***An Optimized Shortest job first Scheduling Algorithm for CPU Scheduling***

**CONTENT :**

* Abstract
* Problem Statement
* Introduction
* Literature review
* Methodology
* Algorithm
* Flow Chart
* Experiment
* Graph
* Conclusion

**Abstract:**

The multi task environment is turned on and off in modern operating systems. In today's computing, resource consumption must be maximized. To schedule CPU resources, a variety of scheduling algorithms are available. Due to frequent context switches, these methods cannot be applied in a real-time setting.

High response times, long wait times on average, low throughput, and lengthy turnaround times. The goal of this work is to provide a novel preemptive shortest job first strategy that boosts CPU effectiveness in time-sharing and real-time environments. The shortcomings of the shortest job first preemptive scheduling algorithm are improved by the suggested method. Round robin and preemptive SJF algorithms are used to compare the suggested approach.The comparison findings demonstrate that the suggested approach enhances system performance by reducing context switching to an acceptable level. CPU scheduling, operating system, and keywords.

**Problem statement :**

* There are numerous scheduling algorithms for allocating system resources utilizing current scheduling algorithms . Every one of the mentioned algorithms has some kind of Limits. Some have minimum waiting or latency, whereas others offer high throughput. It has multiple context switches and isn't entirely flawless.
* Our main focus is to optimize the system performance by minimizing the context switching.

**Introduction :**

SJF is an optimal scheduling algorithm as its average waiting time is less than other algorithms. It’s throughput is quite better than the other scheduling algorithms it shows the problem of high context switching. Context switching is also an overhead. It decreases the system performance. So there is a need to minimize the context switching to maximize the system performance Therefore the proposed algorithm minimize the context switching of the preemptive sjf.

**literature review :**

• Multitasking environments replace single tasking environments in current operating systems. In today's computing, efficiency must be minimized. To schedule CPU resources, a variety of scheduling algorithms can be used. The implementation of these algorithms is not real-time.

• The ready queue for this algorithm is comparable to the ready queue that exists in the FIFO. environments owing to multiple context shifts, high reaction times, long average latency, low throughput, and turn around time. The process picks a starting point in the ready queue and starts running. The currently active process terminates and is taken from the list of processes once it has finished its execution. When using priority-based scheduling strategies, the system's priority is used to determine where to place the available processes.

With **priority-based scheduling techniques**, the available processes are placed according to the priority specified by the system.

• **Round robin** is another scheduling technique used to schedule processes. This is a preemptive scheduling technique, and this algorithm uses the concept of quantum time.

• "**Short Remaining Time First Schedule"** is a planning algorithm. With this scheduling algorithm, ready queues are organized according to the process burst time.

• **The Smallest Job Scheduled First (SJF)** scheduling algorithm arranges ready queues to pool CPU time periods. With these algorithms, the activity with the shortest burst time is queued before or queued to the queue.

**Methodology:**

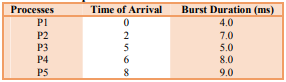
The algorithm that is being suggested is a hybrid scheduling algorithm. This algorithm is made up of two algorithms: the shortest job first (SJF) algorithm and the constrained on remaining burst time of running process algorithm. The preemptive scheduling algorithm category includes this algorithm. The idea behind this method is that it chooses the work with the least burst time at startup and begins running this process. Because it is a preemptive scheduling algorithm—as was said above—it decides whether to preempt a current process when a new work enters the ready queue or not. Running processes will not preempt if their remaining execution time is equal to or less than the burst time of the newly arrived process.

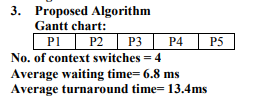
1. If half of the remaining execution time of running process is less than the burst time of newly process, then

running process will not preempt.

2. If half of the remaining execution time of running process is greater than the burst time of newly process,

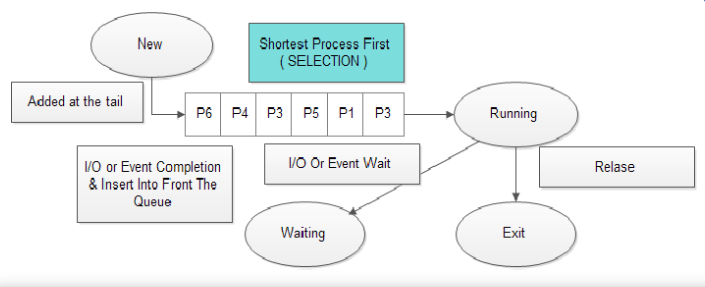
then we will preempt running process and assign processor to newly arrived process.

****

****

**Algorithm :**

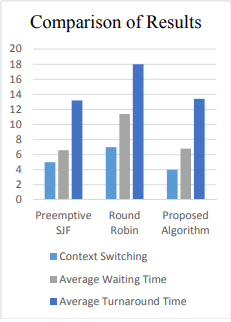
* Choose the method with the shortest burst time and shortest arrival time. After the process is finished, choose the one with the shortest burst time from the ready queue. Repeat the aforementioned steps until they are all complete.
* If half of the remaining execution time of running process is less than the burst time of newly process, thenrunning process will not preempt.
* .If half of the remaining execution time of running process is greater than the burst time of newly process, then we will preempt running process and assign processor to newly arrived process.

****

**Experiment:**

Give several scheduling algorithms the same set of processes to compare how well they execute when used with the newly created scheduling methods. The comparison findings reveal that the suggested method has less context switching than round robin and preemptive SJF. Context switching, as you know, increases overhead and reduces system performance. By reducing context switching, the proposed approach applicability can fully maximize system performance. The suggested algorithm's performance evaluation in comparison to round robin and preemptive SJF is displayed in the following table and graphic.

**Results Graph :**

****

**Conclusion:**

Our suggested algorithm is a development of preemptive SJF. SJF is an ideal scheduling algorithm, as shown by the fact that its average waiting time is less than different algorithms Its throughput is slightly higher than that of the methods. In comparison to SJF, the proposed approach significantly reduces context switching while just slightly altering the average waiting time. The average waiting time has decreased over time. Therefore, by switching contexts less frequently, these techniques improve system performance.