OS Assignment 3

SCHEDULING ALGORITHMS

1. FCFS (First come, first server)

```python
# FCFS (First Come, First Server): Processes are executed in the order they arrive.

# Code:

def fcfs(processes, arrival_times, burst_times):
    n = len(processes)
    start_times = [0] * n
    completion_times = [0] * n
    waiting_times = [0] * n
    turnaround_times = [0] * n

    current_time = 0
    for i in range(n):
        if current_time < arrival_times[i]:
            current_time = arrival_times[i]
        start_times[i] = current_time
        completion_times[i] = current_time + burst_times[i]
        turnaround_times[i] = completion_times[i] - arrival_times[i]
        waiting_times[i] = turnaround_times[i] - burst_times[i]
        current_time += burst_times[i]

    print("Process\tArrival\tBurst\tStart\tCompletion\tWaiting\tTurnaround")
    for i in range(n):
        print(f"{processes[i]}\t{arrival_times[i]}\t{burst_times[i]}\t{start_times[i]}\t{completion_times[i]}\t\t{wait

processes = ['P1', 'P2', 'P3']
arrival_times = [0, 2, 4]
burst_times = [5, 3, 1]
fcfs(processes, arrival_times, burst_times)
```

```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ nano fcfs.py

┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python fcfs.py
Process Arrival Burst   Start   Completion      Waiting Turnaround
P1      0       5       0       5               0       5
P2      2       3       5       8               3       6
P3      4       1       8       9               4       5
```

2. SJF (Shortest Job First):

```
# SJF: Processes with the shortest burst time are executed first.

# CODE:

def sjf(processes, arrival_times, burst_times):
    n = len(processes)
    completed = [False] * n
    current_time = 0
    completed_count = 0

    start_times = [0] * n
    completion_times = [0] * n
    waiting_times = [0] * n
    turnaround_times = [0] * n

    while completed_count < n:
        # Select process with minimum burst time among those that have arrived and not completed
        idx = -1
        min_burst = float('inf')
        for i in range(n):
            if arrival_times[i] <= current_time and not completed[i]:
                if burst_times[i] < min_burst:
                    min_burst = burst_times[i]
                    idx = i

        if idx == -1:
            current_time += 1
        else:
            start_times[idx] = current_time
```

```
        else:
            start_times[idx] = current_time
            completion_times[idx] = current_time + burst_times[idx]
            turnaround_times[idx] = completion_times[idx] - arrival_times[idx]
            waiting_times[idx] = turnaround_times[idx] - burst_times[idx]
            current_time += burst_times[idx]
            completed[idx] = True
            completed_count += 1

    print("Process\tArrival\tBurst\tStart\tCompletion\tWaiting\tTurnaround")
    for i in range(n):
        print(f"{processes[i]}\t{arrival_times[i]}\t{burst_times[i]}\t{start_times[i]}\t{com

# Example usage:
processes = ['P1', 'P2', 'P3']
arrival_times = [0, 1, 2]
burst_times = [6, 8, 7]
sjf(processes, arrival_times, burst_times)
```

```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ nano SJF.py

┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python SJF.py
Process Arrival Burst   Start   Completion      Waiting Turnaround
P1      0       6       0       6               0       6
P2      1       8       13      21              12      20
P3      2       7       6       13              4       11
```

3. Round Robin:

```
# Round Robin: Each process gets a fixed time quantum. Processes are executed in a cyclic order.

# CODE:

from collections import deque

def round_robin(processes, arrival_times, burst_times, time_quantum):
    n = len(processes)
    remaining_bt = burst_times[:]
    current_time = 0
    waiting_times = [0] * n
    completion_times = [0] * n
    turnaround_times = [0] * n
    start_times = [-1] * n

    queue = deque()
    visited = [False] * n

    # Add processes that arrive at time 0
    for i in range(n):
        if arrival_times[i] <= current_time and not visited[i]:
            queue.append(i)
            visited[i] = True

    while queue:
        i = queue.popleft()

        if start_times[i] == -1:
            start_times[i] = max(current_time, arrival_times[i])
            current_time = start_times[i]

        exec_time = min(time_quantum, remaining_bt[i])
        remaining_bt[i] -= exec_time
        current_time += exec_time

        # Add newly arrived processes to queue
        for j in range(n):
            if arrival_times[j] <= current_time and not visited[j]:
                queue.append(j)
                visited[j] = True

        if remaining_bt[i] > 0:
            queue.append(i)
        else:
            completion_times[i] = current_time
            turnaround_times[i] = completion_times[i] - arrival_times[i]
            waiting_times[i] = turnaround_times[i] - burst_times[i]

    print("Process\tArrival\tBurst\tStart\tCompletion\tWaiting\tTurnaround")
    for i in range(n):
        print(f"{processes[i]}\t{arrival_times[i]}\t{burst_times[i]}\t{start_times[i]}\t{compl

processes = ['P1', 'P2', 'P3']
arrival_times = [0, 1, 2]
burst_times = [10, 5, 8]
time_quantum = 3
round_robin(processes, arrival_times, burst_times, time_quantum)
```

```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ nano rr.py

┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python rr.py
Process Arrival Burst   Start   Completion      Waiting Turnaround
P1      0       10      0       23              13      23
P2      1       5       3       14              8       13
P3      2       8       6       22              12      20
```