OPERATING SYSTEMS
SWE - 208

Oğuzhan Yıldız	B211202029
Nurullah Kapancı	B211202068
Halit Ertaş	B211202050
Yunus Emre Yiğit	B201202009

halidBinVelid/SWE208OperatingSystems(github.com)

1. Introduction

This project involves developing a CPU processor scheduler that assigns processes to two CPUs based on certain criteria. The scheduler will check resource availability before assignment, prioritize processes and use different scheduling algorithms. The main goal is to efficiently manage processes in a simulated environment with limited resources.

2. System Design

- **Overview**

  The scheduler program reads from an input file containing a list of processes with certain attributes. Processes are assigned to two CPUs according to their priority and resource requirements. The assignment is guided by rules that ensure optimal utilization of CPU and RAM.

- **Resources**

  - *CPU-1:* Dedicated to the highest priority processes (priority 0).
  - *CPU-2:* Manages user processes with priority 1 (high), 2 (medium) and 3 (low).

- **Scheduling Algorithms**

  - *CPU-1:*
    - First-Come, First-Served (FCFS)
  - *CPU-2:*
    - *High priority user processes (priority 1):* Shortest Job First (SJF)
    - *Medium priority user processes (priority 2):* Round Robin (RR) with a quantum of 8
    - *Low priority user processes (priority 3):* Round Robin (RR) with a quantum of 16

3. Process Flow

- **Resource Check**

  Before assigning processes to CPUs, the scheduler checks if sufficient resources (RAM and CPU) are available. RAM is limited to 2048 MB, with 512 MB reserved for priority 0 processes.

- **Process Assignment**

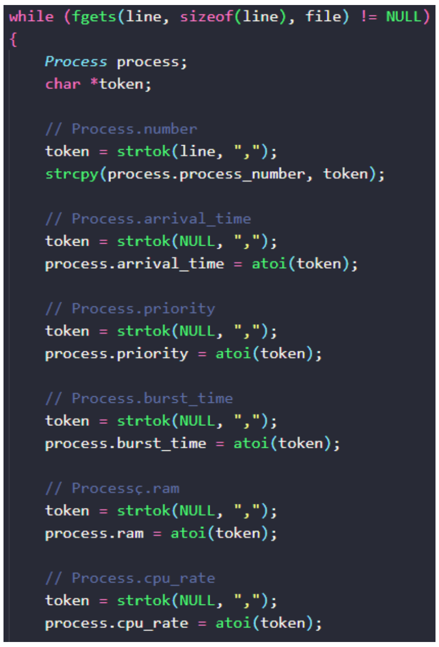
  - *Input Reading:* Extract process details by parsing the input file.
  - *Resource Allocation:* Check and allocate the required resources for each process.
  - *Queuing:* Placing processes in appropriate queues according to their priorities and scheduling algorithm.
  - *Execution:* Assigning processes to CPUs and managing them according to defined scheduling algorithms.

4. Function Descriptions

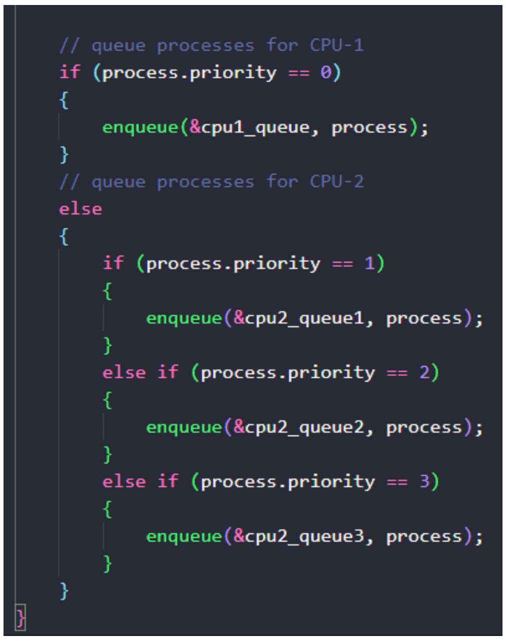
- **Main Function**

  - *Function Name:* main
  - *Purpose:* Initializes the program, reads the input file, and starts the scheduling process.
  - *Key Operations:*
    - Reading input file
    - Initializing resource structures
    - Starting the scheduling loop

The while loop below is used for reading file from input file.



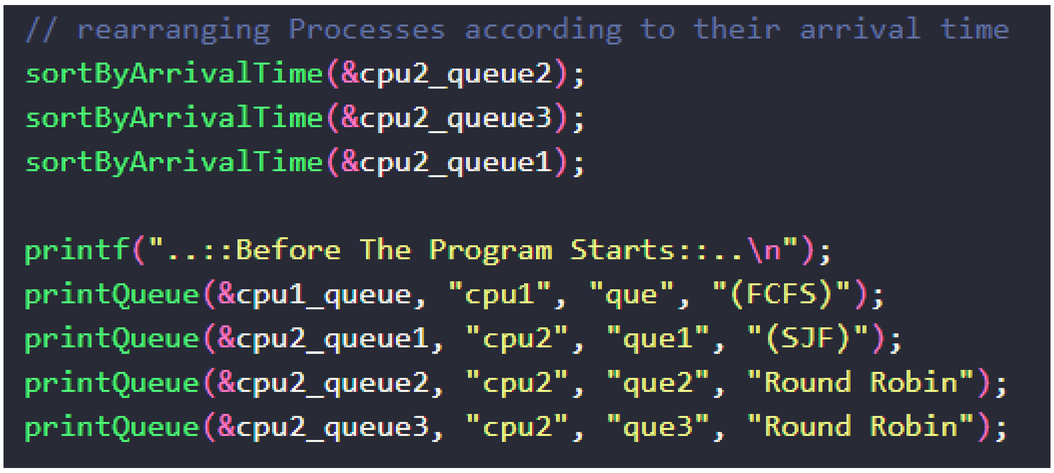
```c
while (fgets(line, sizeof(line), file) != NULL)
{
    Process process;
    char *token;

    // Process.number
    token = strtok(line, ",");
    strcpy(process.process_number, token);

    // Process.arrival_time
    token = strtok(NULL, ",");
    process.arrival_time = atoi(token);

    // Process.priority
    token = strtok(NULL, ",");
    process.priority = atoi(token);

    // Process.burst_time
    token = strtok(NULL, ",");
    process.burst_time = atoi(token);

    // Processç.ram
    token = strtok(NULL, ",");
    process.ram = atoi(token);

    // Process.cpu_rate
    token = strtok(NULL, ",");
    process.cpu_rate = atoi(token);
```



```c
    // queue processes for CPU-1
    if (process.priority == 0)
    {
        enqueue(&cpu1_queue, process);
    }
    // queue processes for CPU-2
    else
    {
        if (process.priority == 1)
        {
            enqueue(&cpu2_queue1, process);
        }
        else if (process.priority == 2)
        {
            enqueue(&cpu2_queue2, process);
        }
        else if (process.priority == 3)
        {
            enqueue(&cpu2_queue3, process);
        }
    }
}
```

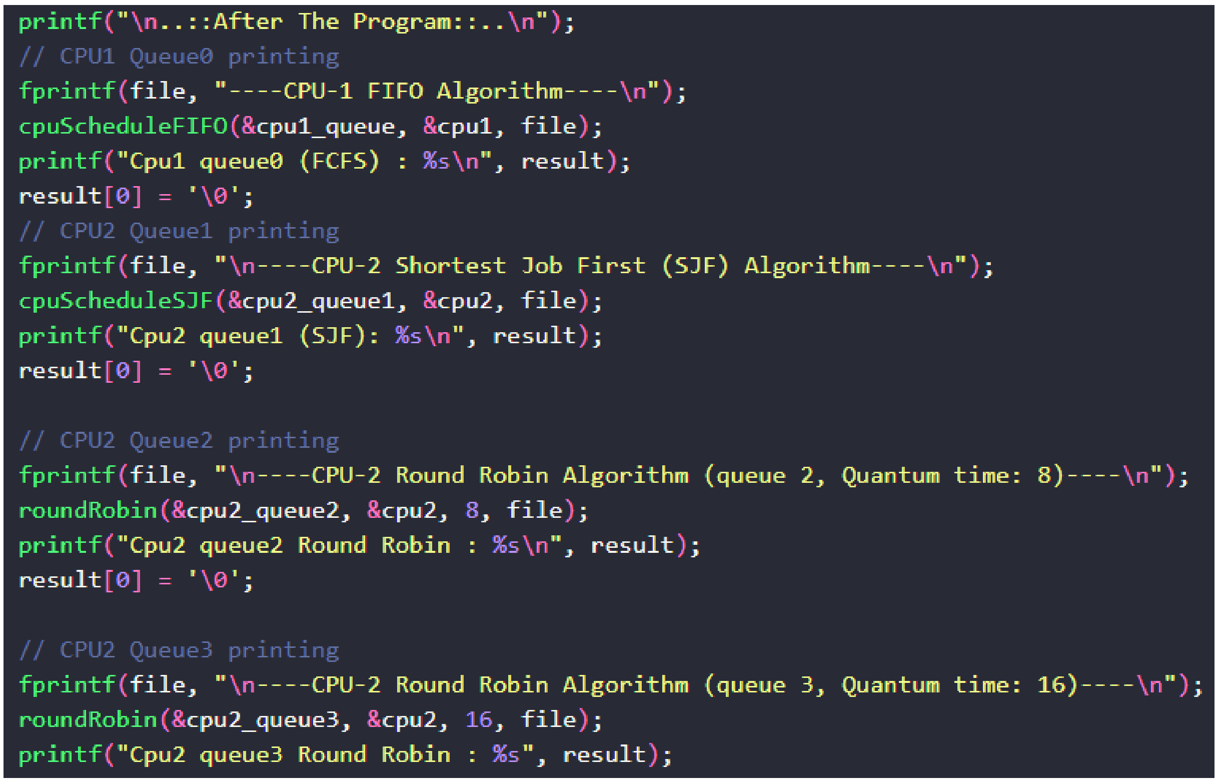
Sorting algorithms that to rearrange processes according to arrival time.

And printing queues before scheduling.



```c
// rearranging Processes according to their arrival time
sortByArrivalTime(&cpu2_queue2);
sortByArrivalTime(&cpu2_queue3);
sortByArrivalTime(&cpu2_queue1);

printf("..::Before The Program Starts::..\n");
printQueue(&cpu1_queue, "cpu1", "que", "(FCFS)");
printQueue(&cpu2_queue1, "cpu2", "que1", "(SJF)");
printQueue(&cpu2_queue2, "cpu2", "que2", "Round Robin");
printQueue(&cpu2_queue3, "cpu2", "que3", "Round Robin");
```

After the scheduling finished, printing queues according to working sequence.

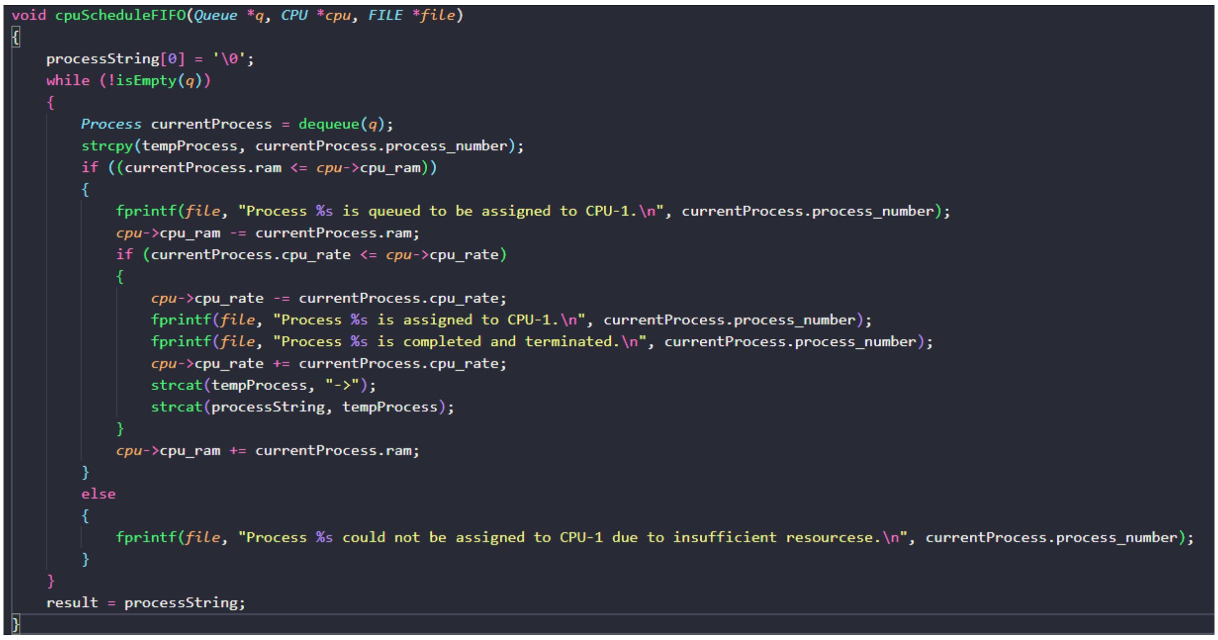


```c
printf("\n..::After The Program::..\n");
// CPU1 Queue0 printing
fprintf(file, "----CPU-1 FIFO Algorithm----\n");
cpuScheduleFIFO(&cpu1_queue, &cpu1, file);
printf("Cpu1 queue0 (FCFS) : %s\n", result);
result[0] = '\0';
// CPU2 Queue1 printing
fprintf(file, "\n----CPU-2 Shortest Job First (SJF) Algorithm----\n");
cpuScheduleSJF(&cpu2_queue1, &cpu2, file);
printf("Cpu2 queue1 (SJF): %s\n", result);
result[0] = '\0';

// CPU2 Queue2 printing
fprintf(file, "\n----CPU-2 Round Robin Algorithm (queue 2, Quantum time: 8)----\n");
roundRobin(&cpu2_queue2, &cpu2, 8, file);
printf("Cpu2 queue2 Round Robin : %s\n", result);
result[0] = '\0';

// CPU2 Queue3 printing
fprintf(file, "\n----CPU-2 Round Robin Algorithm (queue 3, Quantum time: 16)----\n");
roundRobin(&cpu2_queue3, &cpu2, 16, file);
printf("Cpu2 queue3 Round Robin : %s", result);
```

- **Scheduling Functions**

  - FCFS for CPU-1

    - *Function Name:* cpuScheduleFCFS
    - *Purpose:* Schedules processes for CPU-1 using the FCFS algorithm.

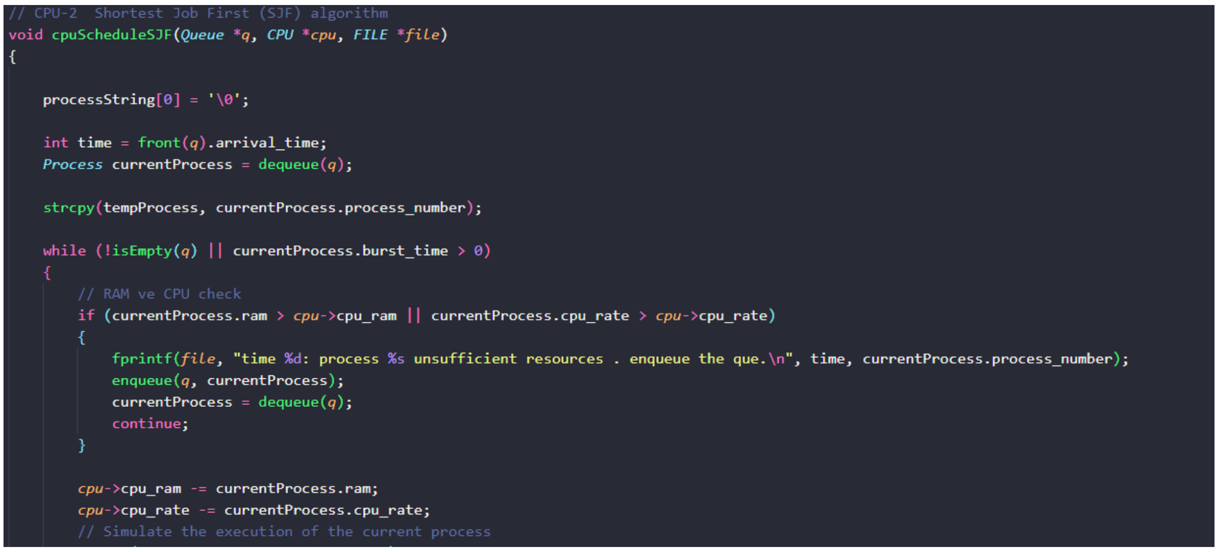


```c
void cpuScheduleFIFO(Queue *q, CPU *cpu, FILE *file)
{
    processString[0] = '\0';
    while (!isEmpty(q))
    {
        Process currentProcess = dequeue(q);
        strcpy(tempProcess, currentProcess.process_number);
        if ((currentProcess.ram <= cpu->cpu_ram))
        {
            fprintf(file, "Process %s is queued to be assigned to CPU-1.\n", currentProcess.process_number);
            cpu->cpu_ram -= currentProcess.ram;
            if (currentProcess.cpu_rate <= cpu->cpu_rate)
            {
                cpu->cpu_rate -= currentProcess.cpu_rate;
                fprintf(file, "Process %s is assigned to CPU-1.\n", currentProcess.process_number);
                fprintf(file, "Process %s is completed and terminated.\n", currentProcess.process_number);
                cpu->cpu_rate += currentProcess.cpu_rate;
                strcat(tempProcess, "->");
                strcat(processString, tempProcess);
            }
            cpu->cpu_ram += currentProcess.ram;
        }
        else
        {
            fprintf(file, "Process %s could not be assigned to CPU-1 due to insufficient resourcese.\n", currentProcess.process_number);
        }
    }
    result = processString;
}
```

➢ SJF for High Priority Queue

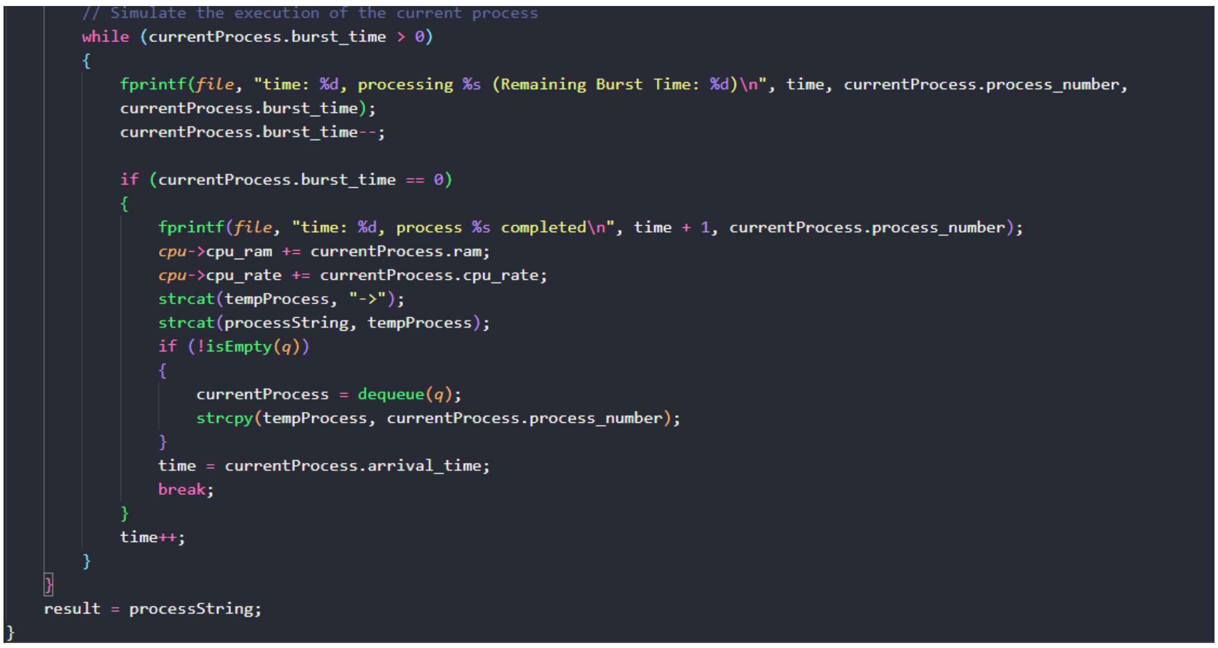
❖ *Function Name:* schedule_sjf

❖ *Purpose:* Schedules high priority user processes using the SJF algorithm.



```
// CPU-2  Shortest Job First (SJF) algorithm
void cpuScheduleSJF(Queue *q, CPU *cpu, FILE *file)
{

    processString[0] = '\0';

    int time = front(q).arrival_time;
    Process currentProcess = dequeue(q);

    strcpy(tempProcess, currentProcess.process_number);

    while (!isEmpty(q) || currentProcess.burst_time > 0)
    {
        // RAM ve CPU check
        if (currentProcess.ram > cpu->cpu_ram || currentProcess.cpu_rate > cpu->cpu_rate)
        {
            fprintf(file, "time %d: process %s unsufficient resources . enqueue the que.\n", time, currentProcess.process_number);
            enqueue(q, currentProcess);
            currentProcess = dequeue(q);
            continue;
        }

        cpu->cpu_ram -= currentProcess.ram;
        cpu->cpu_rate -= currentProcess.cpu_rate;
        // Simulate the execution of the current process
```

❖

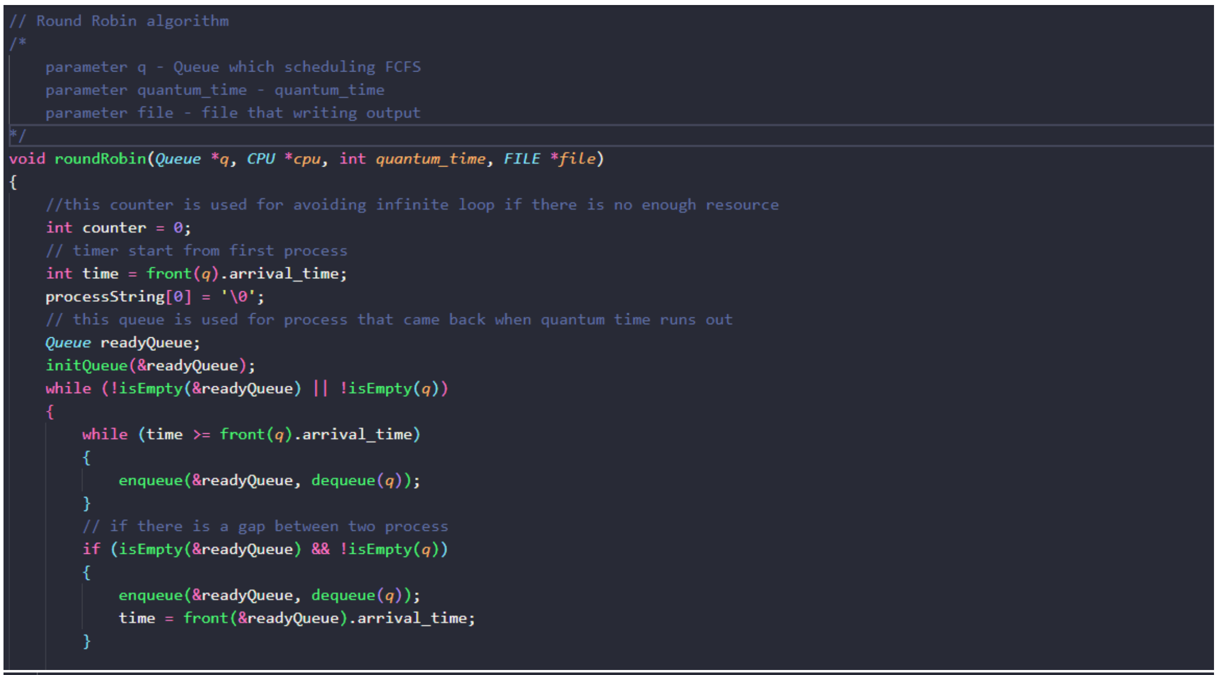


```
        // Simulate the execution of the current process
        while (currentProcess.burst_time > 0)
        {
            fprintf(file, "time: %d, processing %s (Remaining Burst Time: %d)\n", time, currentProcess.process_number,
            currentProcess.burst_time);
            currentProcess.burst_time--;

            if (currentProcess.burst_time == 0)
            {
                fprintf(file, "time: %d, process %s completed\n", time + 1, currentProcess.process_number);
                cpu->cpu_ram += currentProcess.ram;
                cpu->cpu_rate += currentProcess.cpu_rate;
                strcat(tempProcess, "->");
                strcat(processString, tempProcess);
                if (!isEmpty(q))
                {
                    currentProcess = dequeue(q);
                    strcpy(tempProcess, currentProcess.process_number);
                }
                time = currentProcess.arrival_time;
                break;
            }
            time++;
        }
    }
    result = processString;
}
```

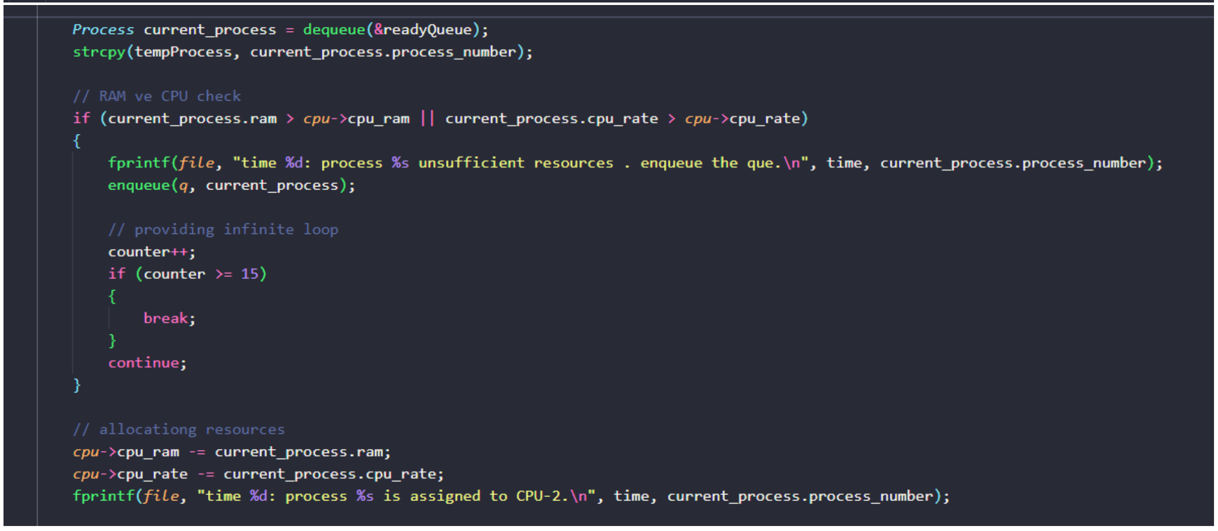
➢ RR for Medium Priority Queue

❖ *Function Name:* schedule_rr_medium

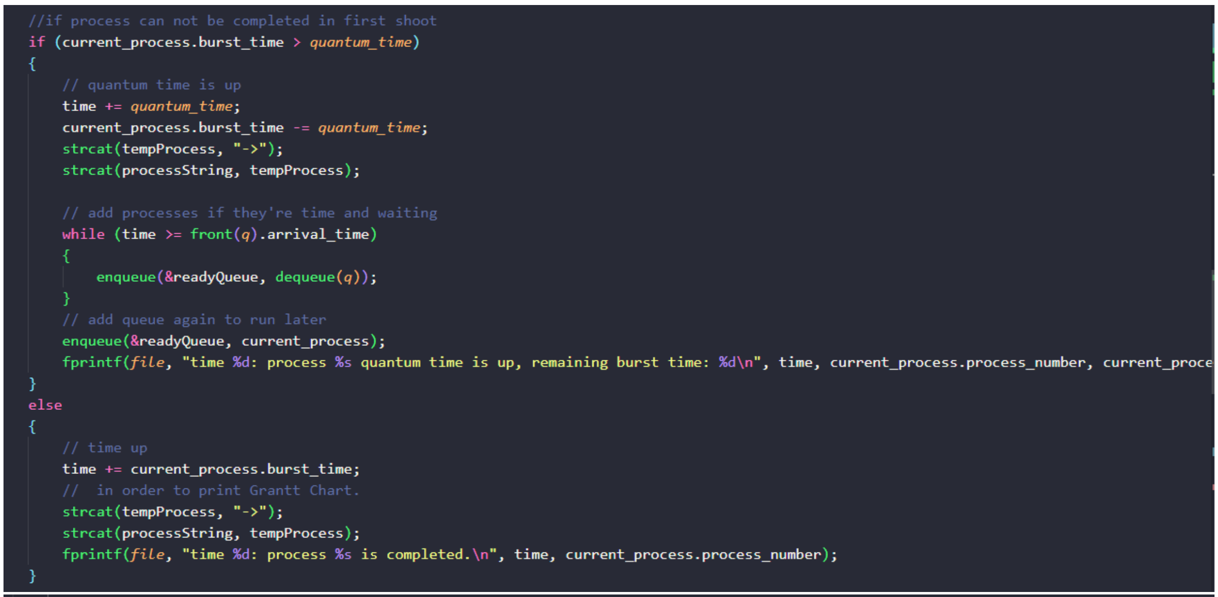
❖ *Purpose:* Schedules medium priority user processes using RR with a quantum of 8.



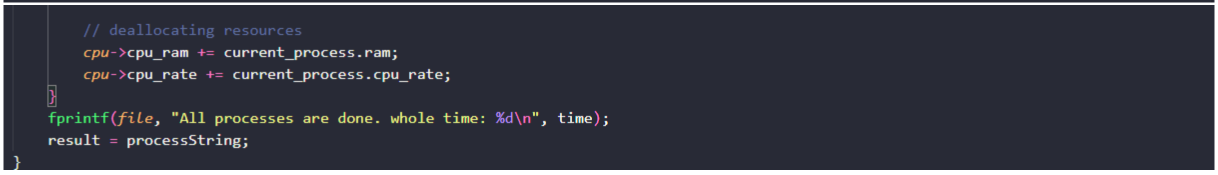
```c
// Round Robin algorithm
/*
    parameter q - Queue which scheduling FCFS
    parameter quantum_time - quantum_time
    parameter file - file that writing output
*/
void roundRobin(Queue *q, CPU *cpu, int quantum_time, FILE *file)
{
    //this counter is used for avoiding infinite loop if there is no enough resource
    int counter = 0;
    // timer start from first process
    int time = front(q).arrival_time;
    processString[0] = '\0';
    // this queue is used for process that came back when quantum time runs out
    Queue readyQueue;
    initQueue(&readyQueue);
    while (!isEmpty(&readyQueue) || !isEmpty(q))
    {
        while (time >= front(q).arrival_time)
        {
            enqueue(&readyQueue, dequeue(q));
        }
        // if there is a gap between two process
        if (isEmpty(&readyQueue) && !isEmpty(q))
        {
            enqueue(&readyQueue, dequeue(q));
            time = front(&readyQueue).arrival_time;
        }
```



```c
        Process current_process = dequeue(&readyQueue);
        strcpy(tempProcess, current_process.process_number);

        // RAM ve CPU check
        if (current_process.ram > cpu->cpu_ram || current_process.cpu_rate > cpu->cpu_rate)
        {
            fprintf(file, "time %d: process %s unsufficient resources . enqueue the que.\n", time, current_process.process_number);
            enqueue(q, current_process);

            // providing infinite loop
            counter++;
            if (counter >= 15)
            {
                break;
            }
            continue;
        }

        // allocationg resources
        cpu->cpu_ram -= current_process.ram;
        cpu->cpu_rate -= current_process.cpu_rate;
        fprintf(file, "time %d: process %s is assigned to CPU-2.\n", time, current_process.process_number);
```



```
//if process can not be completed in first shoot
if (current_process.burst_time > quantum_time)
{
    // quantum time is up
    time += quantum_time;
    current_process.burst_time -= quantum_time;
    strcat(tempProcess, "->");
    strcat(processString, tempProcess);

    // add processes if they're time and waiting
    while (time >= front(q).arrival_time)
    {
        enqueue(&readyQueue, dequeue(q));
    }
    // add queue again to run later
    enqueue(&readyQueue, current_process);
    fprintf(file, "time %d: process %s quantum time is up, remaining burst time: %d\n", time, current_process.process_number, current_proce
}
else
{
    // time up
    time += current_process.burst_time;
    //  in order to print Grantt Chart.
    strcat(tempProcess, "->");
    strcat(processString, tempProcess);
    fprintf(file, "time %d: process %s is completed.\n", time, current_process.process_number);
}
```



```
        // deallocating resources
        cpu->cpu_ram += current_process.ram;
        cpu->cpu_rate += current_process.cpu_rate;
    }
    fprintf(file, "All processes are done. whole time: %d\n", time);
    result = processString;
}
```

5. Output

The output of the scheduler is written to output.txt, detailing the sequence of operations for each process. The log includes:

- Queueing of processes
- Assignment to CPUs
- Completion and termination of processes
- **Sample Output:**

```
----CPU-1 FIFO Algorithm----
Process P2 is queued to be assigned to CPU-1.
Process P2 is assigned to CPU-1.
Process P2 is completed and terminated.
Process P4 is queued to be assigned to CPU-1.
Process P4 is assigned to CPU-1.
Process P4 is completed and terminated.
Process P7 is queued to be assigned to CPU-1.
Process P7 is assigned to CPU-1.
Process P7 is completed and terminated.
Process P8 is queued to be assigned to CPU-1.
Process P8 is assigned to CPU-1.
Process P8 is completed and terminated.
Process P9 is queued to be assigned to CPU-1.
Process P9 is assigned to CPU-1.
Process P9 is completed and terminated.
Process P11 is queued to be assigned to CPU-1.
Process P11 is assigned to CPU-1.
Process P11 is completed and terminated.
Process P17 is queued to be assigned to CPU-1.
Process P17 is assigned to CPU-1.
Process P17 is completed and terminated.
Process P18 is queued to be assigned to CPU-1.
Process P18 is assigned to CPU-1.
Process P18 is completed and terminated.
Process P20 is queued to be assigned to CPU-1.
Process P20 is assigned to CPU-1.
Process P20 is completed and terminated.

----CPU-2 Shortest Job First (SJF) Algorithm----
time: 0, processing P1 (Remaining Burst Time: 2)
time: 1, processing P1 (Remaining Burst Time: 1)
time: 2, process P1 completed
time: 6, processing P14 (Remaining Burst Time: 2)
time: 7, processing P14 (Remaining Burst Time: 1)
time: 8, process P14 completed
```

```
time: 8, processing P15 (Remaining Burst Time: 4)
time: 9, processing P15 (Remaining Burst Time: 3)
time: 10, processing P15 (Remaining Burst Time: 2)
time: 11, processing P15 (Remaining Burst Time: 1)
time: 12, process P15 completed
time: 24, processing P25 (Remaining Burst Time: 2)
time: 25, processing P25 (Remaining Burst Time: 1)
time: 26, process P25 completed

----CPU-2 Round Robin Algorithm (queue 2, Quantum time: 8)----
time 1: process P5 is assigned to CPU-2.
time 3: process P5 is completed.
time 3: process P6 is assigned to CPU-2.
time 6: process P6 is completed.
time 6: process P10 is assigned to CPU-2.
time 14: process P10 quantum time is up, remaining burst time: 32
time 14: process P19 is assigned to CPU-2.
time 22: process P19 quantum time is up, remaining burst time: 2
time 22: process P10 is assigned to CPU-2.
time 30: process P10 quantum time is up, remaining burst time: 24
time 30: process P23 is assigned to CPU-2.
time 38: process P23 quantum time is up, remaining burst time: 25
time 38: process P19 is assigned to CPU-2.
time 40: process P19 is completed.
time 40: process P10 is assigned to CPU-2.
time 48: process P10 quantum time is up, remaining burst time: 16
time 48: process P23 is assigned to CPU-2.
time 56: process P23 quantum time is up, remaining burst time: 17
time 56: process P10 is assigned to CPU-2.
time 64: process P10 quantum time is up, remaining burst time: 8
time 64: process P23 is assigned to CPU-2.
time 72: process P23 quantum time is up, remaining burst time: 9
```

time 72: process P10 is assigned to CPU-2.

time 80: process P10 is completed.

time 80: process P23 is assigned to CPU-2.

time 88: process P23 quantum time is up, remaining burst time: 1

time 88: process P23 is assigned to CPU-2.

time 89: process P23 is completed.

All processes are done. whole time: 89

----CPU-2 Round Robin Algorithm (queue 3, Quantum time: 16)----

time 1: process P3 is assigned to CPU-2.

time 3: process P3 is completed.

time 5: process P12 is assigned to CPU-2.

time 7: process P12 is completed.

time 7: process P13 is assigned to CPU-2.

time 9: process P13 is completed.

time 9: process P16 is assigned to CPU-2.

time 13: process P16 is completed.

time 16: process P21 is assigned to CPU-2.

time 19: process P21 is completed.

time 19: process P22 is assigned to CPU-2.

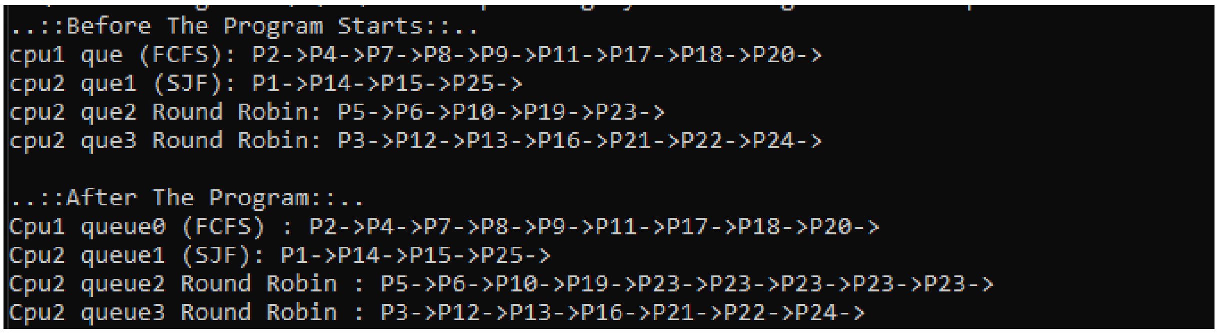
time 21: process P22 is completed.

time 23: process P24 is assigned to CPU-2.

time 25: process P24 is completed.

All processes are done. whole time: 25

CMD Output:



```
..::Before The Program Starts::..
cpu1 que (FCFS): P2->P4->P7->P8->P9->P11->P17->P18->P20->
cpu2 que1 (SJF): P1->P14->P15->P25->
cpu2 que2 Round Robin: P5->P6->P10->P19->P23->
cpu2 que3 Round Robin: P3->P12->P13->P16->P21->P22->P24->

..::After The Program::..
Cpu1 queue0 (FCFS) : P2->P4->P7->P8->P9->P11->P17->P18->P20->
Cpu2 queue1 (SJF): P1->P14->P15->P25->
Cpu2 queue2 Round Robin : P5->P6->P10->P19->P23->P23->P23->P23->P23->
Cpu2 queue3 Round Robin : P3->P12->P13->P16->P21->P22->P24->
```

6. Conclusion

The CPU processor scheduler efficiently assigns processes to CPUs according to priority and resource requirements. The use of different scheduling algorithms ensures optimal CPU utilization and process management. The project illustrates key concepts in operating system design such as process scheduling, resource allocation and queue management.