**Experiment No. 14**

**Experiment Name:** Implementation of SJF preemptive scheduling algorithm .

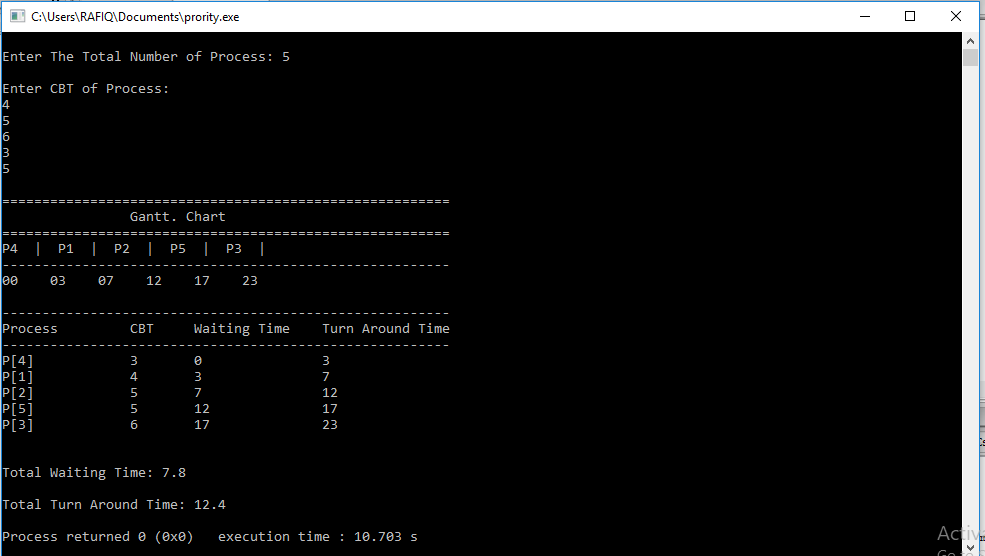
**Aim and Objectives:**

* What is SJF preemptive scheduling algorithm ..
* How to implementation this algorithm

**Source Code:**

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  int main()  {  int p, i, j, sum=0, min, index;  float awt=0, atat=0;  cout<<"\nEnter The Total Number of Process: ";  cin>>p;  int proc[p];  int \*cbt = new int[p];  int \*wt = new int[p];  int \*gc = new int[p];  int \*tat = new int[p];  int \*tmp = new int[p];  cout<<"\nEnter CBT of Process:\n";  for(i=0; i<p; i++)  { cin>>cbt[i];  tmp[i]=cbt[i];  }  sort(cbt, cbt+p);  cout<<"\n========================================================\n";  cout<<"\t\tGantt. Chart";  cout<<"\n========================================================\n";  for(j=0; j<=p; j++)  {  min=100;  for(i=0; i<p; i++)  {  if(min>tmp[i]&&tmp[i]!=-1)  {  min=tmp[i];  index=i;  }  }  gc[j]=sum;  wt[j]=sum;  sum+=tmp[index];  tat[j]=sum;  tmp[index]=-1;  if(j==p)  break;  cout<<'P'<<index+1<<" | ";  proc[j]=index+1;  }  cout<<"\n--------------------------------------------------------\n";  sum=0;  for(j=0; j<=p; j++)  {  if(gc[j]<10)  cout<<0;  cout<<gc[j]<<" ";  sum+=gc[j];  }  cout<<endl;  atat=(sum\*1.0)/p;  cout<<"\n--------------------------------------------------------";  cout<<"\nProcess\t\tCBT\tWaiting Time\tTurn Around Time";  cout<<"\n--------------------------------------------------------\n";  for(i=0; i<p; i++)  {  cout<<"P["<<proc[i]<<"]\t\t"<<cbt[i]<<"\t"<<wt[i]<<"\t\t"<<tat[i]<<endl;  awt=awt+wt[i];  }  awt=(awt\*1.0)/p;  cout<<"\n\nTotal Waiting Time: "<<awt;  cout<<"\n\nTotal Turn Around Time: "<<atat<<endl;  return 0;  } |

**Output:**



**Conclusion:** The process which is currently in execution, runs until it complete  or a new process is added in the CPU Scheduler that requires smaller amount of  time for execution. It is also known as shortest remaining time first(SRTF).  
Unlike [round robin scheduling algorithm](http://javahungry.blogspot.com/2013/09/round-robin-scheduling-algorithm-with-example-java-program-code.html)  , shortest remaining time scheduling algorithm may lead to starvation . If the short processes are continually added to the **CPU** scheduler then the currently running process will never be able to execute , hence SRT is not starvation free .