
166                     done++;
167                 }
168                 else
169                 {
170                     time += timeQuantum;
171                     remaining[i] -= timeQuantum;
172                     printf("Task %s processed until time %d\n",
173                           tasks[i].name, time);
174                 }
175             }
176         }
177     }
178     printf("\n");
179 }

```

Listing 1: Scheduling Algorithms Code

```

amber@amber-HP-EliteBook-840-G3: ~/Desktop/OS Project
amber@amber-HP-EliteBook-840-G3:~/Desktop/OS Project $ gcc task_1.c -o task_1
amber@amber-HP-EliteBook-840-G3:~/Desktop/OS Project $ ./task_1
Task List:
T1 (Priority: 4, CPU Burst: 20)
T2 (Priority: 2, CPU Burst: 25)
T3 (Priority: 3, CPU Burst: 25)
T4 (Priority: 3, CPU Burst: 15)
T5 (Priority: 10, CPU Burst: 10)

FCFS Scheduling:
Task T1 (Priority: 4, CPU Burst: 20)
Wait Time: 0
Task T2 (Priority: 2, CPU Burst: 25)
Wait Time: 20
Task T3 (Priority: 3, CPU Burst: 25)
Wait Time: 45
Task T4 (Priority: 3, CPU Burst: 15)
Wait Time: 70
Task T5 (Priority: 10, CPU Burst: 10)
Wait Time: 85
Average Wait Time: 44

```

Figure 1: Output of First-Come-First-Served Scheduling

```

amber@amber-HP-EliteBook-840-G3: ~/Desktop/OS Project
Task T5 (Priority: 10, CPU Burst: 10)
Wait Time: 85
Average Wait Time: 44

SJF Scheduling:
Task T5 (Priority: 10, CPU Burst: 10)
Wait Time: 0
Task T4 (Priority: 3, CPU Burst: 15)
Wait Time: 10
Task T1 (Priority: 4, CPU Burst: 20)
Wait Time: 25
Task T2 (Priority: 2, CPU Burst: 25)
Wait Time: 45
Task T3 (Priority: 3, CPU Burst: 25)
Wait Time: 70
Average Wait Time: 30

```

Figure 2: Output of Shortest Job First Scheduling

```

amber@amber-HP-EliteBook-840-G3: ~/Desktop/OS Project
Priority Scheduling:
Task T5 (Priority: 10, CPU Burst: 10)
Wait Time: 0
Task T1 (Priority: 4, CPU Burst: 20)
Wait Time: 10
Task T3 (Priority: 3, CPU Burst: 25)
Wait Time: 30
Task T4 (Priority: 3, CPU Burst: 15)
Wait Time: 55
Task T2 (Priority: 2, CPU Burst: 25)
Wait Time: 70
Average Wait Time: 33

```

Figure 3: Output of Priority Scheduling

2 Task 2: Local System Socket Programming

2.1 Server Code

The following C code implements the server-side of a simple client-server application using sockets and threads.⁶

```
amber@amber-HP-EliteBook-840-G3: ~/Desktop/OS Project
Average Wait Time: 33

Round Robin Scheduling (Time Quantum: 5):
Task T5 processed until time 5
Task T1 processed until time 10
Task T3 processed until time 15
Task T4 processed until time 20
Task T2 processed until time 25
Task T5 completed at time 30
Task T1 processed until time 35
Task T3 processed until time 40
Task T4 processed until time 45
Task T2 processed until time 50
Task T1 processed until time 55
Task T3 processed until time 60
Task T4 completed at time 65
Task T2 processed until time 70
Task T1 completed at time 75
Task T3 processed until time 80
Task T2 processed until time 85
Task T3 completed at time 90
Task T2 completed at time 95

amber@amber-HP-EliteBook-840-G3:~/Desktop/OS Project $
```

Figure 4: Output of Round Robin Scheduling

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6  #include <pthread.h>
7
8  #define MAX_CLIENTS 5
9  #define PORT 8080
10
11 int client_socket[MAX_CLIENTS] = {0};
12 pthread_mutex_t lock;
13 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
14 int ready_to_prompt = 0;
15
16 void *client_handler(void *socket_desc);
17 void *server_sender(void *arg);
18
19 int main() {
20     int server_fd, new_socket, addr_len;
21     struct sockaddr_in server_addr, client_addr;
22     pthread_t sender_thread;
23
24     // Initialize socket
25     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
26         perror("Socket failed");
```

```

27         exit(EXIT_FAILURE);
28     }
29
30     server_addr.sin_family = AF_INET;
31     server_addr.sin_addr.s_addr = INADDR_ANY;
32     server_addr.sin_port = htons(PORT);
33
34     if (bind(server_fd, (struct sockaddr *)&server_addr,
35             sizeof(server_addr)) < 0) {
36         perror("Bind failed");
37         exit(EXIT_FAILURE);
38     }
39
40     if (listen(server_fd, 3) < 0) {
41         perror("Listen failed");
42         exit(EXIT_FAILURE);
43     }
44
45     // Create a thread for the server to send messages
46     pthread_create(&sender_thread, NULL, server_sender, NULL);
47
48     ;
49
50     printf("Server started. Waiting for clients...\n");
51
52     while (1) {
53         addr_len = sizeof(client_addr);
54         if ((new_socket = accept(server_fd, (struct sockaddr
55             *)&client_addr, (socklen_t*)&addr_len)) < 0) {
56             perror("Accept failed");
57             exit(EXIT_FAILURE);
58         }
59
60         // Handle client communication in a new thread
61         pthread_t client_thread;
62         pthread_create(&client_thread, NULL, client_handler,
63             (void *)&new_socket);
64     }
65
66     return 0;
67 }
68
69 // Function to handle communication with clients
70 void *client_handler(void *socket_desc) {
71     int sock = *(int*)socket_desc;
72     char buffer[1024];

```



```

68     int bytes_read;
69
70     // Add client socket to client list
71     pthread_mutex_lock(&lock);
72     for (int i = 0; i < MAX_CLIENTS; i++) {
73         if (client_socket[i] == 0) {
74             client_socket[i] = sock;
75             break;
76         }
77     }
78     pthread_mutex_unlock(&lock);
79
80     while ((bytes_read = read(sock, buffer, sizeof(buffer)))
81           > 0) {
82         buffer[bytes_read] = '\0';
83         printf("Message from client: %s\n", buffer);
84
85         // Signal the server to send broadcast message
86         pthread_mutex_lock(&lock);
87         ready_to_prompt = 1;
88         pthread_cond_signal(&cond);
89         pthread_mutex_unlock(&lock);
90     }
91
92     close(sock);
93     pthread_mutex_lock(&lock);
94     for (int i = 0; i < MAX_CLIENTS; i++) {
95         if (client_socket[i] == sock) {
96             client_socket[i] = 0;
97             break;
98         }
99     }
100     pthread_mutex_unlock(&lock);
101
102     pthread_exit(NULL);
103 }
104
105 // Server sender thread
106 void *server_sender(void *arg) {
107     char buffer[1024];
108     while (1) {
109         pthread_mutex_lock(&lock);
110         while (ready_to_prompt == 0) {
111             pthread_cond_wait(&cond, &lock);

```

```

112         ready_to_prompt = 0;
113         pthread_mutex_unlock(&lock);
114
115         printf("Enter message to broadcast to clients: ");
116         fgets(buffer, sizeof(buffer), stdin);
117         buffer[strcspn(buffer, "\n")] = '\0';    % Remove
            newline
118
119         pthread_mutex_lock(&lock);
120         for (int i = 0; i < MAX_CLIENTS; i++) {
121             if (client_socket[i] != 0) {
122                 send(client_socket[i], buffer, strlen(buffer)
123                     , 0);
124             }
125         pthread_mutex_unlock(&lock);
126     }
127 }

```

129 \subsection{Client Code}

131 The following C code implements the client-side of the client
-server communication program.

```

133 \begin{lstlisting}[language=C, caption={Client Code for
134 Client-Server Communication}]
135 #include <stdio.h>
136 #include <stdlib.h>
137 #include <string.h>
138 #include <sys/socket.h>
139 #include <arpa/inet.h>
140 #include <unistd.h>
141
142 #define PORT 8080
143
144 int main() {
145     int sock = 0;
146     struct sockaddr_in serv_addr;
147     char buffer[1024] = {0};
148
149     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
150         perror("Socket creation failed");
151         return -1;
152     }

```

```

153
154     serv_addr.sin_family = AF_INET;
155     serv_addr.sin_port = htons(PORT);
156     char *server_ip = "192.168.43.209";    % Replace with
        server IP address
157
158     if (inet_pton(AF_INET, server_ip, &serv_addr.sin_addr) <=
        0) {
159         perror("Invalid address");
160         return -1;
161     }
162
163     if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(
        serv_addr)) < 0) {
164         perror("Connection failed");
165         return -1;
166     }
167
168     printf("Connected to server\n");
169
170     while (1) {
171         printf("Enter message: ");
172         fgets(buffer, 1024, stdin);
173
174         send(sock, buffer, strlen(buffer), 0);
175         memset(buffer, 0, sizeof(buffer));
176
177         int valread = read(sock, buffer, 1024);
178         if (valread > 0) {
179             printf("Server: %s\n", buffer);
180         }
181     }
182
183     return 0;
184 }

```

Listing 2: Server Code for Client-Server Communication

3 Task 2: Distributed System Socket Programming

3.1 Server Code

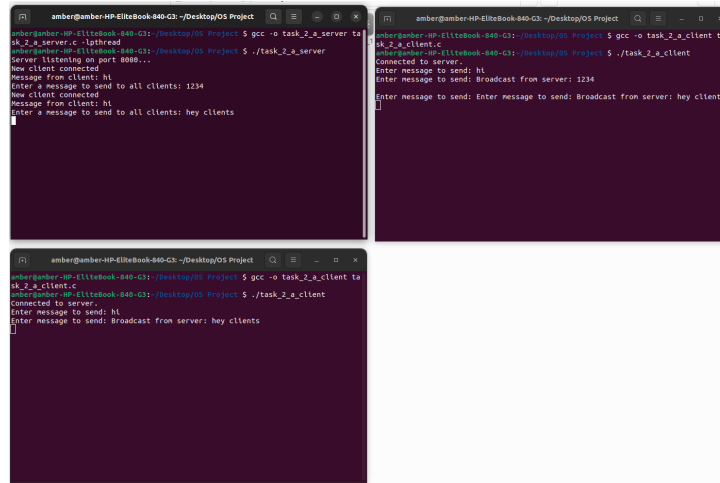


Figure 5: Output of the Client-Server Communication

```

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 8080
9
10 int main() {
11     int server_fd, new_socket, addrlen = sizeof(struct
        sockaddr_in);
12     struct sockaddr_in address;
13     char buffer[1024] = {0};
14
15     // Create socket
16     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
17         perror("Socket creation failed");
18         return -1;
19     }
20
21     address.sin_family = AF_INET;
22     address.sin_addr.s_addr = INADDR_ANY; // Accept
        connections from any IP address
23     address.sin_port = htons(PORT); // The port we will
        listen on

```

```

24
25 // Bind socket to the address and port
26 if (bind(server_fd, (struct sockaddr*)&address, sizeof(
27     address)) < 0) {
28     perror("Bind failed");
29     return -1;
30 }
31 // Listen for incoming connections
32 if (listen(server_fd, 3) < 0) {
33     perror("Listen failed");
34     return -1;
35 }
36
37 printf("Server listening on port %d...\n", PORT);
38
39 // Accept a client connection
40 if ((new_socket = accept(server_fd, (struct sockaddr*)&
41     address, (socklen_t*)&addrlen)) < 0) {
42     perror("Accept failed");
43     return -1;
44 }
45
46 printf("Client connected\n");
47
48 // Handle communication with the client
49 while (1) {
50     // Read message from client
51     int valread = read(new_socket, buffer, 1024);
52     if (valread <= 0) {
53         printf("Client disconnected or error occurred\n");
54         ;
55         break;
56     }
57
58     printf("Client: %s\n", buffer);
59
60     // Send a response back to the client
61     send(new_socket, "Message received", strlen("Message
62         received"), 0);
63 }
64
65 // Close the connection
66 close(new_socket);
67 close(server_fd);

```

```

65     return 0;
66 }

```

Listing 3: Server Code for Client-Server Communication

3.2 Client 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 8080
9
10 int main() #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <sys/socket.h>
14 #include <arpa/inet.h>
15 #include <unistd.h>
16
17 #define PORT 8080
18
19 int main() {
20     int sock = 0;
21     struct sockaddr_in serv_addr;
22     char buffer[1024] = {0};
23
24     // Create socket
25     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
26         perror("Socket creation failed");
27         return -1;
28     }
29
30     serv_addr.sin_family = AF_INET;
31     serv_addr.sin_port = htons(PORT);
32
33     char *server_ip = "192.168.43.209"; // Replace with the
                                         server laptop's IP address
34     if (inet_pton(AF_INET, server_ip, &serv_addr.sin_addr) <=
35         0) {
36         perror("Invalid address");
37     }
38 }

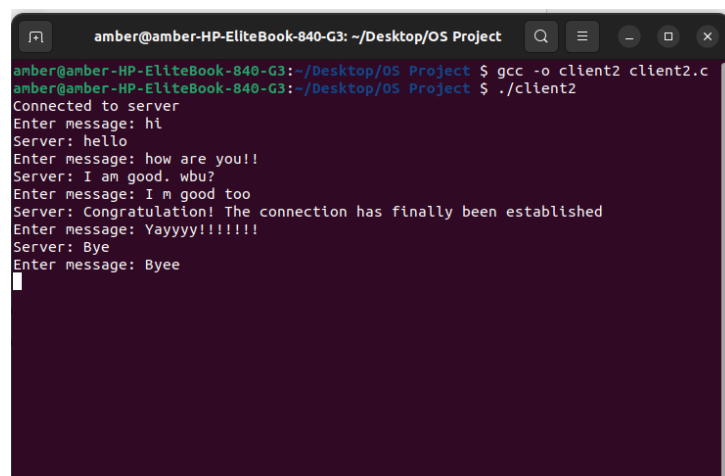
```

```

36         return -1;
37     }
38
39     // Connect to the server
40     if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(
41         serv_addr)) < 0) {
42         perror("Connection failed");
43         return -1;
44     }
45
46     printf("Connected to server\n");
47
48     // Send a message to the server
49     while (1) {
50         printf("Enter message: ");
51         fgets(buffer, 1024, stdin);
52
53         // Send message to the server
54         send(sock, buffer, strlen(buffer), 0);
55
56         // Clear the buffer
57         memset(buffer, 0, sizeof(buffer));
58
59         // Receive a message from the server
60         int valread = read(sock, buffer, 1024);
61         if (valread > 0) {
62             printf("Server: %s\n", buffer);
63         }
64     }
65
66     return 0;

```

Listing 4: Client Code for Client-Server Communication

A terminal window with a dark purple background and white text. The window title is "amber@amber-HP-EliteBook-840-G3: ~/Desktop/OS Project". The terminal shows the following sequence of commands and output:

```
amber@amber-HP-EliteBook-840-G3:~/Desktop/OS Project $ gcc -o client2 client2.c
amber@amber-HP-EliteBook-840-G3:~/Desktop/OS Project $ ./client2
Connected to server
Enter message: hi
Server: hello
Enter message: how are you!!
Server: I am good. wbu?
Enter message: I n good too
Server: Congratulation! The connection has finally been established
Enter message: Yayyyy!!!!!!
Server: Bye
Enter message: Byee
```

Figure 6: Output of the Client-Server Communication