OS Project

By Devashish Tushar (2022uec1487).

This C code demonstrates a basic implementation of the Round Robin scheduling algorithm, which is a preemptive scheduling algorithm commonly used in operating systems to schedule tasks or processes on a CPU. In Round Robin scheduling, each process is assigned a fixed time slice (quantum), and the scheduler switches between processes in a circular queue, giving each process a chance to execute for a fixed time before moving to the next one. This continues until all processes are completed.

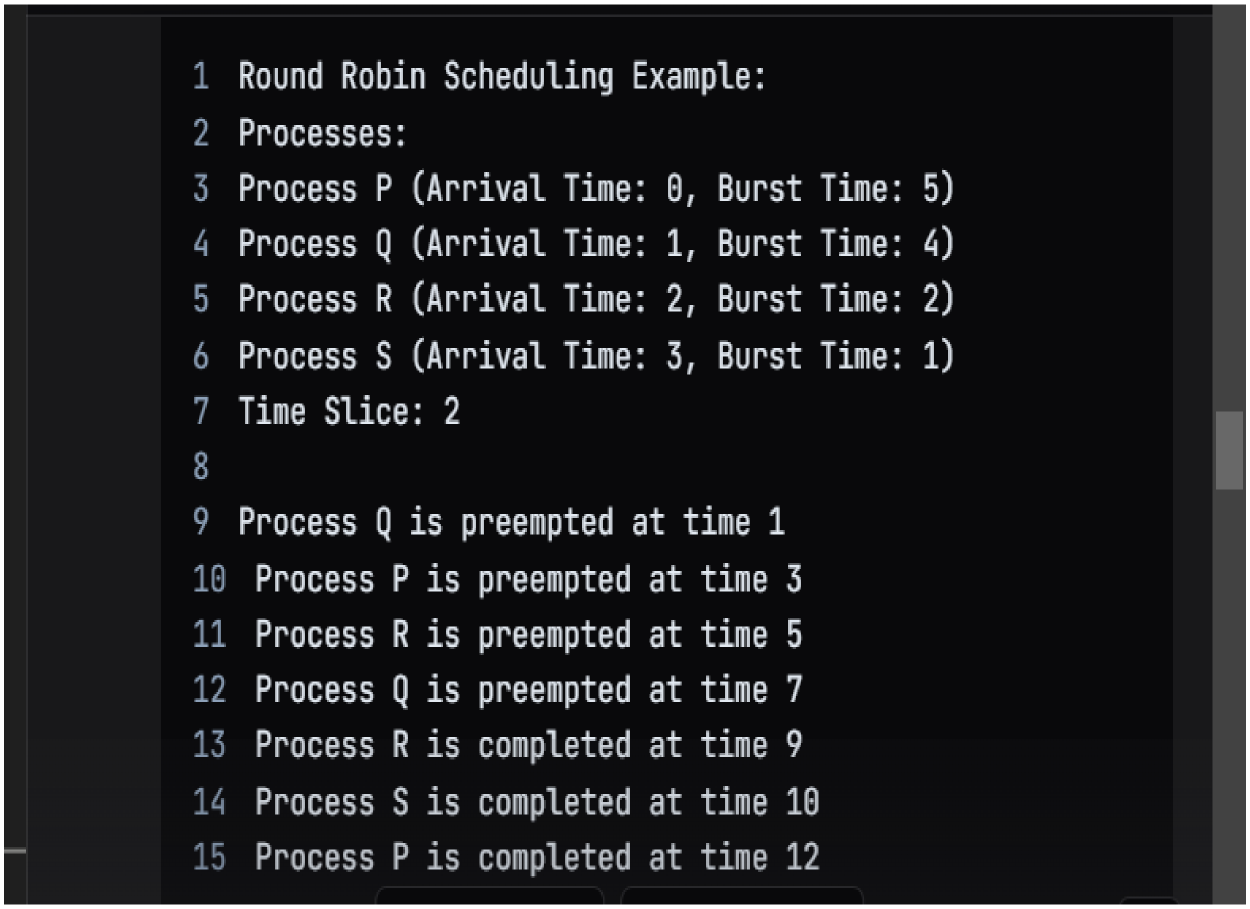
Here's an explanation of the key components of the code:

- **Data Structure**:
- The code defines a **`struct Process`** to represent a process with the following attributes:
- **`name`**: A character representing the name or identifier of the process.
- **`arrival_time`**: The time at which the process arrives and becomes ready for execution.
- **`burst_time`**: The total time required for the process to complete its execution.
- **`remaining_time`**: The remaining time to complete the process (initialized to the burst time initially).
- **round_robin_scheduler** Function:
- This function takes an array of **`struct Process`**, the number of processes (**`num_processes`**), and the time slice (**`time_slice`**) as parameters.
- It maintains a circular queue (**`queue`**) to store processes that are ready to be scheduled.
- It initializes variables for the current time, the number of completed processes, and pointers to the front and rear of the queue.
- It loops until all processes are completed:
- Processes that have arrived and still have remaining time are added to the queue.
- If the queue is empty, the current time is incremented (no processes are ready to run).
- If the queue is not empty, the first process in the queue is taken and executed:
- If the remaining time of the process is less than or equal to the time slice, the process is completed, and its remaining time becomes zero.
- If the remaining time is greater than the time slice, the process is preempted (its remaining time is reduced by the time slice).

- The process's completion or preemption is printed with the process name and the current time.
- The number of completed processes is incremented.
- **main** Function:
- In the `main` function, an array of `struct Process` is defined to represent the set of processes to be scheduled.
- The number of processes and the time slice are initialized.
- The details of the processes and time slice are printed.
- The `round_robin_scheduler` function is called with the process array, number of processes, and time slice.
- The program returns 0 to indicate successful execution.
- **Example Data**:
- In the provided example, four processes (P, Q, R, and S) with their arrival times and burst times are defined.
- The time slice is set to 2.
- **Output**:
- The code simulates the execution of these processes using the Round Robin scheduling algorithm and prints the completion or preemption of processes along with the current time.

The output will demonstrate how processes are scheduled and executed with the specified time slice until all processes are completed. The code gives you a basic idea of how Round Robin scheduling works in a simplified context.

```c
#include <stdio.h>

#include <stdlib.h>



struct Process {

    char name;

    int arrival_time;

    int burst_time;

    int remaining_time;

};



void round_robin_scheduler(struct Process processes[], int num_processes, int
time_slice) {

    struct Process queue[num_processes];

    int front = 0, rear = 0;



    for (int i = 0; i < num_processes; i++) {

        processes[i].remaining_time = processes[i].burst_time;

    }



    int current_time = 0;

    int completed_processes = 0;



    while (completed_processes < num_processes) {

        for (int i = 0; i < num_processes; i++) {

            if (processes[i].arrival_time <= current_time &&
processes[i].remaining_time > 0) {

                queue[rear] = processes[i];
                rear = (rear + 1) % num_processes;
            }
        }


        if (front == rear) {
            current_time++;
        } else {
            struct Process current_process = queue[front];
            front = (front + 1) % num_processes;


            if (current_process.remaining_time <= time_slice) {
                current_time += current_process.remaining_time;
                current_process.remaining_time = 0;
                printf("Process %c completed at time %d\n", current_process.name,
current_time);
                completed_processes++;
            } else {
                current_time += time_slice;
                current_process.remaining_time -= time_slice;
                printf("Process %c is preempted at time %d\n",
current_process.name, current_time);
            }
        }
    }
}


int main() {
    struct Process processes[] = {
        {'P', 0, 5, 0},

        {'Q', 1, 4, 0},

        {'R', 2, 2, 0},

        {'S', 3, 1, 0}

    };



    int num_processes = sizeof(processes) / sizeof(processes[0]);

    int time_slice = 2;



    printf("Round Robin Scheduling Example:\n");

    printf("Processes:\n");

    for (int i = 0; i < num_processes; i++) {

        printf("Process %c (Arrival Time: %d, Burst Time: %d)\n",
processes[i].name, processes[i].arrival_time, processes[i].burst_time);

    }

    printf("Time Slice: %d\n", time_slice);



    round_robin_scheduler(processes, num_processes, time_slice);



    return 0;

}
```



```
Round Robin Scheduling Example:
Processes:
Process P (Arrival Time: 0, Burst Time: 5)
Process Q (Arrival Time: 1, Burst Time: 4)
Process R (Arrival Time: 2, Burst Time: 2)
Process S (Arrival Time: 3, Burst Time: 1)
Time Slice: 2

Process Q is preempted at time 1
 Process P is preempted at time 3
 Process R is preempted at time 5
 Process Q is preempted at time 7
 Process R is completed at time 9
 Process S is completed at time 10
 Process P is completed at time 12
```