Name: Alistair Shaan Saldanha **SAPID: 60009200024**
Batch: K1

Experiment 4

(CPU Scheduling –Non Preemptive algorithm)

Aim: Implement CPU Scheduling –Non Preemptive Algorithm.

Theory: CPU scheduling is the task of selecting a waiting process from the ready queue and allocating the CPU to it. The CPU is allocated to the selected process by the dispatcher (Itis the module that gives control of the CPU to the processes by short-term scheduler). Scheduling is a fundamental operating system function. In a single processor system, onlyone process can run at a time; any other process must wait until the CPU is free and can berescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the readystate.
3. When a process switches from the waiting state to the readystate.
4. When a process terminates.

Depending on the above circumstances the two types of scheduling are:

1. Non-Preemptive

2. Preemptive

1. **Non-Preemptive:** Under this scheduling, once the CPU has been allocatedto aprocess, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
2. **Preemptive:** Under this scheduling, once the CPU has been allocated to aprocess, the process does not keep the CPU but can be utilized by some other process. This incurs a cost associated with access to shared data. It also affects the design of the operating system kernel.

Scheduling Criteria:

1. **CPU utilization:** It can range from 0-100%. In a real system, it ranges shouldrange from 40-90%.
2. **Throughput:** Number of processes that are completed per unit time.
3. **Turnaround time:** How long a process takes to execute. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O
4. **Waiting time:** It is the sum of the periods spent waiting in the ready queue.
5. **Response time:** Time from the submission of a request until the first response is produced. It is desirable to maximize CPU utilization and Throughput and minimize Turnaround time, waiting time and Response time.

Name: Alistair Shaan Saldanha **SAPID: 60009200024**
Batch: K1

NON-PREEMPTIVE SCHEDULING ALGORITHMS:

1. FCFS
2. SJF
3. PRIORITY

1. FCFS (First-Come, First-Served):

- It is the simplest algorithm and NON-PREEMPTIVE.
- The process that requests the CPU first is allocated to the CPU first.
- The implementation is easily managed by queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue
- The average waiting time, however, is long. It is not minimal and may vary substantially if the process's CPU burst time varies greatly.
- This algorithm is particularly troublesome for time-sharing systems.

2. SJF (Shortest Job First):

- This algorithm associates with each process the length of the process's next CPUburst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS is used to break the tie.
- It is also called shortest next CPU burst algorithm or shortest remaining time first scheduling.
- It is provably optimal; in that it gives the minimum average waiting time for a given set of processes.
- The real difficulty with SJF knows the length of the next CPU request.
- It can be either PREEMPTIVE (SRTF- Shortest Remaining Time First) or NON-PREEMPTIVE.

3. PRIORITY SCHEDULING:

- The SJF is a special case of priority scheduling.
- In priority scheduling algorithm, a priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- It can be either PREEMPTIVE or NON-PREEMPTIVE
- A major problem with priority scheduling algorithms is indefinite blocking, or starvation.
- A solution to starvation is AGING. It is a technique of gradually increasing the priority of process that wait in the system for long time.

Name: Alistair Shaan Saldanha **SAPID: 60009200024**
Batch: K1

Expected Output:

Enter the no. of process: 5

Enter the arrival time for process P1: 0

Enter the arrival time for process P2: 1

Enter the arrival time for process P3: 3

Enter the arrival time for process P4: 9

Enter the arrival time for process P5: 12

Enter the burst time for process P1: 3

Enter the burst time for process P2: 5

Enter the burst time for process P3: 2

Enter the burst time for process P4: 5

Enter the burst time for process P5: 5

Inputs given by the user are:

==========

Process AT BT

P1 0 3

P2 1 5

P3 3 2

P4 9 5

P5 12 5

==========

Gant Chart is as follows:

0->P1->3->P2->8->P3->10->P4->15->P5->20

Name: Alistair Shaan Saldanha **SAPID: 60009200024**
Batch: K1

TABLE

Process	AT	BT	FT	TAT	WT
P1	0	3	3	3.000000	0.000000
P2	1	5	8	7.000000	2.000000
P3	3	2	10	7.000000	5.000000
P4	9	5	15	6.000000	1.000000
P5	12	5	20	8.000000	3.000000

Average Turn Around Time: 6.200000

Average Waiting Time: 2.200000

Lab Assignments to complete in this session

1. Consider the set of 6 processes arrived at zero instance and burst time are 7, 5, 3, 1, 2, 1. If the CPU scheduling policy is First Come First Serve, calculate the average waiting time.
2. Consider the set of 3 processes arrived at zero instance and burst time are 9, 4, 9 respectively and priority 2, 1 and 3 respectively. If theCPU scheduling policy is SJF, calculate the average waiting time and average turnaround time.

NAME: ALISTAIR SALDANHA

SAPID: 60009200024

BATCH: K1

EXPERIMENT-4

FIRST COME FIRST SERVE (FCFS)

In [3]:

```
from prettytable import PrettyTable
x = PrettyTable()
```

Consider the set of 6 processes arrived at zero instance and burst time are 7, 5, 3, 1, 2, 1.

If the CPU scheduling policy is First Come First Serve, calculate the average waiting time.

INPUT

In [4]:

```
nop= int(input("Enter number of processes: "))
processes = {}

for i in range(nop):
    a = int(input(f"Enter the arrival time for process P{i+1}: "))
    b = int(input(f"Enter the burst time for process P{i+1}: "))
    lst = []
    lst.append(a)
    lst.append(b)
    processes[f'P{i+1}'] = lst

# SORTING ACCORDING TO THEIR ARRIVAL TIME
processes = sorted(processes.items(), key=lambda x: x[1][0])
print(processes)
```

```
[('P1', [0, 7]), ('P2', [0, 5]), ('P3', [0, 3]), ('P4', [0, 1]), ('P5', [0, 2]), ('P6', [
0, 1])]
```

GANTT CHART

In [5]:

```
def gantt_c(processes):
    g_c = []
    for i in range(len(processes)):
        if(i==0):
            g_c.append(processes[i][1][1])
        elif(g_c[i-1] < processes[i][1][0]):
            g_c.append(processes[i][1][0])
            completion_time=processes[i][1][0] + processes[i][1][1]
            g_c.append(completion_time)
        else:
            g_c.append(g_c[i-1] + processes[i][1][1])
    return g_c
```

```
gantt_chart = gantt_c(processes)
print(gantt_chart)

def display_gc(g_c):
    print("Gantt Chart")
    for i in range(len(g_c)+1):
        if(i==0):
            print(f"0->P{i+1}->{g_c[i]}->",end="")
        else:
            print(f"P{i+1}->{g_c[i]}",end="")
            if(i < len(g_c)-1):  print("->",end="")
            else:   break
# print(gantt_chart)
display_gc(gantt_chart)
```

```
[7, 12, 15, 16, 18, 19]
Gantt Chart
0->P1->7->P2->12->P3->15->P4->16->P5->18->P6->19
```

CALCULATING TURN AROUND TIME & WAITING TIME

TURN AROUND TIME = COMPLETION TIME - ARRIVAL TIME

WAITING TIME = TURN AROUND TIME - BURST TIME

In [6]:

```
def calc_TAT(gantt_chart,processes):
    TAT = []
    for i in range(len(processes)):
        TAT.append(gantt_chart[i] - processes[i][1][0])
    return TAT
TAT = calc_TAT(gantt_chart,processes)
print("TAT :",TAT)

def calc_WT(TAT,processes):
    WT = []
    for i in range(len(processes)):
        WT.append(TAT[i] - processes[i][1][1])
    return WT
WT = calc_WT(TAT,processes)
print("WT :",WT)

def avg_WT(WT):
    avg_WT = 0.0
    for i in range(len(WT)):
        avg_WT += WT[i]
    avg_WT = (avg_WT/nop)
    return avg_WT
average_WT = avg_WT(WT)

def avg_TAT(TAT):
    avg_TAT = 0.0
    for i in range(len(TAT)):
        avg_TAT += TAT[i]
    avg_TAT = (avg_TAT/nop)
    return avg_TAT
average_TAT = avg_TAT(TAT)
```

```
TAT : [7, 12, 15, 16, 18, 19]
WT : [0, 7, 12, 15, 16, 18]
```

DISPLAY TABLE

In [7]:

```
def table(processes,WT,TAT):
    x.field_names = ["Process","AT","BT","FT","TAT","WT"]
    for i in range(len(processes)):
```

```
        x.add_row([f"P{i+1}",processes[i][1][0],processes[i][1][1],gantt_chart[i],TAT[i]
,WT[i]])
    print(x)

table(processes,WT,TAT)
print(f"Average Waiting Time: {average_WT}")
```

```
+---------+----+----+----+-----+----+
| Process | AT | BT | FT | TAT | WT |
+---------+----+----+----+-----+----+
|    P1   | 0  | 7  | 7  |  7  | 0  |
|    P2   | 0  | 5  | 12 |  12 | 7  |
|    P3   | 0  | 3  | 15 |  15 | 12 |
|    P4   | 0  | 1  | 16 |  16 | 15 |
|    P5   | 0  | 2  | 18 |  18 | 16 |
|    P6   | 0  | 1  | 19 |  19 | 18 |
+---------+----+----+----+-----+----+
Average Waiting Time: 11.333333333333334
```

SHORTEST JOB FIRST

In [8]:

```
from prettytable import PrettyTable
x = PrettyTable()
```

Consider the set of 3 processes arrived at zero instance and burst time are 9, 4, 9 respectively and priority 2, 1 and 3 respectively.

If the CPU scheduling policy is SJF, calculate the average waiting time and average turnaround time.

INPUT

In [9]:

```
nop= int(input("Enter number of processes: "))
processes = {}

for i in range(nop):
    a = int(input(f"Enter the arrival time for process P{i+1}: "))
    b = int(input(f"Enter the burst time for process P{i+1}: "))
    lst = []
    lst.append(a)
    lst.append(b)
    processes[f'P{i+1}'] = lst

# SORTING ACCORDING TO THEIR ARRIVAL TIME
processes = sorted(processes.items(), key=lambda x: x[1][1])
print(processes)
```

```
[('P2', [0, 4]), ('P1', [0, 9]), ('P3', [0, 9])]
```

GANTT CHART

In [10]:

```
def gantt_c(processes):
    g_c = []
    for i in range(len(processes)):
        if(i==0):
            g_c.append(processes[i][1][1])
        elif(g_c[i-1] < processes[i][1][0]):
            g_c.append(processes[i][1][0])
            completion_time=processes[i][1][0] + processes[i][1][1]
            g_c.append(completion_time)
        else:
```

```
            g_c.append(g_c[i-1] + processes[i][1][1])
    return g_c
print(processes)
gantt_chart = gantt_c(processes)

def display_gc(g_c):
    print("Gantt Chart")
    for i in range(len(g_c)+1):
        if(i==0):
            print(f"0->P{i+1}->{g_c[i]}->",end="")
        else:
            print(f"P{i+1}->{g_c[i]}",end="")
            if(i < len(g_c)-1):  print("->",end="")
            else:   break
# print(gantt_chart)
display_gc(gantt_chart)
```

```
[('P2', [0, 4]), ('P1', [0, 9]), ('P3', [0, 9])]
Gantt Chart
0->P1->4->P2->13->P3->22
```

CALCULATING TURN AROUND TIME & WAITING TIME

TURN AROUND TIME = COMPLETION TIME - ARRIVAL TIME

WAITING TIME = TURN AROUND TIME - BURST TIME

In [11]:

```
def calc_TAT(gantt_chart,processes):
    TAT = []
    for i in range(len(processes)):
        TAT.append(gantt_chart[i] - processes[i][1][0])
    return TAT
TAT = calc_TAT(gantt_chart,processes)
print("TAT :",TAT)

def calc_WT(TAT,processes):
    WT = []
    for i in range(len(processes)):
        WT.append(TAT[i] - processes[i][1][1])
    return WT
WT = calc_WT(TAT,processes)
print("WT :",WT)

def avg_WT(WT):
    avg_WT = 0.0
    for i in range(len(WT)):
        avg_WT += WT[i]
    avg_WT = (avg_WT/nop)
    return avg_WT
average_WT = avg_WT(WT)

def avg_TAT(TAT):
    avg_TAT = 0.0
    for i in range(len(TAT)):
        avg_TAT += TAT[i]
    avg_TAT = (avg_TAT/nop)
    return avg_TAT
average_TAT = avg_TAT(TAT)
```

```
TAT : [4, 13, 22]
WT : [0, 4, 13]
```

DISPLAY TABLE

In [12]:

```
def table(processes,WT,TAT):
```

```
    x.field_names = ["Process","AT","BT","FT","TAT","WT"]
    for i in range(len(processes)):
        x.add_row([f"P{i+1}",processes[i][1][0],processes[i][1][1],gantt_chart[i],TAT[i]
,WT[i]])
    print(x)

table(processes,WT,TAT)
print(f"Average Waiting Time: {average_WT} and Average Turnaround Time: {average_TAT}")
```

```
+---------+----+----+----+-----+----+
| Process | AT | BT | FT | TAT | WT |
+---------+----+----+----+-----+----+
|    P1   | 0  | 4  | 4  |  4  | 0  |
|    P2   | 0  | 9  | 13 |  13 | 4  |
|    P3   | 0  | 9  | 22 |  22 | 13 |
+---------+----+----+----+-----+----+
Average Waiting Time: 5.666666666666667 and Average Turnaround Time: 13.0
```