SHORTEST JOB FIRST(SJF)

-----------------------------------------------------

****

**OPERATING SYSTEM MINI PROJECT ON**

**SHORTEST JOB FIRST(SJF) SCHEDULING ALGORITHM**

**Submitted in the partial fulfillment for the academic requirements**

**lV th Semester B.tech**

**In**

**Computer Science Engineering (CSE)**

**Submitted by**

**P.Surya narayana.**

**REG.NO :: 11802507**

**ROLL.NO:: 26**

**Email :: suryanarayana13271@gmail.com**

**GUIDE**

**Dr.Aarti Madam.**

**2019-2020**

**PROBLEM STATEMENT** :

Implementation of Shortest Job first(SJF) scheduling algorithm.

**INTRODUCTION** :

Shortest job first is a scheduling algorithm in which the process with the smallest execution time is selected for execution next. Shortest job first can be either preemptive or non-preemptive. Owing to its simple nature, shortest job first is considered optimal. It also reduces the average waiting time for other processes awaiting execution.

Shortest job first is also known as shortest job next (SJN) and shortest process next (SPN).

Shortest job first depends on the average running time of the processes. The accurate estimates of these measures help in the implementation of the shortest job first in an environment, which otherwise makes the same nearly impossible to implement. This is because often the execution

burst of processes does not happen beforehand. It can be used in interactive environments where past patterns are available to determine the average time between the waiting time and the commands. Although it is disadvantageous to use the shortest-job-first concept in short-term CPU scheduling, it is considered highly advantageous in long-term CPU scheduling. Moreover, the throughput is high in the case of shortest job first.  
  
Shortest job first also has its share of disadvantages. For one, it can cause process starvation for longer jobs if there are a large number of shorter processes. Another is the need to know the execution time for each process beforehand. Often, this is almost impossible in many environments.

**ALGORITHM:**

1- sort all the process to increasing order of their burst time (bt). get arrival time (at) and obtain completion time ct.

2- calculate turn-around time tat=ct-at and Find waiting time (wt=tat-bt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find **waiting time** for all other processes i.e. for

process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

6- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

7- Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

**PROGRAM:**

**#include<stdio.h>**

**#include<conio.h>**

**# define max 30**

**int main()**

**{**

**int i,j,n,t,p[max],at[max],bt[max],wt[max],tat[max];**

**float awt=0,atat=0;**

**printf("Enter the number of processes :");**

**scanf("%d",&n);**

**printf("Enter the process number : ");**

**for(i=0;i<n;i++)**

**{**

**scanf("%d",&p[i]);**

**}**

**printf("Enter The Arrival Time : ");**

**for(i=0;i<n;i++)**

**{**

**scanf("%d",&at[i]);**

**}**

**printf("Enter the Burst Time of the process : ");**

**for(i=0;i<n;i++)**

**{**

**scanf("%d",&bt[i]);**

**}**

**printf("Sorting the process according to the burst time :");**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<n;j++)**

**{**

**if(bt[j]>bt[j+1])**

**{**

**t=bt[j];**

**bt[j]=bt[j+1];**

**bt[j+1]=t;**

**}**

**}**

**}**

**printf("\n process \t burst time \t Arrival Time \t waiting time \t turn around time \n");**

**for(i=0;i<n;i++)**

**{**

**wt[i]=0;**

**tat[i]=0;**

**for(j=0;j<i;j++)**

**{**

**wt[i]=wt[i]+bt[j];**

**}**

**tat[i]=wt[i]+bt[i];**

**awt=awt+wt[i];**

**atat=atat+tat[i];**

**printf("%d\t\t\t %d\t\t %d\t\t %d\t\t %d\n",p[i],bt[i],at[i],wt[i],tat[i]);**

**}**

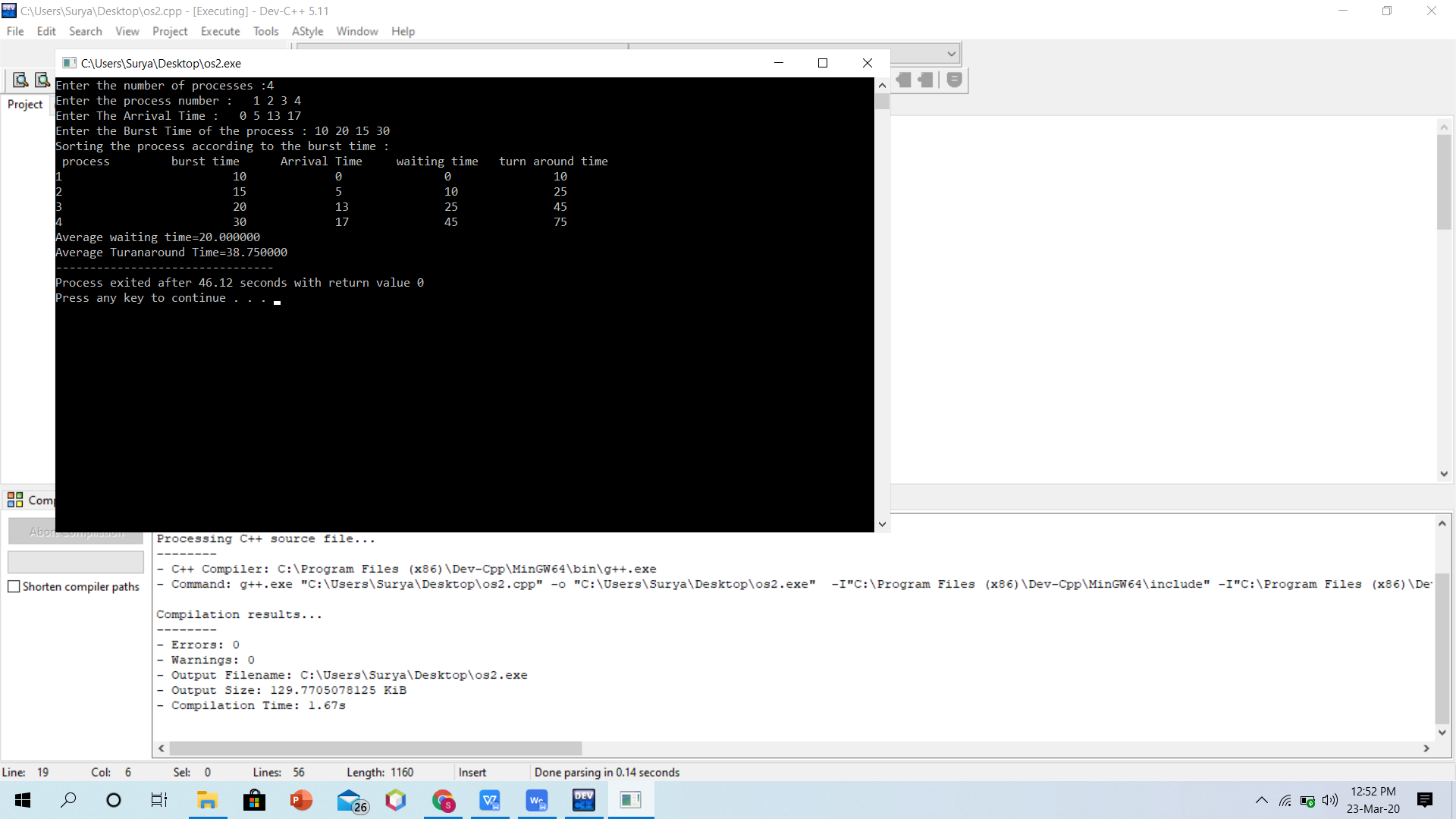
**awt=awt/n;**

**atat=atat/n;**

**printf("Average waiting time=%f\n",awt);**

**printf("Average Turanaround Time=%f",atat);**

**}**

****

**Time complexity :: O(n)+O(n)+O(n)+O(n^2)+O(n^2)**

**:: 3O(n)+2O(n^2)**

**::O(n^2)**

**Functional Requirements:**

**1.** **Software Hardware Requirement**

Dev c++ (c programming)

The Software Requirements are dev c++, Windows XP.

The Hardware Requirements are minimum 512 MB RAM, processor

* 1. **Introduction:**

1.1 Purpose of this document

To schedule the operating system process based on the sjf,srt algorithm s

1.2 Scope of this document

The scope of this study was limited to the Linux scheduler and the Preemptive

priority scheduling algorithm. At the evaluation phase, performance and stability

characteristics were compared with those of the Linux 2.6.11 CPU scheduler.

The C programming language was used in the development of the scheduler prototype.

C has many desirable features, however there was not much of a choice

since this is the language of the Linux kernel (Linen, 2000 [15]). The software development

and tests were carried out on a single processor Intel architecture based

personal computer.

1.3 Definitions, Acronyms and Abbreviations

* Srs – Software Requirement Specification
* User – a person who will be using the system for its intended purpose
* SJF-Shortest Job First
* SRT-Shortest Remaining Time

The goals are:

The goal is developa model CPU scheduler to guard against runway dispatch latency, and gurantee a optimum level of scheduler effiency.

**1.4 General Description**

The remaining part of this document follows the IEEE STD 830-1993 format. It will cover the product perspective, functions, user characteristics, constraints and assumptions. It will cover the functional and non-functional requirements. It will cover the operational requirements.

1.5 .Product Perspective::

The system will be developedusing JAVA

1.6 Product Functions

To schedule the proceses of operating system CPU

1.7 User characteristics

User will enter the number of process to be schedule, bursttime, arrival time

**2.Functional Requirements**

2.1 Functional Requirements

The user should be able to schedule the processes

2.2 Operation performed by user

2.3.Introduction

Scheduling algorithm will schedule the process.

2.4 Inputs

No. of process , burst time arrival time

2.5 Processing

These will schedule the process.

2.6 Outputs

Turn around time,Average throught put, Waiting Time.

2.7 Interface Requirements

2.8 User Interfaces

2.9 The Command line Screen

Input will be provided in command window.

2.10 Applet viewer window

Output should be shown in this window

2.11 Hardware Interfaces

All necessary hardware interfaces will be proided by the developer.

2.12:: Software Interfaces

The system shall be compatible to work as a applet window based application in JAVA .

2.13:: Performance Requirements  
Burst time of process

Arrival time

2.14:: Design Constraints

2.15:: Standard Compliance

The IEEE format is used for technical documentation specification in this document.

2.16 Hardware Constraints

The software system runs on Windows XP that requires minimum 1GB RAM and Centrino Duo Processor

2.17 Software Constraints (limitations)

The system shall meet all performance requirements running on the C application

**Conclusion:**

Shortest job next is advantageous because of its simplicity and because it minimizes the average amount of time each process has to wait until its execution is complete. However, it has the potential for [process starvation](https://en.wikipedia.org/wiki/Process_starvation" \o "Process starvation) for processes which will require a long time to complete if short processes are continually added. [Highest response ratio next](https://en.wikipedia.org/wiki/Highest_response_ratio_next" \o "Highest response ratio next) is similar but provides a solution to this problem using a technique called [aging](https://en.wikipedia.org/wiki/Aging_(scheduling)" \o "Aging (scheduling)).

Shortest job next can be effectively used with interactive processes which generally follow a pattern of alternating between waiting for a command and executing it. If the execution burst of a process is regarded as a separate "job", past behaviour can indicate which process to run next, based on an estimate of its running time.