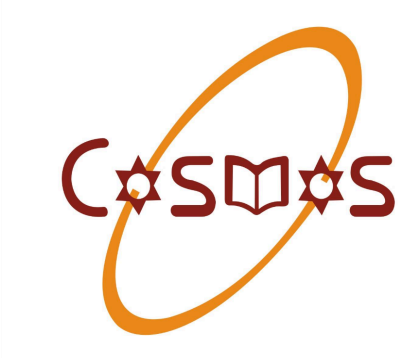
Cosmos College of Management and Technology
Affiliated to Pokhara University
Saddobato, Lalitpur



Lab Report on: Implementation of SRTF Algorithm

Lab report number: 06

Submitted By

Name: Aayush Karki

Roll num: 200101

Faculty: BE IT

Semester: V

Sub: AOS

Group: A

Submitted To

Department of ICT

2080/10/05

.....

Lab Number: 06

Lab Title: Implementation of Shortest Remaining Time First (SRTF) Algorithm

Lab Objective

1. Understand the concept of Shortest Job First and its preemptive version which is Shortest Remaining Time First (SRTF)
2. Understand the Scheduling algorithm and its need in the Operating system.
3. Understand about preemptive and non preemptive nature of algorithms

Theory

A scheduling algorithm is a method or set of rules used by an operating system to determine the order in which processes or tasks are executed on a computer's central processing unit (CPU). The primary goal of a scheduling algorithm is to efficiently utilize the CPU and provide a fair and responsive environment for running processes. Scheduling algorithms are crucial for managing the execution of multiple processes in a system, especially in multi-user or multitasking environments where several processes may be competing for CPU time simultaneously. The scheduler decides which process to run next, when to switch between processes, and how to allocate CPU resources. There are various kinds of scheduling algorithms that is used in the CPU based on the type of operation to be performed and some of them are as follows:

1. First come first serve (FCFS)
2. Shortest Job First (SJF)
3. Shortest remaining time first (SRTF)
4. Priority Scheduling
5. Round Robin algorithms & many more

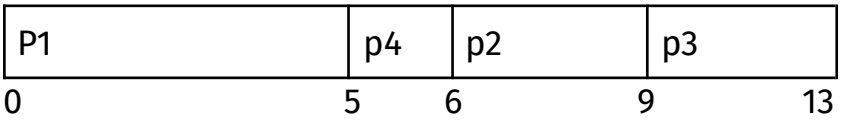
Preemptive scheduling is the nature of the algorithm where the operating system has the capability to stop or interrupt a currently executing process and start or resume another process. The decision to switch between processes can be made at any time. Preemptive scheduling allows for more responsiveness and flexibility, particularly in environments where tasks with varying priorities need to be handled dynamically.

Similarly non preemptive scheduling is the nature of the algorithm where the operating where a running process cannot be interrupted; it continues executing until it completes its burst or time quantum. The operating system only switches between processes under specific conditions, such as when the currently executing process voluntarily releases the CPU (e.g., through I/O request or completion). Non-preemptive scheduling is simpler to implement but may result in less responsiveness, especially in situations where a higher-priority task arrives after a lower-priority task has started execution.

Shortest Job First is one of the types of scheduling algorithm and this algorithm is non preemptive in nature. The Shortest Job First algorithm with preemptive nature is called to be ***Shortest Remaining Time First*** algorithm. Talking about the SJF with non preemptive nature, it basically uses the CPU based on the arrival time. And if two processes arrive at the same time where one has lower burst time as compared to the other one, the process with lowest burst time uses the CPU at first. The below example demonstrates the ***SJF*** algorithm in detailed manner:

Process	Arrival Time	Burst Time
P1	0	5
p2	1	3
p3	2	4
p4	4	1

Gantt Chart



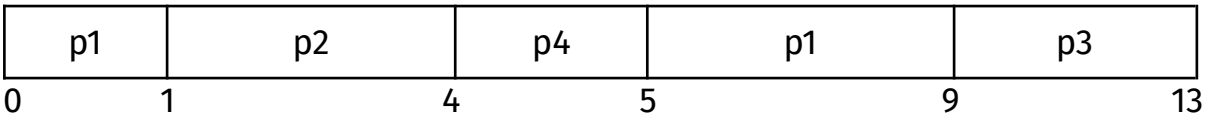
Here at first the arrival time of p1 is 0 having a burst time of 5 sec. Since no other process has arrived at that time, there comes no question of comparison of burst time. Hence p1 gets the CPU till it gets completed i.e. till 5 second. When 5 seconds have passed, processes p2, p3, p4 have already arrived in the ready queue, so now they will

get the CPU based on their burst time. The burst time of p4 is smaller among p2, p3, p4. So it gets the CPU till it gets completed. After that, out of p2 and p3, p2 has smaller burst time, due to which p2 gets the CPU and at last from 9 to 13 second, p3 gets the CPU. So in this way the SJF algorithm manages the CPU access.

Similarly, the SRTF works based on checking the burst time after every instant of time. That means the process gets the CPU for 1 second and again its burst time is checked by the other processes that have already arrived in the ready queue and according to their burst time, the CPU access is provided. The simple example of ***SRTF*** scheduling algo is as follows:

Process	Arrival Time	Burst Time
p1	0	5
p2	1	3
p3	2	4
p4	4	1

Gantt Chart



CODE

In this lab 06, we only implemented the SRTF algorithm not the SJF one. Hence the below code demonstrates the implementation of SRTF algorithm.

```
/* Here we are implementing the SRTF algo which is the preemptive version of SJF */
#include <stdio.h>
#include <stdbool.h>
```

```
struct Process
{
	int process_id;
	int burst_time;
	int remaining_time;
	bool completed;
};

void swap(struct Process *a, struct Process *b)
{
	struct Process temp = *a;
	*a = *b;
	*b = temp;
}

void sort_by_remaining_time(struct Process process[], int n)
{
	for (int i = 0; i < n - 1; i++)
	{
		for (int j = 0; j < n - 1; j++)
		{
			if (process[j].remaining_time>process[j + 1].remaining_time)
			{
				swap(&process[j], &process[j + 1]);
			}
		}
	}
}

void srtf(struct Process process[], int n)
{
	int current_time = 1;

	while (true)
	{
		int shortest = -1;
		// find the process with the shortest remaining time
		for (int i = 0; i < n; i++)
		{
```

```
            if (!process[i].completed && process[i].burst_time > 0 &&
            (shortest == -1 || process[i].remaining_time <
            process[shortest].remaining_time))
            {
            shortest = i;
            }
        }

        if (shortest == -1)
        {
            // All process are completed
            break;
        }

        // Execute the process
        process[shortest].burst_time--;
        process[shortest].remaining_time--;

        printf("Time %d: Process%d\n", current_time,
        process[shortest].process_id);
        current_time++;

        // Check if the process is completed
        if (process[shortest].burst_time == 0)
        {
            process[shortest].completed == true;
        }

        // sort process based on remaining time for the next iteration
        sort_by_remaining_time(process, n);
    }
}

int main()
{
    int n;
    printf("Enter the number of process: ");
    scanf("%d", &n);

    struct Process processes[n];
```

```
    // Input burst times for each process
    for (int i = 0; i < n; i++)
    {
        processes[i].process_id = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].completed = false;
    }

    // Sort process based on burst time for the initial state
    sort_by_remaining_time(processes, n);
    // execute processes using SRTF
    srtf(processes, n);

    return 0;
}
```

Discussion

The Shortest Remaining Time First (SRTF) algorithm, demonstrated in the code, is a preemptive version of the Shortest Job First (SJF) algorithm. The algorithm selects the process with the shortest remaining burst time for execution at any given time. The implementation involves continually checking and selecting the process with the shortest remaining time, allowing for dynamic adaptation to arriving processes. The Gantt chart provided in the theory section helps visualize how the SRTF algorithm allocates CPU time to processes based on their burst times. An example with processes P1, P2, P3, and P4 illustrates how the algorithm executes and switches between processes based on their burst times.

SJF is non-preemptive, meaning that once a process starts executing, it continues until completion. SRTF, being preemptive, allows for dynamic switching between processes based on their remaining burst times. The preemptive nature of SRTF enhances responsiveness, as shorter processes can be given CPU time even if a longer process is currently running. The C code provided implements the SRTF algorithm. It defines a

Process structure and includes functions for swapping processes, sorting them by remaining time, and implementing the SRTF scheduling logic.

CONCLUSION

SRTF improves system responsiveness by dynamically selecting processes with the shortest remaining burst times. It can lead to lower turnaround times and better overall system performance. Its preemptive nature of SRTF allows for adaptability in a multitasking environment, where processes may have varying burst times.

While SRTF has advantages, it may suffer from the "starvation" problem, where long processes may be continuously preempted by shorter ones, leading to delayed execution. Talking about its application, SRTF is suitable for scenarios where responsiveness and efficient CPU utilization are crucial, such as interactive systems or real-time environments.

Thus, in a nutshell, SRTF is a preemptive scheduling algorithm that dynamically selects processes based on their remaining burst times, providing improved responsiveness in multitasking environments. However, careful consideration is needed to address potential issues such as starvation.