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

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

// Define task structure
typedef struct
{
    char name[10];
    int priority;
    int cpuBurst;
} Task;

// Function prototypes
void loadTasks(Task tasks[], int *n);
void fcfs(Task tasks[], int n);
void sjf(Task tasks[], int n);
void priorityScheduling(Task tasks[], int n);
```

```c
void roundRobin(Task tasks[], int n, int timeQuantum);
int compareBurst(const void *a, const void *b);
int comparePriority(const void *a, const void *b);

int main()
{
    Task tasks[100];
    int n;

    // Load tasks from file
    loadTasks(tasks, &n);

    // Call each scheduling algorithm
    printf("Task List:\n");
    for (int i = 0; i < n; i++)
    {
        printf("%s (Priority: %d, CPU Burst: %d)\n", tasks[i
            ].name, tasks[i].priority, tasks[i].cpuBurst);
    }
    printf("\n");

    fcfs(tasks, n);
    sjf(tasks, n);
    priorityScheduling(tasks, n);
    roundRobin(tasks, n, 5); // Example time quantum

    return 0;
}

// Function to load tasks from schedule.txt
void loadTasks(Task tasks[], int *n)
{
    FILE *file = fopen("schedule.txt", "r");
    if (file == NULL)
    {
        printf("File does not exist");
        exit(1);
    }
    *n = 0;
    while (fscanf(file, "%[^,], %d, %d\n", tasks[*n].name, &
        tasks[*n].priority, &tasks[*n].cpuBurst) != EOF)
    {
        (*n)++;
    }
    fclose(file);
```

```
}

// First-Come-First-Served Scheduling
void fcfs(Task tasks[], int n)
{
    printf("FCFS Scheduling:\n");
    int waitTime = 0;
    int totalWait = 0;
    for (int i = 0; i < n; i++)
    {
        printf("Task %s (Priority: %d, CPU Burst: %d)\n",
            tasks[i].name, tasks[i].priority, tasks[i].
            cpuBurst);
        printf("Wait Time: %d\n", waitTime);
        totalWait += waitTime;
        waitTime += tasks[i].cpuBurst;
    }
    int average_wait_time = 0;
    average_wait_time = totalWait / n;
    printf("Average Wait Time: %.2d\n\n", average_wait_time);
}

// Shortest Job First Scheduling
void sjf(Task tasks[], int n)
{
    // Sort tasks by burst time (ascending order)
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (tasks[j].cpuBurst > tasks[j + 1].cpuBurst)
            {
                Task temp = tasks[j];
                tasks[j] = tasks[j + 1];
                tasks[j + 1] = temp;
            }
        }
    }
    printf("SJF Scheduling:\n");
    int waitTime = 0;
    int totalWait = 0;
    for (int i = 0; i < n; i++)
    {
        printf("Task %s (Priority: %d, CPU Burst: %d)\n",
            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 &&
                       strcmp(tasks[j].name, tasks[j + 1].name) >
                       0)) {
122                 Task temp = tasks[j];
123                 tasks[j] = tasks[j + 1];
124                 tasks[j + 1] = temp;
125             }
126         }
127     }
128
129     int waitTime = 0;
130     int totalWait = 0;
131     for (int i = 0; i < n; i++)
132     {
133         printf("Task %s (Priority: %d, CPU Burst: %d)\n",
               tasks[i].name, tasks[i].priority, tasks[i].
               cpuBurst);
134         printf("Wait Time: %d\n", waitTime);
135         totalWait += waitTime;
136         waitTime += tasks[i].cpuBurst;
137     }
138     int average_wait_time = 0;
139     average_wait_time = totalWait / n;
140     printf("Average Wait Time: %.2d\n\n", average_wait_time);
141 }
142
```

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

Listing 1: Scheduling Algorithms Code

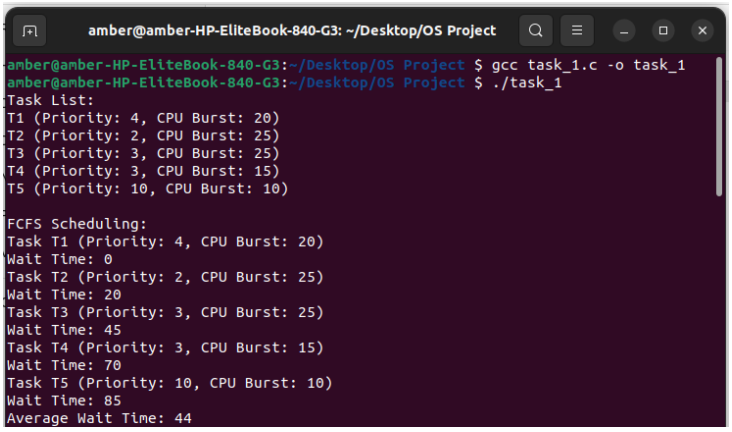

Figure 1: Output of First-Come-First-Served Scheduling

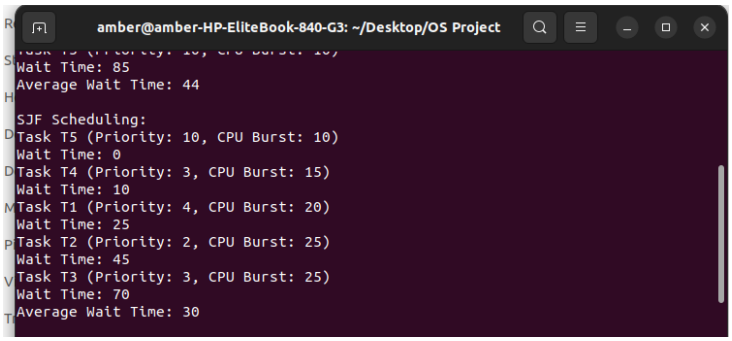

Figure 2: Output of Shortest Job First Scheduling

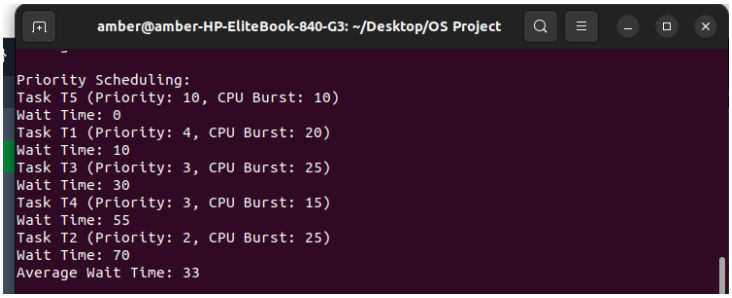

Figure 3: Output of Priority Scheduling

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.

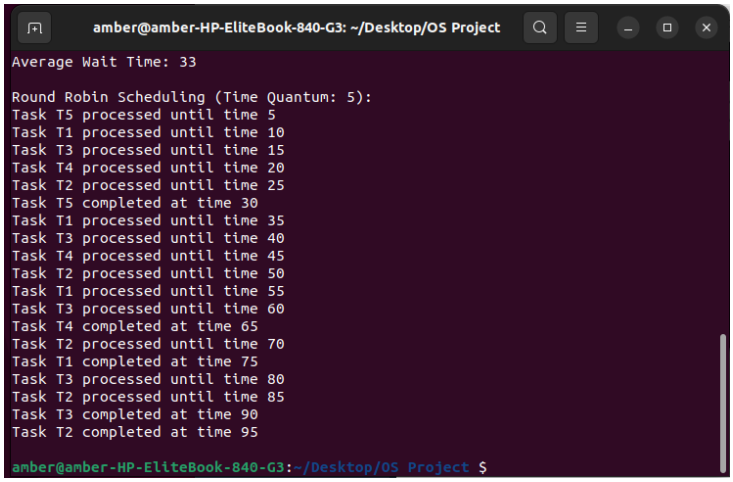

Figure 4: Output of Round Robin Scheduling

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

#define MAX_CLIENTS 5
#define PORT 8080

int client_socket[MAX_CLIENTS] = {0};
pthread_mutex_t lock;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int ready_to_prompt = 0;

void *client_handler(void *socket_desc);
void *server_sender(void *arg);

int main() {
    int server_fd, new_socket, addr_len;
    struct sockaddr_in server_addr, client_addr;
    pthread_t sender_thread;

    // Initialize socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        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,
            sizeof(server_addr)) < 0) {
35          perror("Bind failed");
36          exit(EXIT_FAILURE);
37      }
38
39      if (listen(server_fd, 3) < 0) {
40          perror("Listen failed");
41          exit(EXIT_FAILURE);
42      }
43
44      // Create a thread for the server to send messages
45      pthread_create(&sender_thread, NULL, server_sender, NULL)
            ;
46
47      printf("Server started. Waiting for clients...\n");
48
49      while (1) {
50          addr_len = sizeof(client_addr);
51          if ((new_socket = accept(server_fd, (struct sockaddr
                *)&client_addr, (socklen_t*)&addr_len)) < 0) {
52              perror("Accept failed");
53              exit(EXIT_FAILURE);
54          }
55
56          // Handle client communication in a new thread
57          pthread_t client_thread;
58          pthread_create(&client_thread, NULL, client_handler,
                (void *)&new_socket);
59      }
60
61      return 0;
62  }
63
64  // Function to handle communication with clients
65  void *client_handler(void *socket_desc) {
66      int sock = *(int*)socket_desc;
67      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)))
            > 0) {
81          buffer[bytes_read] = '\0';
82          printf("Message from client: %s\n", buffer);
83
84          // Signal the server to send broadcast message
85          pthread_mutex_lock(&lock);
86          ready_to_prompt = 1;
87          pthread_cond_signal(&cond);
88          pthread_mutex_unlock(&lock);
89      }
90
91      close(sock);
92      pthread_mutex_lock(&lock);
93      for (int i = 0; i < MAX_CLIENTS; i++) {
94          if (client_socket[i] == sock) {
95              client_socket[i] = 0;
96              break;
97          }
98      }
99      pthread_mutex_unlock(&lock);
100
101     pthread_exit(NULL);
102 }
103
104 // Server sender thread
105 void *server_sender(void *arg) {
106     char buffer[1024];
107     while (1) {
108         pthread_mutex_lock(&lock);
109         while (ready_to_prompt == 0) {
110             pthread_cond_wait(&cond, &lock);
111         }
```

```
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)
                       , 0);
123             }
124         }
125         pthread_mutex_unlock(&lock);
126     }
127 }
128 
129 
130 \subsection{Client Code}
131 
132 The following C code implements the client-side of the client
       -server communication program.
133 
134 \begin{lstlisting}[language=C, caption={Client Code for
       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

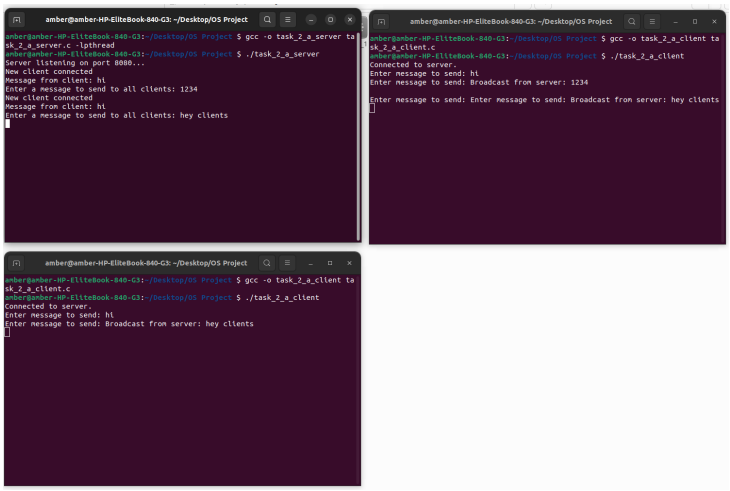

Figure 5: Output of the Client-Server Communication

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

#define PORT 8080

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

    // Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        return -1;
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;  // Accept
        connections from any IP address
    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(
            address)) < 0) {
27          perror("Bind failed");
28          return -1;
29      }
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*)&
            address, (socklen_t*)&addrlen)) < 0) {
41          perror("Accept failed");
42          return -1;
43      }
44
45      printf("Client connected\n");
46
47      // Handle communication with the client
48      while (1) {
49          // Read message from client
50          int valread = read(new_socket, buffer, 1024);
51          if (valread <= 0) {
52              printf("Client disconnected or error occurred\n")
                    ;
53              break;
54          }
55
56          printf("Client: %s\n", buffer);
57
58          // Send a response back to the client
59          send(new_socket, "Message received", strlen("Message
                received"), 0);
60      }
61
62      // Close the connection
63      close(new_socket);
64      close(server_fd);
```

```
    return 0;
}
```

Listing 3: Server Code for Client-Server Communication

3.2 Client Code

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

#define PORT 8080

int main() #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 8080

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    char *server_ip = "192.168.43.209";  // Replace with the
       server laptop's IP address
    if (inet_pton(AF_INET, server_ip, &serv_addr.sin_addr) <=
        0) {
        perror("Invalid address");
```

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

Listing 4: Client Code for Client-Server Communication

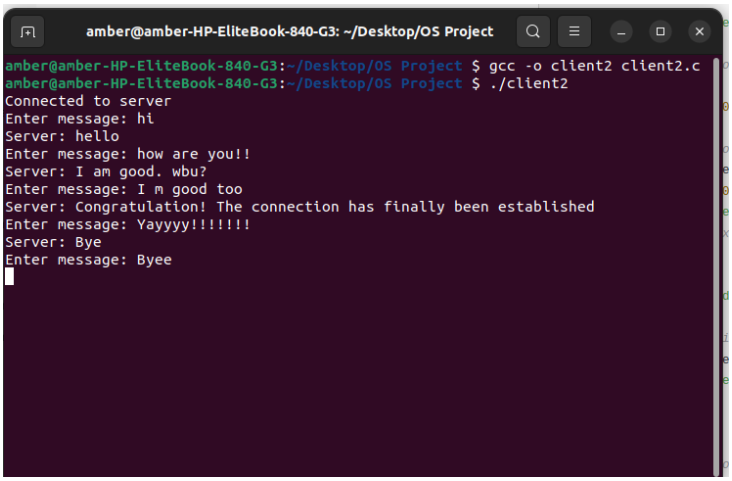

Figure 6: Output of the Client-Server Communication