

**Mawlana Bhashani Science and Technology University**

**Lab-Report**

Report No:08

Course code: ICT-3110

Course title: Operating System Lab

Date of Performance:08/09/2020

Date of Submission:10/09/2020

**Submitted by Submitted To**

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Name:Md Asikur Rahman

ID:IT-18025

3rd  year 1st semester

Session: 2017-2018

Dept. of ICT

MBSTU.

# Experiment no : 08

# Experiment Name : Implementation of SJF Scheduling Algorithm

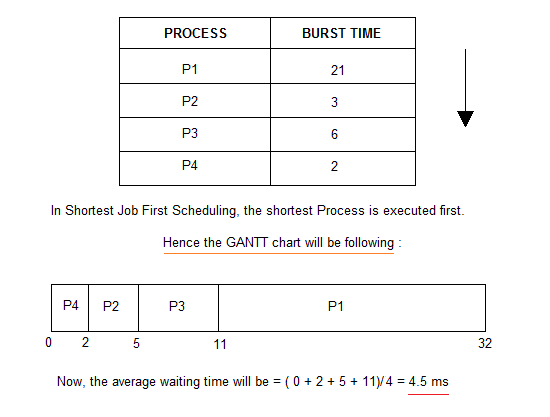
# Theory:

SJF is a scheduling algorithm that assigns to each process the length of its next CPU burst/execution time. CPU is then given to the process with the minimal CPU burst from the waiting queue. SJF is provably optimal, in that for a given set of processes and their CPU bursts/execution times it gives the least average waiting time for each process. The average waiting time for a process is defined by:

S=(W1+W2+...+Wn)/n [1], where Wk=Wk-1+tk-1 [2] is the waiting time for a kth process and ti is the execution time/length of next CPU burst of the ith process; 1<= k, i <=n (actually, the execution time of the last process in the queue, tn, does not affect any waiting times), and W0=0.

If we replace [2] into [1], we get: WS=((n-1)t1+(n-2)t2+...+(n-k)tk+... +tn-1)/n [3]

Now let us suppose that we have an arbitrary set of *n* CPU bursts, { t1, t2, ... , tn }.

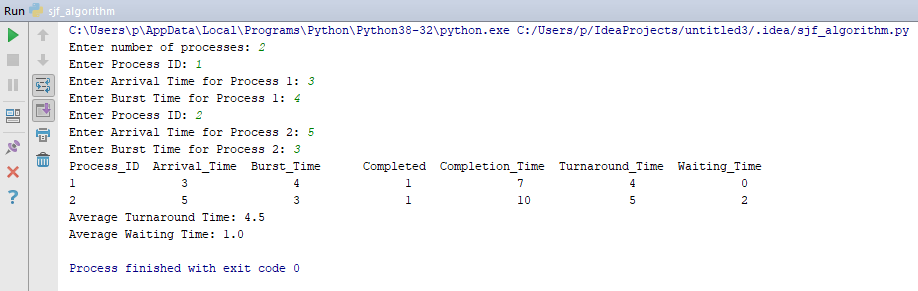


# Working Procedure:

Coding implementation with python…

*'''  
Created by asik  
date:02/09/2020  
  
'''***class** SJF:  
  
 **def** processData(self, no\_of\_processes):  
 process\_data = []  
 **for** i **in** range(no\_of\_processes):  
 temporary = []  
 process\_id = int(input(**"Enter Process ID: "**))  
  
 arrival\_time = int(input(**f"Enter Arrival Time for Process {process\_id}: "**))  
  
 burst\_time = int(input(**f"Enter Burst Time for Process {process\_id}: "**))  
 temporary.extend([process\_id, arrival\_time, burst\_time, 0])  
 **'''  
 '0' is the state of the process. 0 means not executed and 1 means execution complete  
 '''** process\_data.append(temporary)  
 SJF.schedulingProcess(self, process\_data)  
  
 **def** schedulingProcess(self, process\_data):  
 start\_time = []  
 exit\_time = []  
 s\_time = 0  
 process\_data.sort(key=**lambda** x: x[1])  
 **'''  
 Sort processes according to the Arrival Time  
 '''  
 for** i **in** range(len(process\_data)):  
 ready\_queue = []  
 temp = []  
 normal\_queue = []  
  
 **for** j **in** range(len(process\_data)):  
 **if** (process\_data[j][1] <= s\_time) **and** (process\_data[j][3] == 0):  
 temp.extend([process\_data[j][0], process\_data[j][1], process\_data[j][2]])  
 ready\_queue.append(temp)  
 temp = []  
 **elif** process\_data[j][3] == 0:  
 temp.extend([process\_data[j][0], process\_data[j][1], process\_data[j][2]])  
 normal\_queue.append(temp)  
 temp = []  
  
 **if** len(ready\_queue) != 0:  
 ready\_queue.sort(key=**lambda** x: x[2])  
 **'''  
 Sort the processes according to the Burst Time  
 '''** start\_time.append(s\_time)  
 s\_time = s\_time + ready\_queue[0][2]  
 e\_time = s\_time  
 exit\_time.append(e\_time)  
 **for** k **in** range(len(process\_data)):  
 **if** process\_data[k][0] == ready\_queue[0][0]:  
 **break** process\_data[k][3] = 1  
 process\_data[k].append(e\_time)  
  
 **elif** len(ready\_queue) == 0:  
 **if** s\_time < normal\_queue[0][1]:  
 s\_time = normal\_queue[0][1]  
 start\_time.append(s\_time)  
 s\_time = s\_time + normal\_queue[0][2]  
 e\_time = s\_time  
 exit\_time.append(e\_time)  
 **for** k **in** range(len(process\_data)):  
 **if** process\_data[k][0] == normal\_queue[0][0]:  
 **break** process\_data[k][3] = 1  
 process\_data[k].append(e\_time)  
  
 t\_time = SJF.calculateTurnaroundTime(self, process\_data)  
 w\_time = SJF.calculateWaitingTime(self, process\_data)  
 SJF.printData(self, process\_data, t\_time, w\_time)  
  
  
 **def** calculateTurnaroundTime(self, process\_data):  
 total\_turnaround\_time = 0  
 **for** i **in** range(len(process\_data)):  
 turnaround\_time = process\_data[i][4] - process\_data[i][1]  
 **'''  
 turnaround\_time = completion\_time - arrival\_time  
 '''** total\_turnaround\_time = total\_turnaround\_time + turnaround\_time  
 process\_data[i].append(turnaround\_time)  
 average\_turnaround\_time = total\_turnaround\_time / len(process\_data)  
 **'''  
 average\_turnaround\_time = total\_turnaround\_time / no\_of\_processes  
 '''  
 return** average\_turnaround\_time  
  
  
 **def** calculateWaitingTime(self, process\_data):  
 total\_waiting\_time = 0  
 **for** i **in** range(len(process\_data)):  
 waiting\_time = process\_data[i][5] - process\_data[i][2]  
 **'''  
 waiting\_time = turnaround\_time - burst\_time  
 '''** total\_waiting\_time = total\_waiting\_time + waiting\_time  
 process\_data[i].append(waiting\_time)  
 average\_waiting\_time = total\_waiting\_time / len(process\_data)  
 **'''  
 average\_waiting\_time = total\_waiting\_time / no\_of\_processes  
 '''  
 return** average\_waiting\_time  
  
  
 **def** printData(self, process\_data, average\_turnaround\_time, average\_waiting\_time):  
 process\_data.sort(key=**lambda** x: x[0])  
 **'''  
 Sort processes according to the Process ID  
 '''** print(**"Process\_ID Arrival\_Time Burst\_Time Completed Completion\_Time Turnaround\_Time Waiting\_Time"**)  
  
 **for** i **in** range(len(process\_data)):  
 **for** j **in** range(len(process\_data[i])):  
  
 print(process\_data[i][j], end=**" "**)  
 print()  
  
 print(**f'Average Turnaround Time: {average\_turnaround\_time}'**)  
  
 print(**f'Average Waiting Time: {average\_waiting\_time}'**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 no\_of\_processes = int(input(**"Enter number of processes: "**))  
 sjf = SJF()  
 sjf.processData(no\_of\_processes)

# Output:



# Discussion:

1.SJF method gives the lowest average waiting time for a specific set of processes.

2.It is appropriate for the jobs running in batch, where run times are known in advance.

3.For the batch system of long-term scheduling, a burst time estimate can be obtained from the job description.