OS_Project

22P9316 Ahmad Irtaza Meer

November 2024

Task 1: CPU Scheduling Implementation

```
#include <bits/stdc++.h>
using namespace std;

struct Task {
    string name;
    int priority;
    int burst;
};

vector<Task> readTasksFromFile(const string& filename) {
    vector<Task> tasks;
    ifstream file(filename);
    string line;

    while (getline(file, line)) {
        istringstream ss(line);
        Task task;

        getline(ss, task.name, ',');
        ss >> task.priority;
        ss.ignore();
        ss >> task.burst;

        // Trim whitespace around task name
        task.name.erase(task.name.find_last_not_of(" \t\n\r") + 1);

        tasks.push_back(task);
    }

    return tasks;
}

void fcfs(const vector<Task>& tasks) {
    cout << "FCFS Scheduling:\n";
    for (const auto& task : tasks) {
        cout << "Executing " << task.name << " (Burst: " << task.burst << ")\n";
    }
}

void sjf(vector<Task> tasks) {
```

```
    cout << "SJF␣Scheduling:\n";
    sort(tasks.begin(), tasks.end(), [](const Task& a, const Task& b) {
        return a.burst < b.burst;
    });

    for (const auto& task : tasks) {
        cout << "Executing␣" << task.name << "␣(Burst:␣" << task.burst << ")\n";
    }
}

void priorityScheduling(vector<Task> tasks) {
    cout << "Priority␣Scheduling:\n";
    sort(tasks.begin(), tasks.end(), [](const Task& a, const Task& b) {
        return a.priority > b.priority; // Higher priority executed first
    });

    for (const auto& task : tasks) {
        cout << "Executing␣" << task.name << "␣(Priority:␣" << task.priority << "
     ,␣Burst:␣" << task.burst << ")\n";
    }
}

void roundRobin(vector<Task> tasks, int quantum) {
    cout << "Round␣Robin␣Scheduling␣(Quantum:␣" << quantum << "):\n";

    while (!tasks.empty()) {
        for (auto it = tasks.begin(); it != tasks.end();) {
            if (it->burst <= quantum) {
                cout << "Executing␣" << it->name << "␣(Burst␣Remaining:␣" << it->
     burst << ")\n";
                it = tasks.erase(it);
            } else {
                cout << "Executing␣" << it->name << "␣for␣" << quantum << "␣time␣
     units\n";
                it->burst -= quantum;
                ++it;
            }
        }
    }
}

void priorityWithRoundRobin(vector<Task> tasks, int quantum) {
    cout << "Priority␣with␣Round␣Robin␣Scheduling␣(Quantum:␣" << quantum << "):\n
     ";

    sort(tasks.begin(), tasks.end(), [](const Task& a, const Task& b) {
        return a.priority > b.priority;
    });

    while (!tasks.empty()) {
        for (auto it = tasks.begin(); it != tasks.end();) {
            if (it->burst <= quantum) {
                cout << "Executing␣" << it->name << "␣(Priority:␣" << it->
     priority << ",␣Burst␣Remaining:␣" << it->burst << ")\n";
                it = tasks.erase(it);
            } else {
```

```
                cout << "Executing " << it->name << " for " << quantum << " time
    units\n";
                it->burst -= quantum;
                ++it;
            }
        }
    }
}

int main() {
    vector<Task> tasks = readTasksFromFile("schedule.txt");
    int quantum = 5;  // Define a time quantum for Round Robin algorithms

    fcfs(tasks);
    cout << "\n";
    sjf(tasks);
    cout << "\n";
    priorityScheduling(tasks);
    cout << "\n";
    roundRobin(tasks, quantum);
    cout << "\n";
    priorityWithRoundRobin(tasks, quantum);

    return 0;
}
```

```
Type help for instructions on how to use fish
irtazabutt@irtazabutt ~/D/S/flutter_application_2> cd "/home/irtazabutt/Desktop/Semster_V/OS_Theory/Project/" && g++ 2.cpp -o 2 && "/home/irtazabutt/Deskt
op/Semster_V/OS_Theory/Project/"2
FCFS Scheduling:
Executing T1 (Burst: 20)
Executing T2 (Burst: 25)
Executing T3 (Burst: 25)
Executing T4 (Burst: 15)
Executing T5 (Burst: 10)

SJF Scheduling:
Executing T5 (Burst: 10)
Executing T4 (Burst: 15)
Executing T1 (Burst: 20)
Executing T2 (Burst: 25)
Executing T3 (Burst: 25)

Priority Scheduling:
Executing T5 (Priority: 10, Burst: 10)
Executing T1 (Priority: 4, Burst: 20)
Executing T3 (Priority: 3, Burst: 25)
Executing T4 (Priority: 3, Burst: 15)
Executing T2 (Priority: 2, Burst: 25)
```

Figure 1: FCFS,SJF and Priority Scheduling

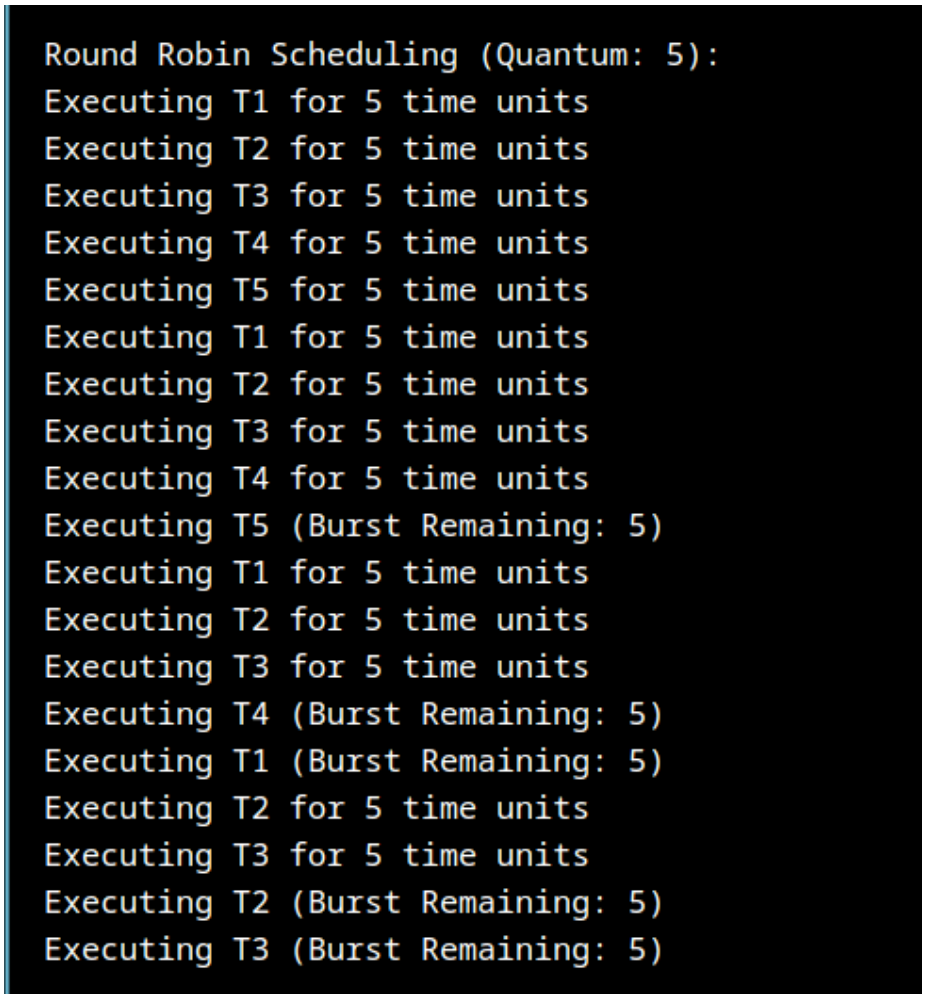

Figure 2: Round Robin Scheduling

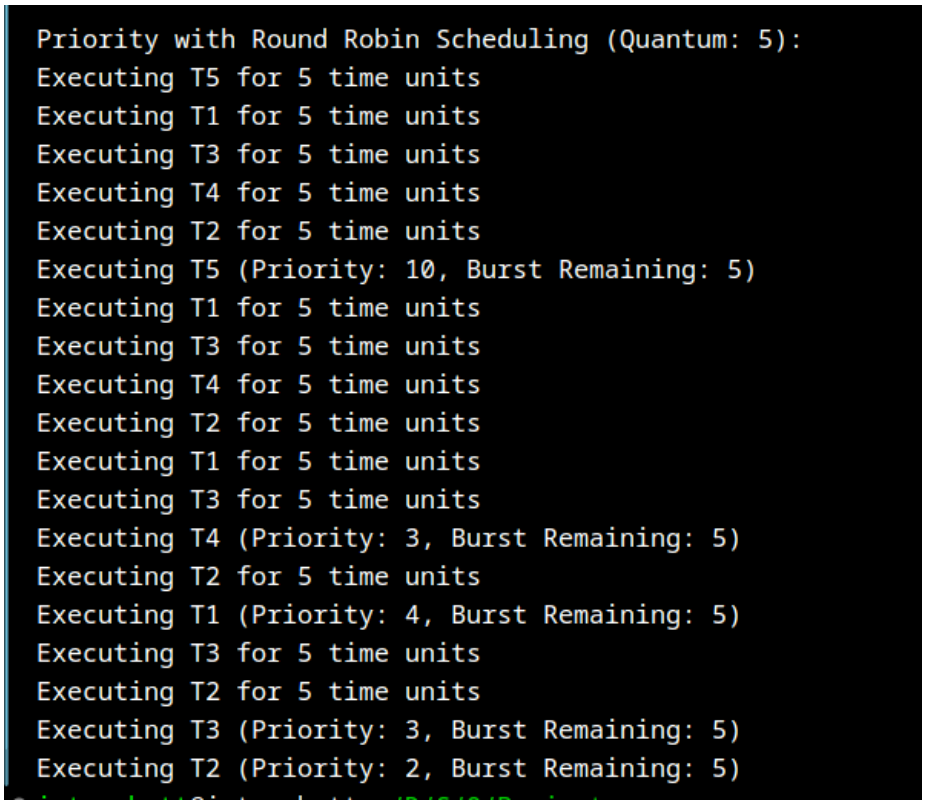

Figure 3: Round Robin with prority Scheduling

Task 2: Part 1: Local System Socket Programming

Server Code

```
#include <iostream>
#include <vector>
#include <thread>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <cstring>
#include <algorithm>  // Ensure std::remove is recognized

#define PORT 8082

// Function to handle communication with clients
void handle_client(int client_socket, std::vector<int>& clients) {
    char buffer[1024];
    int read_size;

    while ((read_size = read(client_socket, buffer, sizeof(buffer))) > 0) {
        buffer[read_size] = '\0';
        std::cout << "Received from client: " << buffer << std::endl;

        // Broadcast the message to all clients
        for (int client : clients) {
            if (client != client_socket) {
                send(client, buffer, strlen(buffer), 0);
            }
        }
    }

    // Remove client from the list when it disconnects
    close(client_socket);

    // Correct use of std::remove and erase
    clients.erase(std::remove(clients.begin(), clients.end(), client_socket),
     clients.end());
}

// Main server function
int main() {
    int server_fd, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    std::vector<int> clients;

    // Create server socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket
    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
      {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Server listening on port " << PORT << std::endl;

    while (true) {
        // Accept incoming client connections
        if ((client_socket = accept(server_fd, (struct sockaddr*)&client_addr, &
     client_addr_len)) < 0) {
            perror("Accept failed");
            continue;
        }

        std::cout << "New client connected" << std::endl;

        // Add client to the client list
        clients.push_back(client_socket);

        // Create a thread to handle client communication
        std::thread(handle_client, client_socket, std::ref(clients)).detach();
    }

    return 0;
}
```

Client Code

```
#include <iostream>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <cstring>
#include <thread>

#define SERVER_IP "127.0.0.1"
#define PORT 8082

// Function to receive broadcasted messages
void receive_messages(int socket_fd) {
    char buffer[1024];
    int read_size;
```

```
    while ((read_size = read(socket_fd, buffer, sizeof(buffer))) > 0) {
        buffer[read_size] = '\0';
        std::cout << "Message from server: " << buffer << std::endl;
    }
}

// Main client function
int main() {
    int socket_fd;
    struct sockaddr_in server_addr;

    // Create client socket
    if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

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

    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(socket_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) <
      0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connected to server!" << std::endl;

    // Start a thread to receive messages from the server
    std::thread(receive_messages, socket_fd).detach();

    // Send messages to the server
    char message[1024];
    while (true) {
        std::cout << "Enter message: ";
        std::cin.getline(message, sizeof(message));

        if (strcmp(message, "exit") == 0) {
            break;
        }

        send(socket_fd, message, strlen(message), 0);
    }

    close(socket_fd);
    return 0;
}
```

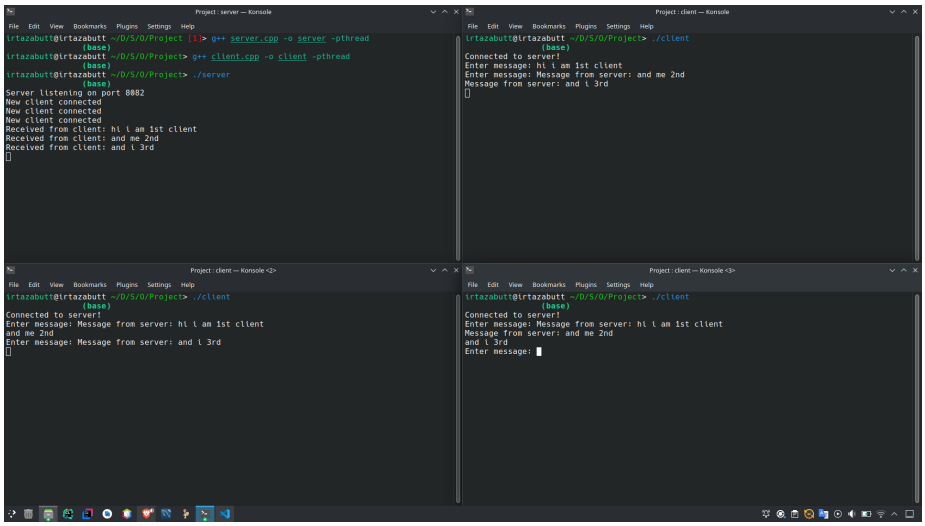

Figure 4: Server and Client on local system

Part 2: Distributed System Socket Programming

0.1 Server Code

```
    #include <iostream>
#include <vector>
#include <thread>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <cstring>
#include <algorithm>  // for std::remove

#define PORT 8085

// Function to handle communication with clients
void handle_client(int client_socket, std::vector<int>& clients) {
    char buffer[1024];
    int read_size;

    while ((read_size = read(client_socket, buffer, sizeof(buffer))) > 0) {
        buffer[read_size] = '\0';
        std::cout << "Received from client: " << buffer << std::endl;

        // Broadcast the message to all clients
        for (int client : clients) {
            if (client != client_socket) {
                send(client, buffer, strlen(buffer), 0);
            }
        }
    }
```

```
    // Remove client from the list when it disconnects
    close(client_socket);
    clients.erase(std::remove(clients.begin(), clients.end(), client_socket),
     clients.end());
}

// Main server function
int main() {
    int server_fd, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    std::vector<int> clients;

    // Create server socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket to a specific port
    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
     {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Server listening on port " << PORT << std::endl;

    while (true) {
        // Accept incoming client connections
        if ((client_socket = accept(server_fd, (struct sockaddr*)&client_addr, &
     client_addr_len)) < 0) {
            perror("Accept failed");
            continue;
        }

        std::cout << "New client connected" << std::endl;

        // Add client to the client list
        clients.push_back(client_socket);

        // Create a thread to handle client communication
        std::thread(handle_client, client_socket, std::ref(clients)).detach();
    }

    return 0;
```

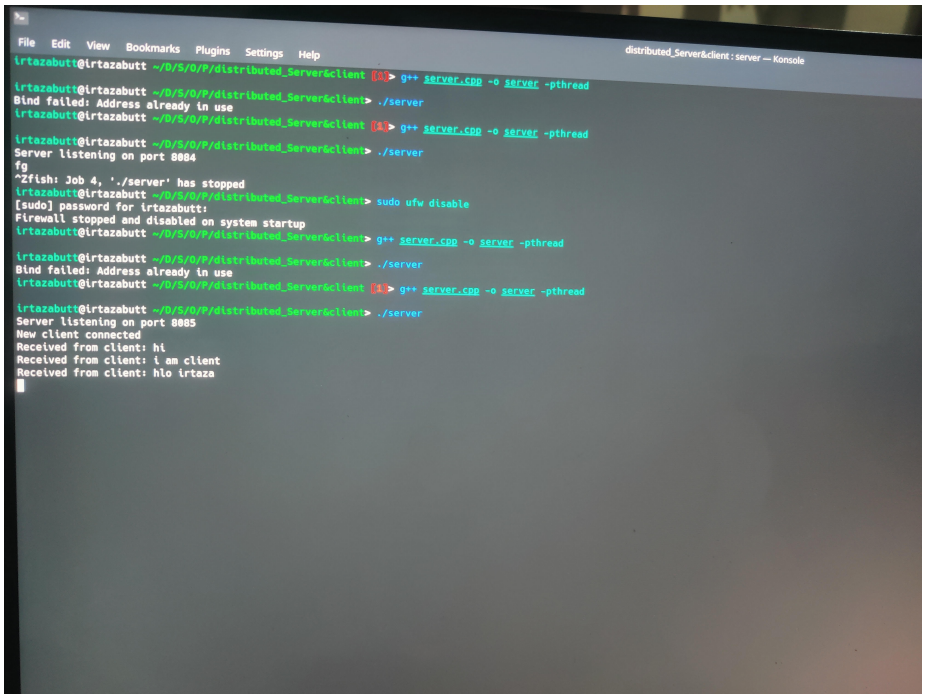

Figure 5: Server in my Laptop

```
}
```

0.2 Client Code

```
    #include <iostream>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <cstring>
#include <thread>

#define SERVER_IP "192.168.1.24"
#define PORT 8080

// Function to receive messages from the server
void receive_messages(int socket_fd) {
    char buffer[1024];
    int read_size;

    while ((read_size = read(socket_fd, buffer, sizeof(buffer))) > 0) {
        buffer[read_size] = '\0';
        std::cout << "Message from server: " << buffer << std::endl;
```

```
    }
}

// Main client function
int main() {
    int socket_fd;
    struct sockaddr_in server_addr;

    // Create the client socket
    if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

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

    // Set the server's IP address (replace with actual server IP)
    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(socket_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) <
      0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connected to the server!" << std::endl;

    // Start a thread to listen for messages from the server
    std::thread(receive_messages, socket_fd).detach();

    // Send messages to the server
    char message[1024];
    while (true) {
        std::cout << "Enter message: ";
        std::cin.getline(message, sizeof(message));

        if (send(socket_fd, message, strlen(message), 0) < 0) {
            perror("Send failed");
            break;
        }
    }

    close(socket_fd);
    return 0;
}
```

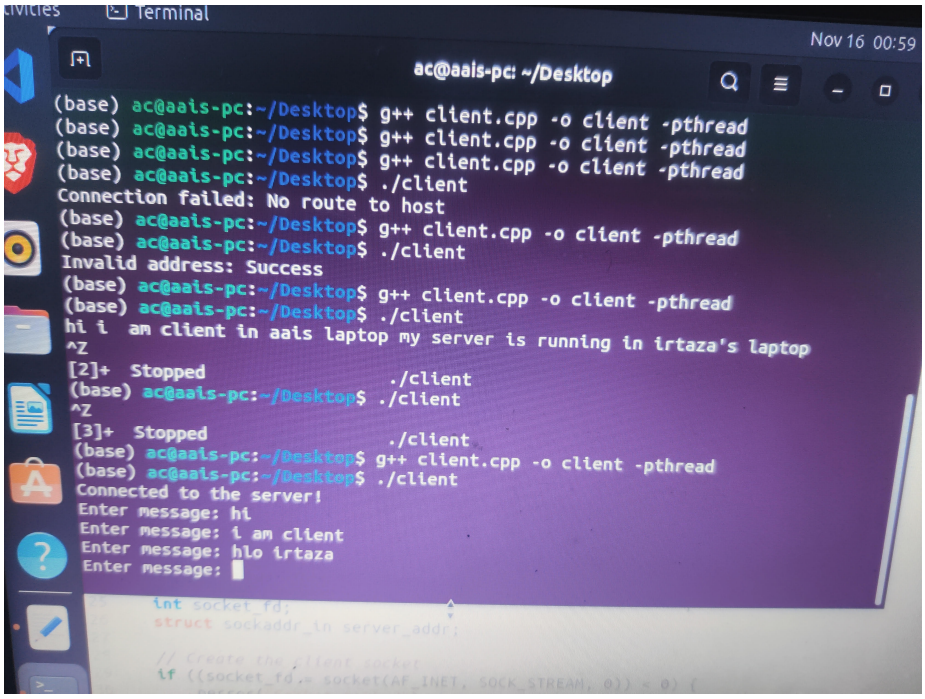

Figure 6: Client on another laptop