K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: E-2	**Roll No.: 16010123325**
Experiment No. 06	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Basic Process management algorithms - Preemptive (SRTN, RR)

AIM: To implement basic Process management algorithms (Round Robin, SRTN

Expected Outcome of Experiment:
CO 2. To understand the concept of process, thread and resource management.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third**
3. **Edition.**
4. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
5. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Most systems handle numerous processes with short CPU bursts interspersed with I/O requests and a few processes with long CPU bursts. To ensure good time-sharing performance, a running process may be preempted to allow another to run. The ready list, or run queue, maintains all processes ready to run and not blocked by I/O or other system requests. Entries in this list point to the process control block, which stores all process information and state. When an I/O request completes, the process moves from the waiting state to the ready state and is placed on the run queue. The process scheduler, a key component of the operating system, decides whether the current process should continue running or if another should take over. This decision is triggered by four events:

1. The current process issues an I/O request or system request, moving it from running to waiting.
2. The current process terminates.
3. A timer interrupt indicates the process has run for its allotted time, moving it from running to ready.
4. An I/O operation completes, moving the process from waiting to ready, potentially preempting the current process.

The scheduling algorithm, or policy, determines the sequence and duration of process execution, a complex task given the limited information about ready processes.

Description of the application to be implemented:

Round Robin Algorithm

Round Robin (RR) is a **preemptive** CPU scheduling algorithm designed for time-sharing systems. Each process gets a fixed time quantum (or time slice) to execute before being preempted and placed at the end of the queue.

Steps of RR Scheduling:

1. Processes are placed in a ready queue in arrival order.
2. The CPU executes the first process for a fixed time quantum.
3. If the process finishes within the quantum, it exits the system.
4. If the process needs more time, it is preempted and moved to the end of the queue.
5. The next process in the queue gets the CPU.
6. The cycle continues until all processes are completed.

Shortest Remaining Time First Algorithm :

Shortest Remaining Time First (SRTF) is a preemptive version **of** Shortest Job Next (SJN), where the process with the shortest remaining execution time is always scheduled.

Steps of SRTF Scheduling:

1. The CPU always executes the process with the shortest remaining time.
2. If a new process arrives with a shorter remaining time than the current process, the CPU switches to the new process.
3. The cycle continues until all processes are completed.

Implementation details: (printout of code)

SRTF-

```
#include <stdio.h>
#include <limits.h>

#define MAX_PROCS 100

typedef struct {
    int pid, at, bt, rem, ct, tat, wt;
} Proc;

void sortByArrival(Proc p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                Proc temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

void srtf(Proc p[], int n) {
    int time = 0, completed = 0, minIdx;

    while (completed < n) {
        minIdx = -1;
        int minRem = INT_MAX;


        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].rem > 0 && p[i].rem < minRem) {
                minRem = p[i].rem;
                minIdx = i;
            }
        }

        if (minIdx == -1) {
            time++;
            continue;
        }
```

```
        p[minIdx].rem--;
        time++;

        if (p[minIdx].rem == 0) {
            completed++;
            p[minIdx].ct = time;
            p[minIdx].tat = p[minIdx].ct - p[minIdx].at;
            p[minIdx].wt = p[minIdx].tat - p[minIdx].bt;
        }
    }
}

void printProcs(Proc p[], int n) {
    printf("\nPID\tAT\tBT\tRT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].rem, p[i].ct, p[i].tat, p[i].wt);
    }
}

void calculateAverages(Proc p[], int n) {
    int totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++) {
        totalWT += p[i].wt;
        totalTAT += p[i].tat;
    }

    printf("\nAverage Waiting Time (AWT): %.2f", (float)totalWT / n);
    printf("\nAverage Turnaround Time (ATAT): %.2f\n", (float)totalTAT / n);
}

int main() {
    int n;
    Proc p[MAX_PROCS];

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter Arrival time and Burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d (AT BT): ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rem = p[i].bt;
    }
```

```
    sortByArrival(p, n);
    srtf(p, n);
    printProcs(p, n);
    calculateAverages(p, n);

    return 0;
}
```

Output-

```
Enter number of processes: 4
Enter Arrival time and Burst time for each process:
Process 1 (AT BT): 0 7
Process 2 (AT BT): 2 4
Process 3 (AT BT): 4 1
Process 4 (AT BT): 5 4

PID     AT      BT      RT      CT      TAT     WT
1       0       7       0       16      16      9
2       2       4       0       7       5       1
3       4       1       0       5       1       0
4       5       4       0       11      6       2

Average Waiting Time (AWT): 3.00
Average Turnaround Time (ATAT): 7.00
```

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

RR-

```c
#include <stdio.h>

#define MAX_PROCS 100

typedef struct {
    int pid, bt, remaining, ct, tat, wt, rt;
} Proc;

void roundRobin(Proc p[], int n, int quantum) {
    int time = 0, completed = 0;

    while (completed < n) {
        int progressMade = 0;
        for (int i = 0; i < n; i++) {
            if (p[i].remaining > 0) {
                progressMade = 1;
                int executeTime = (p[i].remaining <= quantum) ? p[i].remaining : quantum;

                p[i].remaining -= executeTime;
                time += executeTime;


                if (p[i].remaining == 0) {
                    completed++;
                    p[i].ct = time;
                    p[i].tat = p[i].ct;
                    p[i].wt = p[i].tat - p[i].bt;
                }
            }
        }

        if (!progressMade) {
            time++;
        }
    }
}

void printProcs(Proc p[], int n) {
    printf("\nPID\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }
}
```

```c
void calculateAverages(Proc p[], int n) {
    int totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++) {
        totalWT += p[i].wt;
        totalTAT += p[i].tat;
    }

    printf("\nAverage Waiting Time (AWT): %.2f", (float)totalWT / n);
    printf("\nAverage Turnaround Time (ATAT): %.2f\n", (float)totalTAT / n);
}

int main() {
    int n, quantum;
    Proc p[MAX_PROCS];

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter Burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d (BT): ", i + 1);
        scanf("%d", &p[i].bt);
        p[i].remaining = p[i].bt;
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &quantum);

    roundRobin(p, n, quantum);
    printProcs(p, n);
    calculateAverages(p, n);

    return 0;
}
```

Output-

```
Enter number of processes: 3
Enter Burst time for each process:
Process 1 (BT): 24
Process 2 (BT): 3
Process 3 (BT): 3
Enter Time Quantum: 4

PID     BT      CT      TAT     WT
1       24      30      30      6
2       3       7       7       4
3       3       10      10      7

Average Waiting Time (AWT): 5.67
Average Turnaround Time (ATAT): 15.67
```

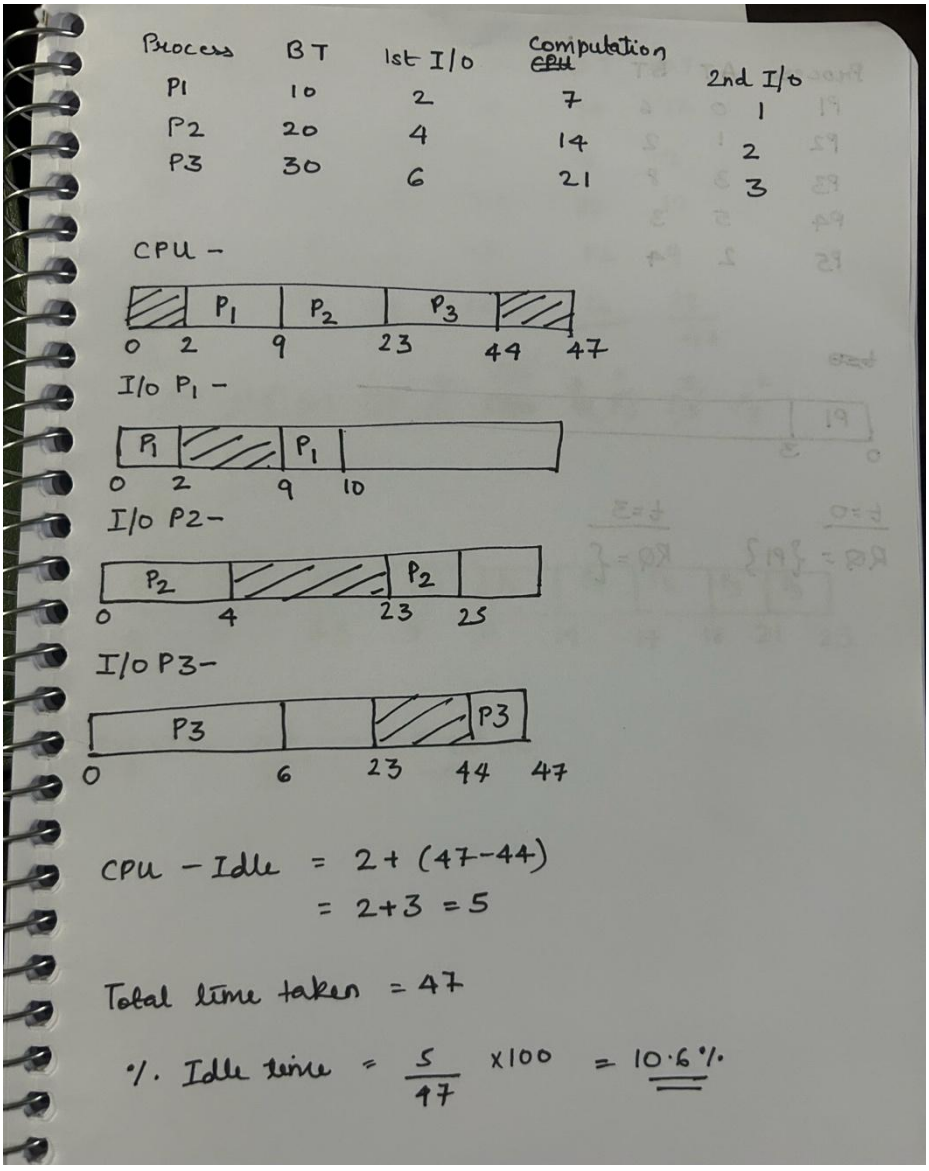
Conclusion:

The above experiment highlights implementation of two preemptive scheduling algorithms – SRTF and Round Robin in C.

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Post Lab Descriptive Questions

1. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?



2. What effect the time quantum has on its performance. What are the advantages and disadvantages of using a small versus a large time quantum?

The time quantum (also called time slice) in Round Robin (RR) scheduling significantly affects the performance of the scheduling algorithm. It determines how long a process gets to execute before the CPU switches to another process.

1. **Small Time Quantum**:

- Leads to frequent context switches.
- Increases CPU overhead (more time spent switching processes rather than executing).
- Improves responsiveness (processes get CPU time more frequently).
- Good for interactive systems (where quick response is needed).
- Can lead **to** high turnaround time due to excessive switching.

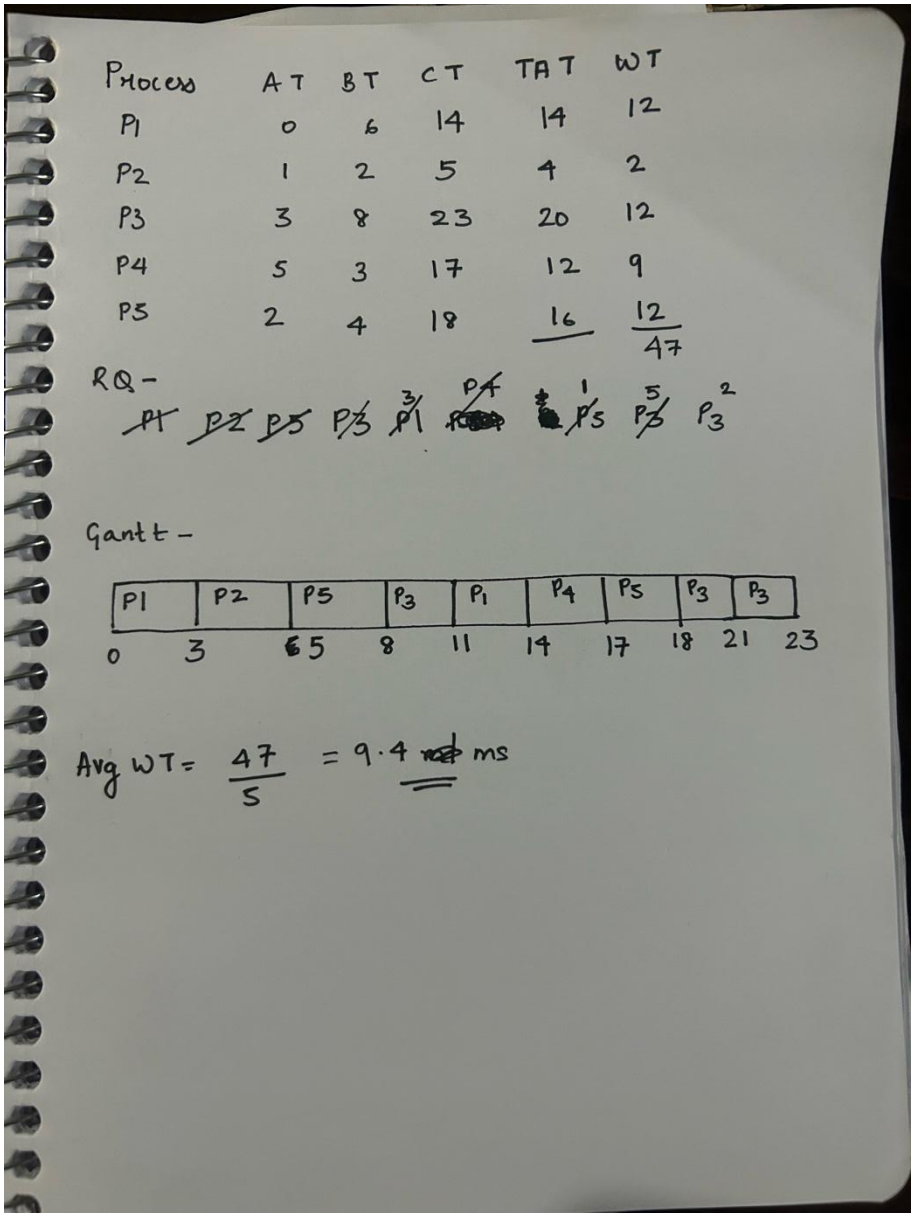
2. **Large Time Quantum**:

- Reduces context switching overhead (CPU spends more time executing processes).
- Behaves more like First-Come-First-Serve (FCFS) if too large.
- Might cause longer response time for shorter processes
- Better for CPU-bound processes that require longer computation time.

3. The following processes are scheduled using the Robin process scheduling policy with a time quantum of 3ms. Determine the average waiting time.

Process	Arrival Time	Burst Time
P1	0	6
P2	1	2
P3	3	8
P4	5	3
P5	2	4

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering



Date: _____ **Signature of faculty in-charge**