

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari  
-2022UEC1616  
-E2

## Algorithm Explanation:

Certainly! Let's break down how the Round Robin scheduling algorithm with a dynamic time quantum works in steps:

### 1. Initialize variables and data structures:

- Initialize necessary data structures, including lists to track remaining time, turnaround time, and waiting time for each process. Set the initial time to 0.
- Load the list of processes with their names and burst times.

### 2. Determine the time quantum dynamically:

- For each iteration of the main loop, the algorithm dynamically determines the time quantum for the currently running process. The quantum is calculated as half of the remaining time of the current process. If the remaining time is less than 2, the quantum is set to 1 to ensure that no process is scheduled for longer than its remaining time.

### 3. Execute the Round Robin algorithm:

- The algorithm enters a loop that continues until all processes are completed.
- For each process with remaining time (i.e., `remaining_time[i] > 0``), the algorithm schedules it for the determined time quantum.
- The current time is incremented by the time quantum, representing the execution of the process for that duration.
- The remaining time of the process is reduced by the time quantum.
- If a process completes (i.e., `remaining_time[i] == 0``), the algorithm updates its waiting time and turnaround time.
- The waiting time for the process is calculated as the current time minus the original burst time of the process.
- The turnaround time is calculated as the current time.

### 4. Calculate average waiting time and average turnaround time:

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari

-2022UEC1616

-E2

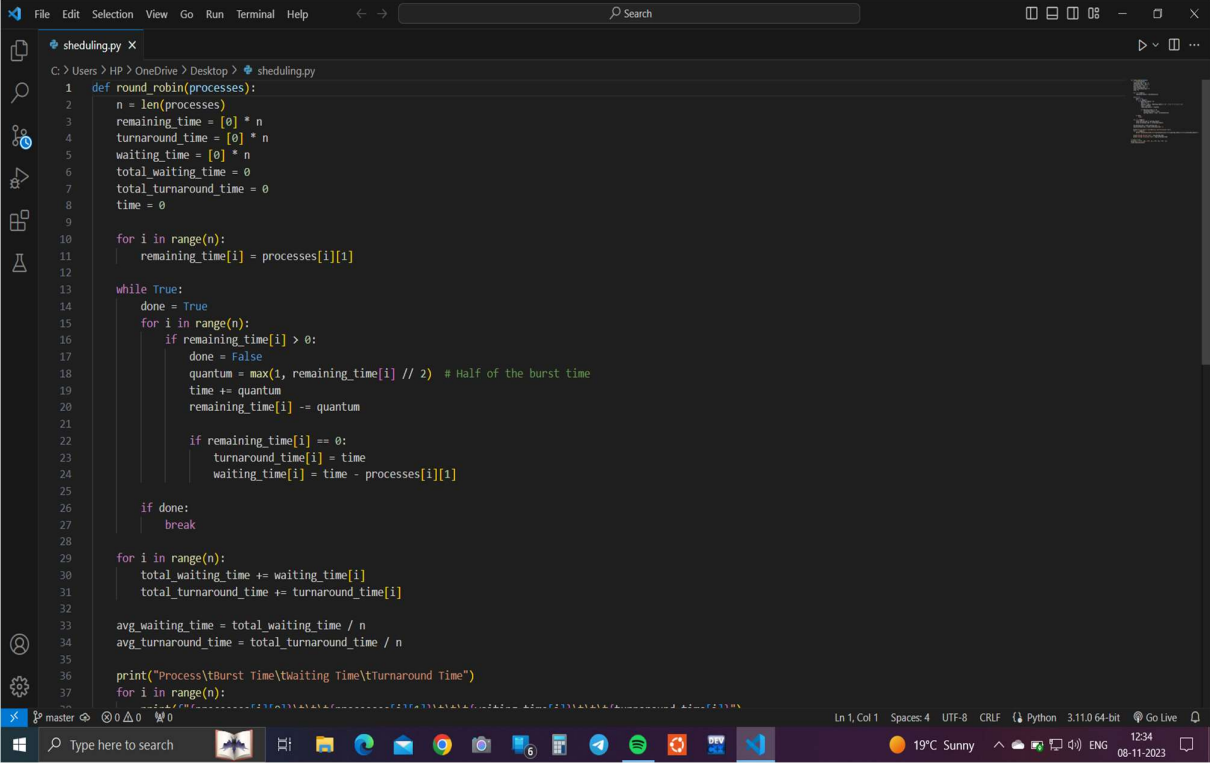
- After the Round Robin algorithm execution, the algorithm calculates the average waiting time and average turnaround time for all processes.

## 5. Print the results:

- The algorithm prints the calculated average waiting time and average turnaround time for the given set of processes, which represent the overall performance of the scheduling algorithm.

In summary, this modified Round Robin scheduling algorithm dynamically adjusts the time quantum for each process based on its remaining time, ensuring fair scheduling and responsiveness to processes with shorter burst times. This approach aims to optimize the utilization of the CPU while avoiding unnecessary overhead when dealing with very short tasks.

## Code:



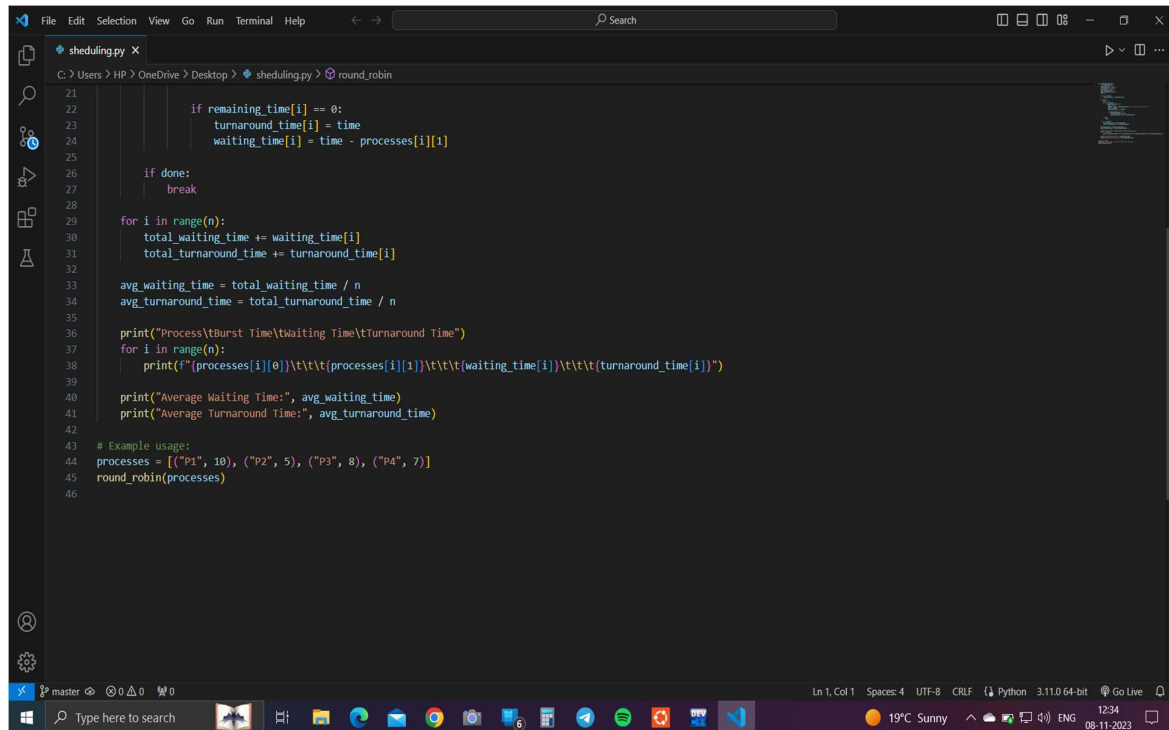
```
1 def round_robin(processes):
2     n = len(processes)
3     remaining_time = [0] * n
4     turnaround_time = [0] * n
5     waiting_time = [0] * n
6     total_waiting_time = 0
7     total_turnaround_time = 0
8     time = 0
9
10    for i in range(n):
11        remaining_time[i] = processes[i][1]
12
13    while True:
14        done = True
15        for i in range(n):
16            if remaining_time[i] > 0:
17                done = False
18                quantum = max(1, remaining_time[i] // 2) # Half of the burst time
19                time += quantum
20                remaining_time[i] -= quantum
21
22            if remaining_time[i] == 0:
23                turnaround_time[i] = time
24                waiting_time[i] = time - processes[i][1]
25
26        if done:
27            break
28
29    for i in range(n):
30        total_waiting_time += waiting_time[i]
31        total_turnaround_time += turnaround_time[i]
32
33    avg_waiting_time = total_waiting_time / n
34    avg_turnaround_time = total_turnaround_time / n
35
36    print("Process\tBurst Time\tWaiting Time\tTurnaround Time")
37    for i in range(n):
38        print(f"{processes[i][0]}\t{processes[i][1]}\t{waiting_time[i]}\t{turnaround_time[i]}")
```

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari

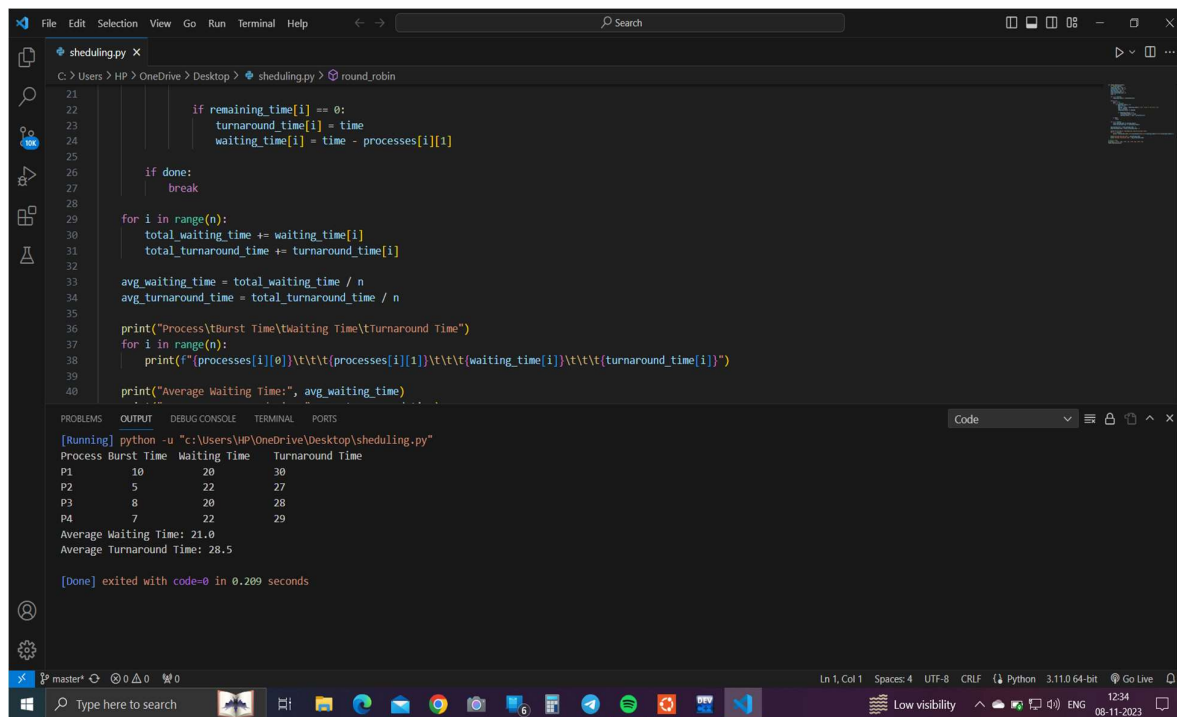
-2022UEC1616

-E2



```
21
22     if remaining_time[i] == 0:
23         turnaround_time[i] = time
24         waiting_time[i] = time - processes[i][1]
25
26     if done:
27         break
28
29     for i in range(n):
30         total_waiting_time += waiting_time[i]
31         total_turnaround_time += turnaround_time[i]
32
33     avg_waiting_time = total_waiting_time / n
34     avg_turnaround_time = total_turnaround_time / n
35
36     print("Process\tBurst Time\tWaiting Time\tTurnaround Time")
37     for i in range(n):
38         print(f"{processes[i][0]}\t\t\t{processes[i][1]}\t\t\t{waiting_time[i]}\t\t\t{turnaround_time[i]}")
39
40     print("Average Waiting Time:", avg_waiting_time)
41     print("Average Turnaround Time:", avg_turnaround_time)
42
43 # Example usage:
44 processes = [("P1", 10), ("P2", 5), ("P3", 8), ("P4", 7)]
45 round_robin(processes)
46
```

## Output:



```
Process Burst Time Waiting Time Turnaround Time
P1      10      20      30
P2       5       22      27
P3       8       20      28
P4       7       22      29
Average Waiting Time: 21.0
Average Turnaround Time: 28.5
```

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari  
-2022UEC1616  
-E2

## Code Explanation:

Certainly! I'll explain the Round Robin scheduling algorithm with a time quantum equal to half of the burst time of the current running process step by step:

### 1. Initialize variables and data structures:

- ``processes``: This is a list of processes, where each process is represented as a tuple containing its name (e.g., "P1") and burst time (the time required for the process to complete).
- ``remaining_time``: This list is used to keep track of the remaining time for each process. It is initially set to the burst time of each process.
- ``turnaround_time``: This list will store the turnaround time for each process (the time from arrival to completion).
- ``waiting_time``: This list will store the waiting time for each process (the time a process spends waiting in the ready queue).
- ``total_waiting_time`` and ``total_turnaround_time``: These variables will be used to calculate the total waiting time and total turnaround time.
- ``time``: This variable represents the current time.

### 2. Determine the time quantum dynamically:

- Within the while loop, for each iteration, the code determines the time quantum for the currently running process dynamically. The quantum is set to half of the remaining time of the current process. If the remaining time is less than 2, the quantum is set to 1 to ensure that no process is scheduled for longer than its remaining time.

### 3. Execute the Round Robin algorithm:

- The loop iterates until all processes are completed (i.e., ``done`` is True).
- For each process with remaining time, the code schedules it for the determined time quantum.
- The process's remaining time is reduced by the quantum.
- If a process completes (i.e., its remaining time becomes 0), the code updates its waiting time and turnaround time.

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari  
-2022UEC1616  
-E2

4. Calculate average waiting time and average turnaround time:

- After the Round Robin algorithm execution, the code calculates the average waiting time and average turnaround time for all processes.

5. Print the results:

- Finally, the code prints the calculated average waiting time and average turnaround time for the given set of processes.

The modified Round Robin algorithm with dynamic time quantum helps ensure that processes with shorter burst times are scheduled more frequently, which can improve the overall performance of the scheduling algorithm.

## Advantages and Disadvantages:

Dynamic time quantum in Round Robin (RR) scheduling is an approach where the time quantum for each process can change during its execution. This can offer some advantages and disadvantages:

### Advantages:

1. **Better Responsiveness**: The time quantum can be adjusted dynamically based on the burst time or priority of processes. This means that high-priority or short-duration tasks can be given more CPU time, improving system responsiveness.

2. **Optimal for Mix of Processes**: Dynamic time quantum RR is well-suited for systems with a mix of short and long jobs. It allows short jobs to complete quickly while still providing a fair share of CPU time to longer jobs.

# Operating System Mini Project (Scheduling Algorithm)

-Tanmay Maheshwari

-2022UEC1616

-E2

3. **Efficiency**: Shorter time quantum for CPU-bound tasks and longer time quantum for I/O-bound tasks can lead to efficient resource utilization, as it minimizes overhead associated with context switching.

4. **Adaptability**: The algorithm adapts to the nature of processes. When a process becomes I/O-bound, it can receive a longer time quantum to reduce context switching, and when it's CPU-bound, it can receive shorter time quanta to ensure fairness.

## Disadvantages:

1. **Complexity**: Implementing dynamic time quantum requires a more complex scheduler. The algorithm must monitor process behavior and make adjustments, which can add overhead and complexity to the scheduler.

2. **Starvation**: In systems with many high-priority or short-duration tasks, long-duration tasks might not receive adequate CPU time, leading to potential starvation. This can be addressed with proper priority handling but adds complexity.

3. **Difficulty in Predicting Behavior**: Predicting the optimal time quantum for a process can be challenging. Incorrect decisions may lead to inefficient resource usage, making it necessary to fine-tune the algorithm.

4. **Potential for Abusive Behavior**: While dynamic time quantum aims to optimize fairness and responsiveness, it can also be exploited by malicious or poorly-behaved processes, causing system instability.

In summary, dynamic time quantum RR can be advantageous in situations where a mix of processes with varying burst times and priorities are present, leading to better system responsiveness and efficiency. However, it can also introduce complexity and potential issues such as starvation and unpredictability in certain scenarios. The choice of whether to use a dynamic time quantum should be based on the specific requirements and characteristics of the system in question.