RESEARCH PAPER

SHORTEST JOB FIRST SCHEDLING ALGORITHM

SUBMITTED BY :

MEHAK ZAHRA (2022 - BSE - 022)

MAHNOOR SEEMAB (2022 - BSE - 018)

RAFIA SHAKIL (2022 - BSE - 026)

ALISHBAH WASEEM MALIK (2022 - BSE- 003)

SUBMITTED:

SIR SHEHZAD

FATIMA JINNAH WOMEN UNIVERSITY

An Optimized Shortest job first Scheduling Algorithm for CPU Scheduling

ABSTRACT:

This paper proposes an enhanced preemptive Shortest Job First (SJF) scheduling algorithm to improve CPU efficiency in real-time and time-sharing environments. The proposed approach aims to mitigate issues associated with high context switching, response time, waiting time, throughput, and turnaround time. Comparative analysis with round robin and traditional preemptive SJF algorithms demonstrates improved system performance, particularly in reducing context switching.

INTRODUCTION:

CPU scheduling, a fundamental aspect of operating systems, involves granting permissions to processes for accessing system resources to enhance performance and ensure fair utilization. It is a critical element in CPU resource management. Determining the order and resources for executing processes, CPU scheduling plays a pivotal role in efficient resource utilization and overall system performance, particularly when multiple processes are ready for execution.

This study focuses on multi-processor systems, exploring their efficiency in scheduling multiple jobs for processing. While scheduling on a single processor is comparatively simpler, real-time processing introduces challenges with timing restrictions.

For a real-time OS, unified scheduling and resource distribution are crucial. The proposed algorithm aims to overcome limitations, providing an efficient and user-friendly solution while bridging gaps in existing scheduling algorithms. It facilitates users in exploring the multiprogramming environment.

Shortest Job First (SJF) challenges in existing scheduling algorithms and introduces a proposed algorithm with the goal of being more efficient and user-friendly, addressing limitations in current approaches.

I. RELATED WORK:

Numerous recent papers delve into operating system scheduling algorithms, with this paper extending previous research Researchers often propose unique algorithms, such as First Come First Served (FCFS), Priority-based, Round Robin, and Short Remaining Time First Schedule (SRTF). FCFS selects processes in the order they arrive, while Priority-based arranges processes based on system-assigned priorities. Round Robin is a preemptive technique using time quantum. "Short Remaining Time First Schedule" is a preemptive algorithm that prioritizes processes based on burst times. Processes with shorter times are given preference, and preemption occurs if a process with a shorter burst time arrives. After completion, the terminated process is removed, and a new one is chosen from the front of the ready queue.

In the Smallest Jobs Scheduled First (SJF) algorithm, the ready queue is ordered by CPU burst duration. SJF is optimal, with lower waiting time and improved throughput compared to other algorithms. However, it faces challenges with high context switching, impacting system performance. Minimizing context switching is crucial for maximizing system efficiency.

II. PROPOSED:

The proposed hybrid scheduling algorithm combines Shortest Job First (SJF) and constraints on the remaining burst time of the running process. This preemptive algorithm initially selects the job with the shortest burst time for execution. When a new job enters the ready queue, the algorithm decides whether to preempt the running process based on the remaining execution time. If the remaining time is less than or equal to the burst time of the new process, no preemption occurs. However, if the remaining time is greater, the decision depends on whether half of the remaining execution time is less than or greater than the burst time of the new process.

This approach aims to optimize system performance by minimizing context switching. Although there is a slight change in average waiting time compared to SJF, the proposed algorithm significantly reduces context switching. The average waiting time of the proposed approach is consistently lower than other scheduling techniques.

III. CASE STUDY:

We select five processes with their arrival time and burst time. These processes are scheduled by Round Robin, Preemptive SJF and proposed algorithm. We calculate and compare average waiting time, contest switching and turnaround time of each scheduling algorithm. We execute many processes sets and calculate and compare them. The example is shown below,

Table 1:A set of five processes with arrival time and burst time

Processes	Time of Arrival	Burst Duration (ms)
P1	0	4.0
P2	2	7.0
P3	5	5.0
P4	6	8.0
P5	8	9.0

1. **Round Robin with Quantum = 5 Gantt Chart:**

P1	P2	P3	P4	P5	P2	P4	P5

No. of Context Switches = 7
Average Waiting Time = 11.4ms
Average Turnaround Time = 18ms

2. **Shortest Remaining Job First:**

Gantt Chart:

P1	P2	P3	P2	P4	P5	

No. of Context Switches = 5
Average Waiting Time = 6.6ms
Average Turnaround Time = 13.2ms

3. **Proposed Algorithm:**

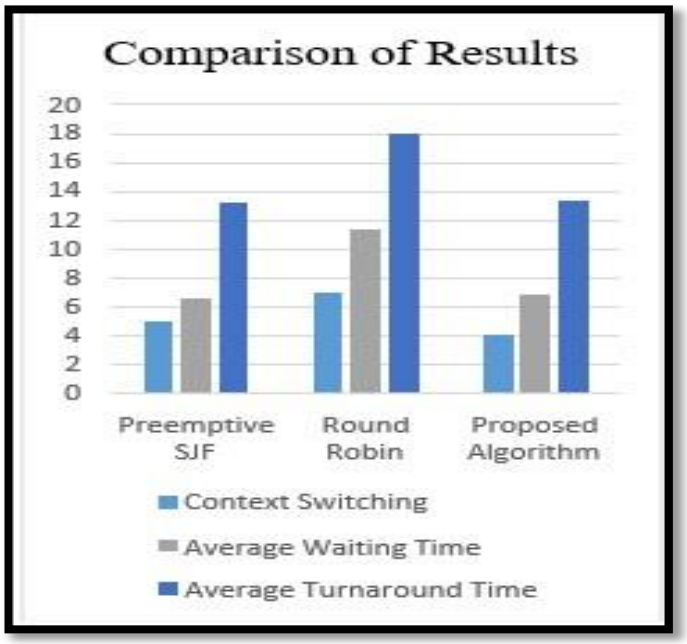
Gantt Chart:

P1	P2	P3	P2	P4	P5	

No. of Context Switches = 4
Average Waiting Time = 6.8ms
Average Turnaround Time = 13.4ms

IV. RESULTS AND DISCUSSIONS:

To assess the new scheduling algorithm's performance, we applied the same set of processes to various scheduling algorithms. Comparative results indicate that the proposed approach experiences fewer context switches than Round Robin and preemptive SJF. Context switching, known for its negative impact on system performance, is minimized by the proposed approach, thereby maximizing overall system performance. The performance evaluation chart below illustrates the superiority of the proposed algorithm over Round Robin and preemptive SJF.



IMPLEMENTATION:

```
 GNU nano 6.2                                                    file.c *
#include <stdio.h>
int main()
{
        int A[100][4];
        int i, j, n, total = 0, index, temp;
        float avg_wt, avg_tat;
        printf("Enter number of process: ");
        scanf("%d", &n);
        printf("Enter Burst Time:\n");
        for (i = 0; i < n; i++)
        {
                printf("P%d: ", i + 1);
                scanf("%d", &A[i][1]);
                A[i][0] = i + 1;
        }
        for (i = 0; i < n; i++)
          {
                index = i;
                for (j = i + 1; j < n; j++)
                        if (A[j][1] < A[index][1])
                                index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;

                temp = A[i][0];
                A[i][0] = A[index][0];
                A[index][0] = temp;
        }
```

```
GNU nano 6.2                                                    file.c *
        }
        for (i = 0; i < n; i++)
          {
                index = i;
                for (j = i + 1; j < n; j++)
                        if (A[j][1] < A[index][1])
                                index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;

                temp = A[i][0];
                A[i][0] = A[index][0];
                A[index][0] = temp;
        }
        A[0][2] = 0;

        for (i = 1; i < n; i++) {
                A[i][2] = 0;
                for (j = 0; j < i; j++)
                        A[i][2] += A[j][1];
                total += A[i][2];
        }
        avg_wt = (float)total / n;
        total = 0;
        printf("P        BT      WT      TAT\n");

        for (i = 0; i < n; i++) {
                A[i][3] = A[i][1] + A[i][2];
                total += A[i][3];
                printf("P%d      %d      %d      %d\n", A[i][0],
                        A[i][1], A[i][2], A[i][3]);
        }
        avg_tat = (float)total / n;
        printf("Average Waiting Time= %f", avg_wt);
        printf("\nAverage Turnaround Time= %f", avg_tat);
}
```

```
~$ nano file.c
~$ gcc file.c
~$ ./a.out
Enter number of process: 5
Enter Burst Time:
P1: 3
P2: 5
P3: 4
P4: 1
P5: 7
P         BT      WT      TAT
P4        1       0       1
P1        3       1       4
P3        4       4       8
P2        5       8       13
P5        7       13      20
Average Waiting Time= 5.200000
Average Turnaround Time= 9.200000~$
```

CONCLUSION:

Previous results reveal that First Come First Serve (FCFS) has minimal computational overhead but yields low throughput and high waiting time, indicating poor performance. Shortest Job First (SJF) reduces average waiting time, but for processes with longer execution times, it may lead to increased waiting and potential starvation if short processes arrive continuously. Round Robin ensures fair CPU sharing but can result in longer turnaround times for processes with lengthy burst times. Priority-based scheduling introduces starvation for lower-priority processes. The proposed algorithm is an extension of preemptive SJF, known for optimal performance with lower waiting time and higher throughput. It minimizes context switching, slightly altering average waiting time. The proposed approach significantly enhances system performance by reducing context switching.

REFERENCES:

https://www.researchgate.net/publication/341621055_Analysis_of_Preemptive_Shortest_Job_First_SJF_ Algorithm_in_CPU_Scheduling

YSR