Batch: A2	**Roll No.: 16010121045**
Experiment No. 04	
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, priority)

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

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 Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Most systems have a large number of processes with short CPU bursts interspersed between I/O requests and a small number of processes with long CPU bursts. To provide good time-sharing performance, we may preempt a running process to let another one run. The ready list, also known as a run queue, in the operating system keeps a list of all processes that are ready to run and not blocked on some I/O or other system request, such as a semaphore. Then entries in this list are pointers to the process control block, which stores all information and state about a process.

When an I/O request for a process is complete, the process moves from the *waiting* state to the *ready* state and gets placed on the run queue.

The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.

2. The current process terminates.

3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.

4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the*ready* state. The scheduler may then decide to preempt the currently-running process and move this *ready* process into the *running* state.

The decisions that the scheduler makes concerning the sequence and length of time that processes may run is called the scheduling algorithm (or scheduling policy). These decisions are not easy ones, as the scheduler has only a limited amount of information about the processes that are ready to run. A good scheduling algorithm should:

1. Be fair – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
2. Be efficient – keep the CPU busy all the time.
3. Maximize throughput – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
4. Minimize response time – interactive users should see good performance
5. Minimize overhead – don't waste too many resources. Keep scheduling time and context switch time at a minimum.
6. Maximize resource use – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We'll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
7. Avoid indefinite postponement – every process should get a chance to run eventually.

Round Robin Algorithm

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the preemptive version of first come first serve scheduling.
The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Advantages

- It can be actually implementable in the system because it is not depending on the burst time.
- It doesn't suffer from the problem of starvation or convoy effect.
- All the jobs get a fare allocation of CPU.

Disadvantages

- The higher the time quantum, the higher the response time in the system.
- The lower the time quantum, the higher the context switching overhead in the system.
- Deciding a perfect time quantum is really a very difficult task in the system.

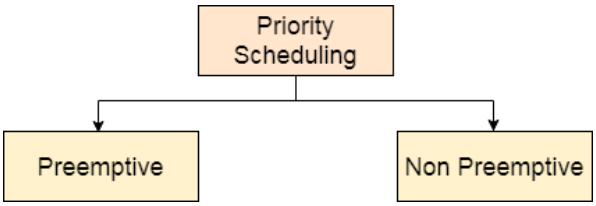
Shortest Remaining Time First Algorithm :

This Algorithm is the preemptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.
Once all the processes are available in the ready queue, No preemption will be done and the algorithm will work as SJF scheduling. The context of the process is saved in the Process Control Block when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the next execution of this process.

Priority scheduling:

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling

algorithm exists. One is Preemptive priority scheduling while the other is Non Preemptive Priority scheduling.



The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called static priority, while if it keeps changing itself at the regular intervals, it is called dynamic priority.

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

Implementation details:

Round Robin:

```
import java.util.*;

public class roundRobin {
    public static void main(String args[]){
        int n,i,qt,count=0,temp,sq=0,bt[],wt[],tat[],rem_bt[];
        float awt=0,atat=0;
        bt = new int[10];
        wt = new int[10];
        tat = new int[10];
        rem_bt = new int[10];
        Scanner s=new Scanner(System.in);
        System.out.print("Enter the number of process= ");
        n = s.nextInt();
        System.out.print("Enter the burst time of the process\n");
        for (i=0;i<n;i++)
        {
        System.out.print("P"+i+" = ");
        bt[i] = s.nextInt();
        rem_bt[i] = bt[i];
        }
        System.out.print("Enter the quantum time: ");
        qt = s.nextInt();
        while(true)
        {
            for (i=0,count=0;i<n;i++)
            {
                temp = qt;
                if(rem_bt[i] == 0)
                {
                    count++;
                    continue;
                }
                if(rem_bt[i]>qt)
                    rem_bt[i]= rem_bt[i] - qt;
                else
                    if(rem_bt[i]>=0)
                    {
                        temp = rem_bt[i];
                        rem_bt[i] = 0;
                    }
                sq = sq + temp;
```

```
                tat[i] = sq;
            }
            if(n == count)
                break;
        }
        System.out.print("\nProcess\t      Burst Time\t       Turnaround
Time\t          Waiting Time\n");
        for(i=0;i<n;i++)
        {
            wt[i]=tat[i]-bt[i];
            awt=awt+wt[i];
            atat=atat+tat[i];
            System.out.print("\n "+(i+1)+"\t "+bt[i]+"\t\t "+tat[i]+"\t\t
"+wt[i]+"\n");
        }
        awt=awt/n;
        atat=atat/n;
        System.out.println("\nAverage waiting Time = "+awt+"\n");
        System.out.println("Average turnaround time = "+atat);
    }
}
```

Output:

```
Enter the number of process= 4
Enter the burst time of the process
P0 = 6
P1 = 34
P2 = 23
P3 = 45
Enter the quantum time: 5

Process       Burst Time              Turnaround Time           Waiting Time

 1       6              21              15

 2       34             93              59

 3       23             74              51

 4       45             108             63

Average waiting Time = 47.0

Average turnaround time = 74.0
```

Priority

```
import java.util.Scanner;

public class priority {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);

        int x,n,p[],pp[],bt[],w[],t[],awt,atat,i;

        p = new int[10];
        pp = new int[10];
        bt = new int[10];
        w = new int[10];
        t = new int[10];

        //n is number of process
        //p is process
        //pp is process priority
        //bt is process burst time
        //w is wait time
        // t is turnaround time
        //awt is average waiting time
        //atat is average turnaround time

        System.out.print("Enter the number of process : ");
        n = s.nextInt();
        System.out.print("\n\t Enter burst time : time priorities \n");

        for(i=0;i<n;i++)
        {
            System.out.print("\nProcess["+(i+1)+"]:");
            bt[i] = s.nextInt();
            pp[i] = s.nextInt();
            p[i]=i+1;
        }

        //sorting on the basis of priority
        for(i=0;i<n-1;i++)
        {
            for(int j=i+1;j<n;j++)
            {
                if(pp[i]>pp[j])
                {
                    x=pp[i];
                    pp[i]=pp[j];
```

```
                    pp[j]=x;
                    x=bt[i];
                    bt[i]=bt[j];
                    bt[j]=x;
                    x=p[i];
                    p[i]=p[j];
                    p[j]=x;
                }
            }
        }
        w[0]=0;
        awt=0;
        t[0]=bt[0];
        atat=t[0];
        for(i=1;i<n;i++)
        {
            w[i]=t[i-1];
            awt+=w[i];
            t[i]=w[i]+bt[i];
            atat+=t[i];
        }

        //Displaying the process

        System.out.print("\n\nProcess \t Burst Time \t Wait Time \t Turn
Around Time   Priority \n");
        for(i=0;i<n;i++)
        System.out.print("\n   "+p[i]+"\t\t   "+bt[i]+"\t\t     "+w[i]+"\
t\t     "+t[i]+"\t\t     "+pp[i]+"\n");
        awt/=n;
        atat/=n;
        System.out.print("\n Average Wait Time : "+awt);
        System.out.print("\n Average Turn Around Time : "+atat);

    }

}
```

```
Enter the number of process : 4

        Enter burst time : time priorities

Process[1]:4 3

Process[2]:2 2

Process[3]:3 4

Process[4]:2 1


Process         Burst Time      Wait Time       Turn Around Time   Priority

   4              2               0               2               1

   2              2               2               4               2

   1              4               4               8               3

   3              3               8               11              4

 Average Wait Time : 3
 Average Turn Around Time : 6
```

Conclusion: Through this experiment we understood the preemptive algorithm for CPU scheduling and implemented Shortest Remaining Time Next (SRTN) and Priority Scheduling.

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?

Ans: Shortest remaining time (SRT) scheduling algorithm selects the process for execution which has the smallest amount of time remaining until completion.
Let three processes be p0, p1 and p2. Their execution time is 10, 20 and 30 respectively. p0 spends first 2 time units in I/O, 7 units of CPU time and finally 1 unit in I/O. p1 spends first 4 units in I/O, 14 units of CPU time and finally 2 units in I/O. p2 spends first 6 units in I/O, 21 units of CPU time and finally 3 units in I/O.

PID	AT	IO	BT	IO
P0	0	2	7	1
P1	0	4	14	2
P2	0	6	21	3

AT- Arrival Time, IO-input/output, BT-Burst Time

First process p0 will spend 2 units in IO, next 7 units in BT, then process p1 will spend 14 units in BT (as its 4 units of IO has been spent already when previous process was running) and ten process p2 will spend 21 units in BT (as its 6 units of IO has been spent already when previous processes were running) and at last 3 units in IO (process p0,p1,p2's last IO included.) idle p0 p1 p2 idle 0 2 9 23 44 47

- Total time spent = 47
- Idle time = 2 + 3 = 5
- Percentage of idle time = (5/47)*100 = **10.6 %**

2. What is Starvation?

Ans: Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

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