Student Name: NIKESH. KUMAR. GUPTA.

Student ID: 11605312.

Email Address: nikeshgupta212@gmail.com

GitHub Link: https://github.com/NIKESH00/K1563_B40_NIKESHGUPTA

Code: Below is the mentioned code of the Shortest Remaining Time First Algorithm(SROT). Using preemptive scheduling technique which is also known as Shortest Job First scheduling Algorithm(SJF).

Program: SJF (Shortest Job First) Scheduling algorithm or Shortest Remaining Time First Algorithm(SROT).

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<malloc.h>
int n,*a;
struct pro
{
int name,at,bt,tt,wt,left,status;
}*p;
void create(int num)
{
a=(int*)malloc(num*sizeof(int));
}
int smallers(int num)
{
int c=0,i;
for(i=0;i<n;i++)
{
if((p+i)->bt<num && (p+i)->status==0)
```

```
{
*(a+c)=(p+i)->bt;
c++;
}
}
return c;
}
int search(int num)
{
int i;
for(i=0;i<n;i++)
{
if((p+i)->bt==num && (p+i)->status==0)
return i;
}
}
int main()
{
int i,c_time=0,b,s_nums,j,pro_num,p_time;
float att=0,awt=0;
printf("-----------------------------------------\n");
printf("-----------------------------------------");
printf("\n SHORTEST REMAINING TIME FIRST ALGORITHM(SROT)/SJF WITH PREEMPTIVE");
printf("\n-----------------------------------------");
printf("\nEnter number of processes to be allocated: ");
scanf("%d",&n);
create(n);
p=(struct pro *)malloc(n*sizeof(struct pro));
```

```
for(i=0;i<n;i++)
(p+i)->status=0;
printf("++++Enter Process NAME,++++++ Arrival Time & +++++++CPU Burst Time++++++\n");
for(i=0;i<n;i++)
scanf("%d %d %d",&(p+i)->name,&(p+i)->at,&(p+i)->bt);
for(i=0;i<n;i++)
{
if((p+i)->status==0)
{
b=(p+i)->bt;
s_nums=smallers(b);
if(s_nums>0)
{
(p+i)->left=(p+i)->bt;
for(j=0;j<s_nums;j++)
{
p_time=c_time;
pro_num=search(*(a+j));
if(c_time<(p+pro_num)->at)
{
if((p+i)->left>(p+pro_num)->at)
{
c_time=(p+pro_num)->at;
(p+i)->left=(p+i)->bt-(c_time-p_time);
c_time=c_time+(p+pro_num)->bt;
(p+pro_num)->status=1;
}
else
```

```
{
c_time=c_time+(p+i)->bt;
(p+i)->tt=c_time-(p+i)->at;
(p+i)->status=1;
if(c_time<(p+pro_num)->at)
break;
}
}
else
{
c_time=c_time+(p+pro_num)->bt;
}
(p+pro_num)->tt=c_time-(p+pro_num)->at;
(p+pro_num)->status=1;
}
if((p+i)->status==0)
{
c_time=c_time+(p+i)->left;
(p+i)->tt=c_time-(p+i)->at;
(p+i)->status=1;
}
}
else
{
c_time=c_time+(p+i)->bt;
(p+i)->tt=c_time-(p+i)->at;
(p+i)->status=1;
}
```

```
}
}
printf("\nProcess\t\tArrival Time\t\tCPU Burst Time\t\tTurnAround Time\t\tWaiting Time\n");
for(i=0;i<n;i++)
{
(p+i)->wt=(p+i)->tt-(p+i)->bt;
awt=awt+(p+i)->wt;
att=att+(p+i)->tt;
}
for(i=0;i<n;i++)
{
printf("%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",(p+i)->name,(p+i)->at,(p+i)->bt,(p+i)->tt,(p+i)->wt);
}
printf("\nAverage Turnaround Time :  %f\n",att/n);
printf("Average Waiting Time :  %f\n\n",awt/n);
return 0;
}
```

Description:

1. Program Description: SJF (Shortest Job First) Scheduling is the non-preemptive while in these program we will implement it in in preemptive version of SJF also known as the Shortest Remaining Time First(SRTF/SROT).

In these scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to be executed. While the currently executing process is one of the shortest amount of time remaining, and hence that the time should only be reduced as an execution progress. Where the progress will always run till they complete processes either the new process is added that is requiring a smaller amount of time.

Advantages of SJF: 1. Short processes are handled very quickly.

2. The system also requires very little time requirement since it only makes a decision when a process completes or the new process is added to it.

3. When the new process is added to the algorithm it only needs to compare it with the currently running/executing process with the new process, ignoring all the other running processes waiting to execute.

Disadvantages of SJF: 1. Similar to the SJF (Shortest Job First) Scheduling algorithm, is having the potential or the ability to process starvation.

2. As long as the process may have held off the indefinitely if the shortest process are continually added.

2. Algorithm of SJF: 1. Start the execution/ Starting of the Algorithm.

2.Enter the number of the processes you want to allocate.

3.Enter the Processes names which you like to set.

4.Enter the Arrival Time of the Processes.

5.Enter the Burst Time of the processes.

6.Finally after entering the input by the user the Completion time is calculated for each of the processes and with the help of it the Turn Around Time is calculated by using the formula (Comletion time-Arrival time).

7.Aswellas the Waiting Time is also calculated with the help of the Turn Around Time by using the formula as (Turn Around Time - Burst Time) of the CPU.

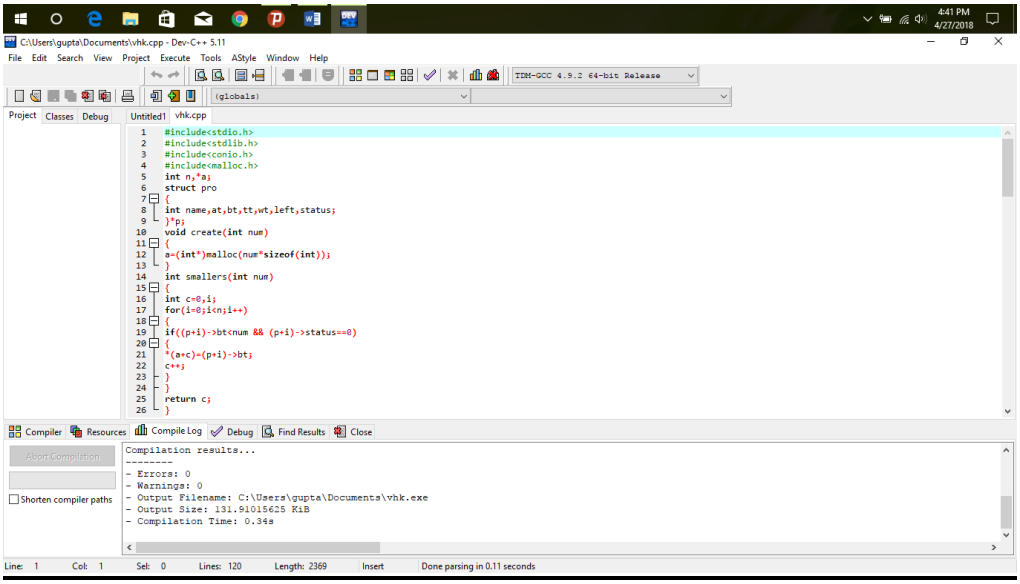
8.After all these internal Calculation the Average Waiting Time is Calculated by adding all the waiting time values of the processes and dividing it by the number of the processes.

9.At last the Average Turn Around Time is calculated by adding all the values obtained by the Turn Around Time divided by the number of the processes.

10.Displaying all the processes along with their Gantt Chart.

11.At the end Termination of the program.

Code Snippet:

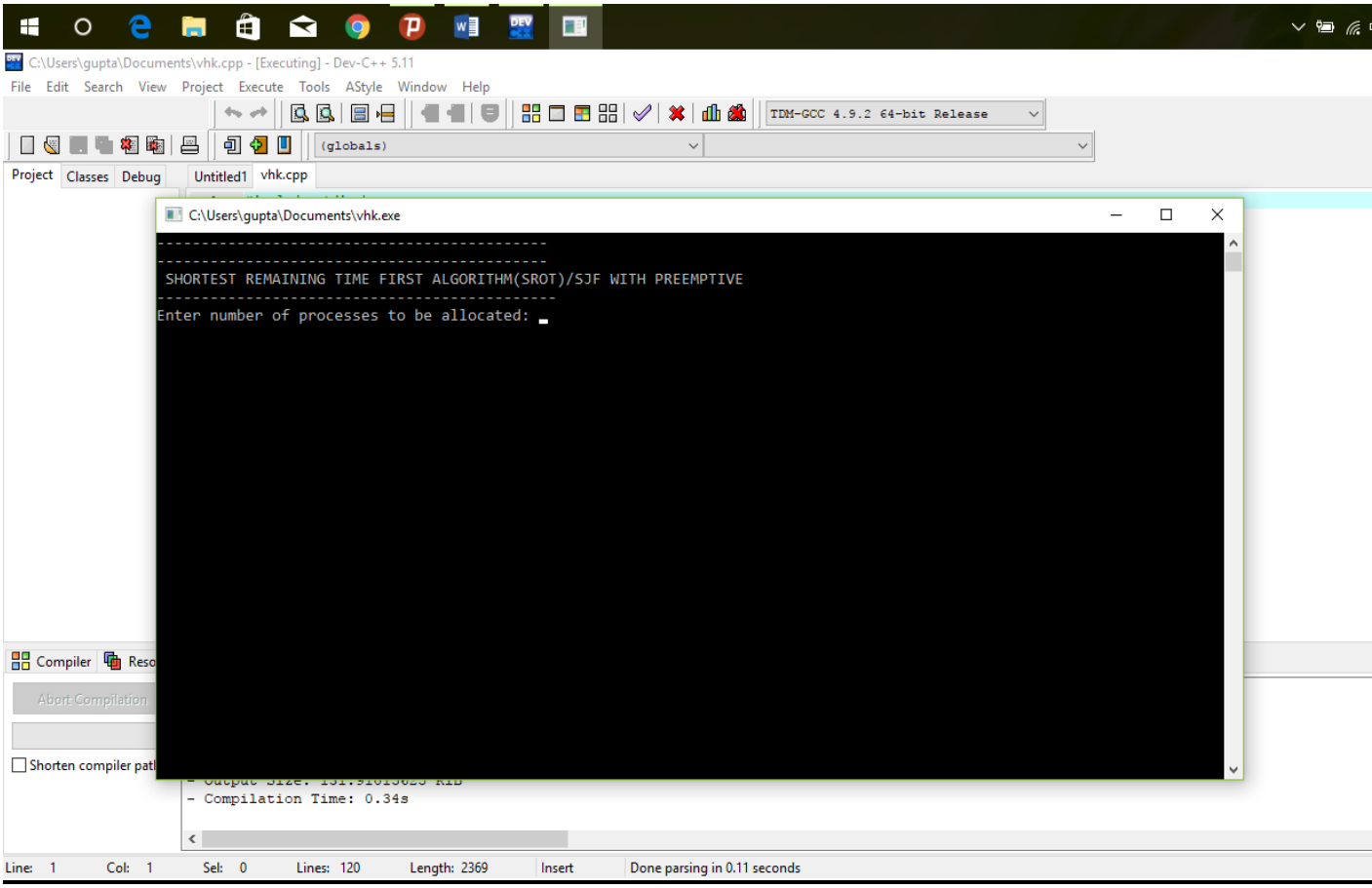


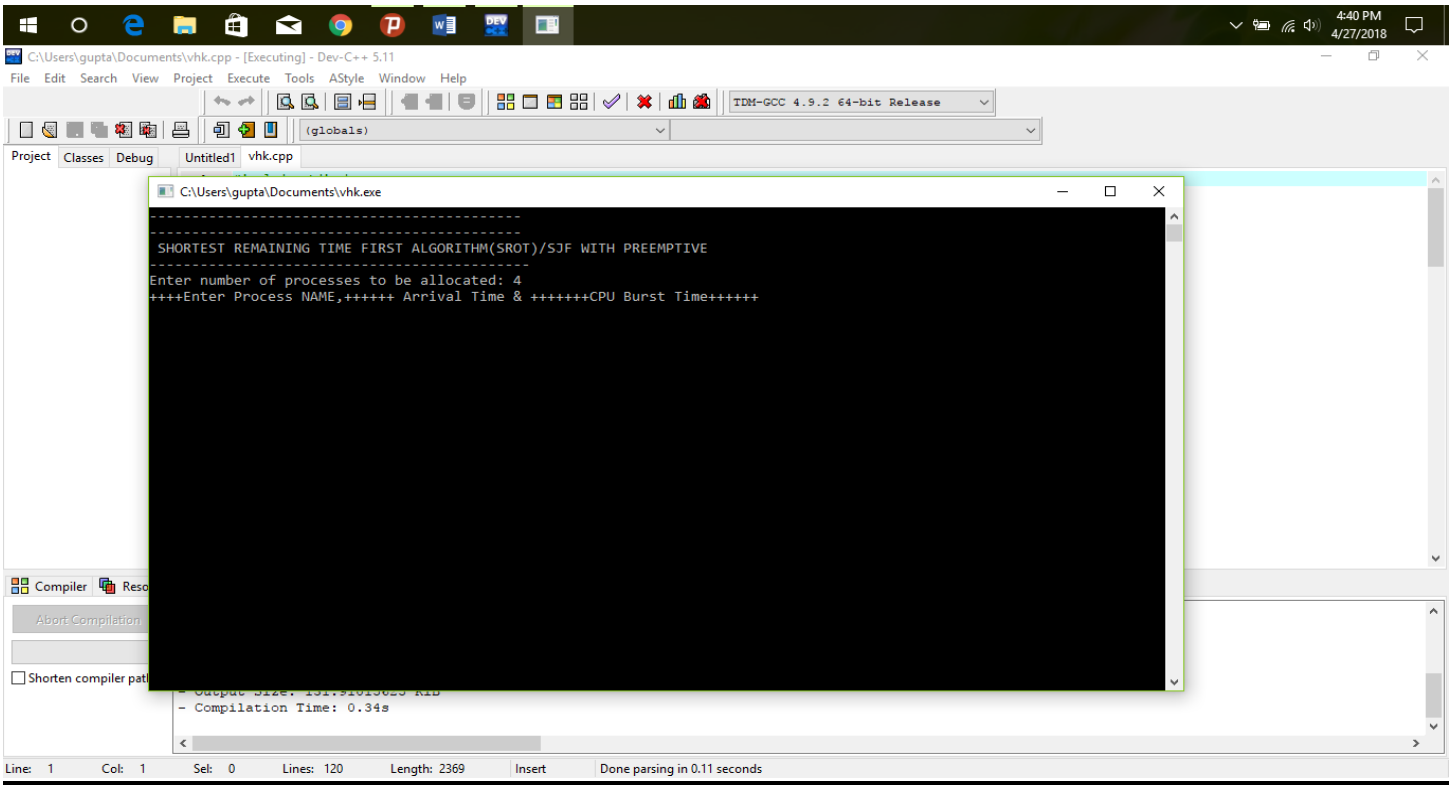
C:\Users\gupta\Documents\vhk.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Untitled1 vhk.cpp

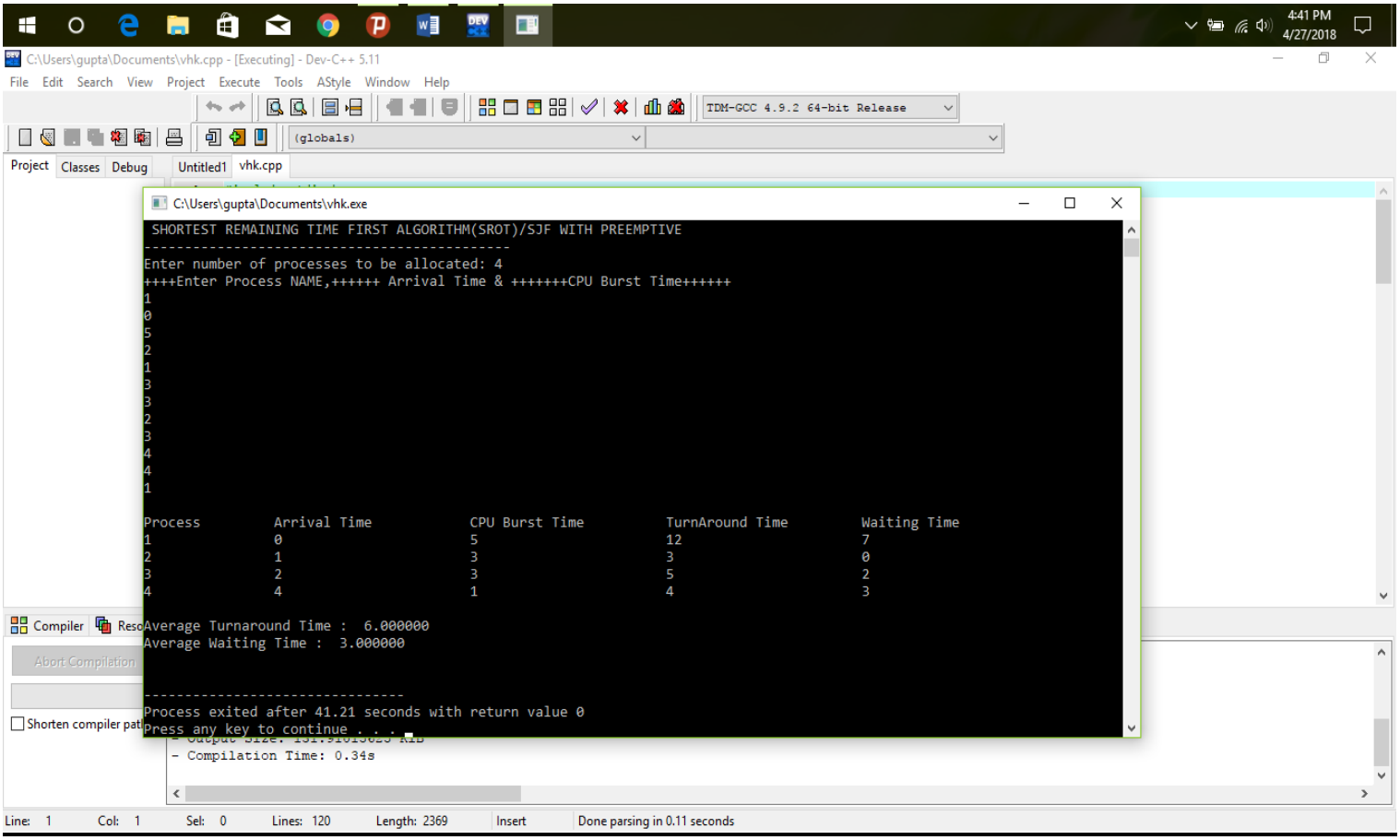
```
(p+i)->status=1;
}
}
else
{
c_time=c_time+(p+i)->bt;
(p+i)->tt=c_time-(p+i)->at;
(p+i)->status=1;
}
}
}
printf("\nProcess\t\tArrival Time\t\tCPU Burst Time\t\tTurnAround Time\t\tWaiting Time\n");
for(i=0;i<n;i++)
{
(p+i)->wt=(p+i)->tt-(p+i)->bt;
awt=awt+(p+i)->wt;
att=att+(p+i)->tt;
}
for(i=0;i<n;i++)
{
printf("%d\t\t%d\t\t\t\t%d\t\t\t\t%d\t\t\t\t%d\n",(p+i)->name,(p+i)->at,(p+i)->bt,(p+i)->tt,(p+i)->wt);
}
printf("\nAverage Turnaround Time :  %f\n",att/n);
printf("Average Waiting Time :  %f\n\n",awt/n);
return 0;
}
```

Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\gupta\Documents\vhk.exe
- Output Size: 131.91015625 KiB
- Compilation Time: 0.34s
Line: 1 Col: 1 Sel: 0 Lines: 120 Length: 2369 Insert Done parsing in 0.11 seconds


C:\Users\gupta\Documents\vhk.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Untitled1 vhk.cpp
C:\Users\gupta\Documents\vhk.exe
SHORTEST REMAINING TIME FIRST ALGORITHM(SROT)/SJF WITH PREEMPTIVE
Enter number of processes to be allocated:
Compiler
Abort Compilation
Shorten compiler path
- Compilation Time: 0.34s
Line: 1 Col: 1 Sel: 0 Lines: 120 Length: 2369 Insert Done parsing in 0.11 seconds


4:40 PM
4/27/2018
C:\Users\gupta\Documents\vhk.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Untitled1 vhk.cpp
C:\Users\gupta\Documents\vhk.exe
SHORTEST REMAINING TIME FIRST ALGORITHM(SROT)/SJF WITH PREEMPTIVE
Enter number of processes to be allocated: 4
++++Enter Process NAME,++++++ Arrival Time & +++++++CPU Burst Time++++++
Compiler
Abort Compilation
Shorten compiler path
- Compilation Time: 0.34s
Line: 1 Col: 1 Sel: 0 Lines: 120 Length: 2369 Insert Done parsing in 0.11 seconds



3. Purpose of Use: It is used to calculate the Average waiting Time of the CPU processes and the Average Turn Around Time of the processes. And it tells how the work is done for the processes in the CPU.

4. Test Cases Applied Are: - output: Shortest Job First for preemptive is as follows: -

Enter the Number of processes you want to allocate: 4

Enter the Arrival Time of the process: p1:0 p2:1 p3:2 p4:4

Enter the Burst Time of the Processes: p1:5 p2:3 p3:3 p4:1

waiting Time (Turn Around Time - Burst Time) : 7:p1 1:p2 3:p3 1:p4

Turn Around Time (Completion Time - Arrival Time): 12:p1 4:p2 6:p3 2:p4

Average Waiting Time for the above output is: 3.00

Average Turn Around Time for the above output is: 6.00

5.Boundary Condition: The boundary condition of the code is that it cannot perform the following operations: -

1. Similar to the SJF (Shortest Job First) Scheduling algorithm is having the potential or the ability to process starvation.
2. As long as the process may have held off the indefinitely if the shortest process are continually added

6.Number of Revisions made to the code:

More than 5-times revision is made to the code in the GitHub account

Yes, I have made the 5 revisions of the solutions in the provided GitHub site.

GitHub Link: - https://github.com/NIKESH00/K1563_B40_NIKESHGUPTA