**Pre-emptive Priority Scheduling**

**EXPT NO: 3 DATE: 7/10/22**

**AIM:**

To Implement pre-emptive priority scheduling algorithm.

**THEORY**

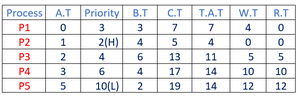
In priority scheduling the process that has highest priority gets the CPU first. The processes get serviced by the CPU in order of their priority in descending order. Higher number always doesn’t represent higher priority. In some cases, 0 may be the highest priority and 100 the lowest. It depends on systems.

Pre-emptive Priority Scheduling Algorithm is a CPU Scheduling algorithm in which at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

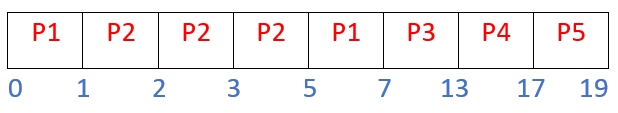
The difference between pre-emptive priority scheduling and non-pre-emptive priority scheduling is that, in the pre-emptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-pre-emptive priority scheduling, which means the job scheduled will run till the completion and no pre-emption will be done.

Pre-emptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.



Gantt Chart



Avg TAT=10         Avg WT=6.2

**SOURCE CODE**

**#include<iostream>**

**#include<algorithm>**

**#include<vector>**

**#define MAX 10**

**using namespace std;**

**int N,TOTbt=0;**

**typedef struct PROCESSES{**

**int index;**

**int BT;**

**int AT;**

**int PRIO;**

**int TAT;**

**int WT;**

**int CT;**

**int CountBT;**

**};**

**typedef struct ARRANGE{vector<PROCESSES> P;};**

**ARRANGE ALIST[50];**

**void processInput(PROCESSES Table[])**

**{**

**cout<<"ENTER NUMBER OF ELEMENTS: ";**

**cin>>N;**

**cout<<"ENTER THE PROCESS DETAILS"<<endl;**

**for(int i=0;i<N;i++)**

**{**

**cout<<"P"<<i+1<<": \n";**

**Table[i].index=i+1;**

**cout<<"ENTER BURST TIME: ";**

**cin>>Table[i].BT; TOTbt+=Table[i].BT;**

**cout<<"ENTER ARRIVAL TIME: ";**

**cin>>Table[i].AT;**

**cout<<"ENTER PRIORITY: ";**

**cin>>Table[i].PRIO;**

**cout<<endl;**

**Table[i].CountBT=0;**

**}**

**}**

**bool CRITERIA2(PROCESSES A,PROCESSES B) {return(A.index<B.index);}**

**void ProcessInfo(PROCESSES Table[])**

**{**

**int TotalTAT=0,TotalWT=0;**

**sort(Table,Table+N,CRITERIA2);**

**cout<<"\n\nPROCESS "<<"BT "<<"AT "<<"CT "<<"PRIORITY "<<"TAT "<<"WT "<<endl;**

**for(int i=0;i<N;i++)**

**{**

**cout.setf(ios::left,ios::adjustfield);**

**cout.width(12);**

**cout<<Table[i].index;**

**cout.width(7);**

**cout<<Table[i].BT;**

**cout.width(7);**

**cout<<Table[i].AT;**

**cout.width(7);**

**cout<<Table[i].CT;**

**cout.width(11);**

**cout<<Table[i].PRIO;**

**cout.width(8);**

**cout<<Table[i].TAT;**

**TotalTAT+=Table[i].TAT;**

**cout.width(7);**

**cout<<Table[i].WT;**

**TotalWT+=Table[i].WT;**

**cout<<endl;**

**}**

**cout<<"\nATAT ="<<TotalTAT/N;**

**cout<<"\nAWT ="<<TotalWT/N<<endl;**

**}**

**bool CRITERIA1(PROCESSES A,PROCESSES B) { return(A.PRIO<B.PRIO); }**

**void processArrange(PROCESSES Table[])**

**{**

**for(int i=0;i<N;i++)**

**{**

**ALIST[Table[i].AT].P.push\_back(Table[i]);**

**}**

**}**

**void PROCESS\_execute(int i,vector<PROCESSES>& WaitQ)**

**{**

**cout<<"| P"<<WaitQ[0].index<<" ";**

**WaitQ[0].CountBT++;**

**}**

**void PRIORITY\_SCHEDULING(PROCESSES Table[])**

**{**

**vector<PROCESSES>WaitQ;**

**vector<PROCESSES>::iterator QT;**

**int i;**

**cout<<endl<<"GHANTT CHART\n";**

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

**{**

**if(ALIST[i].P.empty()==false)**

**{**

**for(QT=ALIST[i].P.begin();QT!=ALIST[i].P.end();++QT)**

**{**

**WaitQ.push\_back(\*QT);**

**}**

**sort(WaitQ.begin(),WaitQ.end(),CRITERIA1);**

**}**

**PROCESS\_execute(i,WaitQ);**

**if(WaitQ[0].CountBT==WaitQ[0].BT)**

**{**

**Table[WaitQ[0].index-1].CT=i+1;**

**Table[WaitQ[0].index-1].TAT=(Table[WaitQ[0].index-1].CT) - (Table[WaitQ[0].index-1].AT);**

**Table[WaitQ[0].index-1].WT=(Table[WaitQ[0].index-1].TAT) - (Table[WaitQ[0].index-1].BT);**

**WaitQ.erase(WaitQ.begin()); //If process BT done then delete from waiting queue**

**}**

**}**

**cout<<"|"<<endl;**

**for(int j=0;j<=TOTbt;j++)**

**{**

**cout.setf(ios::left,ios::adjustfield);**

**cout.width(5);**

**cout<<j;**

**}**

**}**

**int main()**

**{**

**PROCESSES Table[MAX];**

**processInput(Table);**

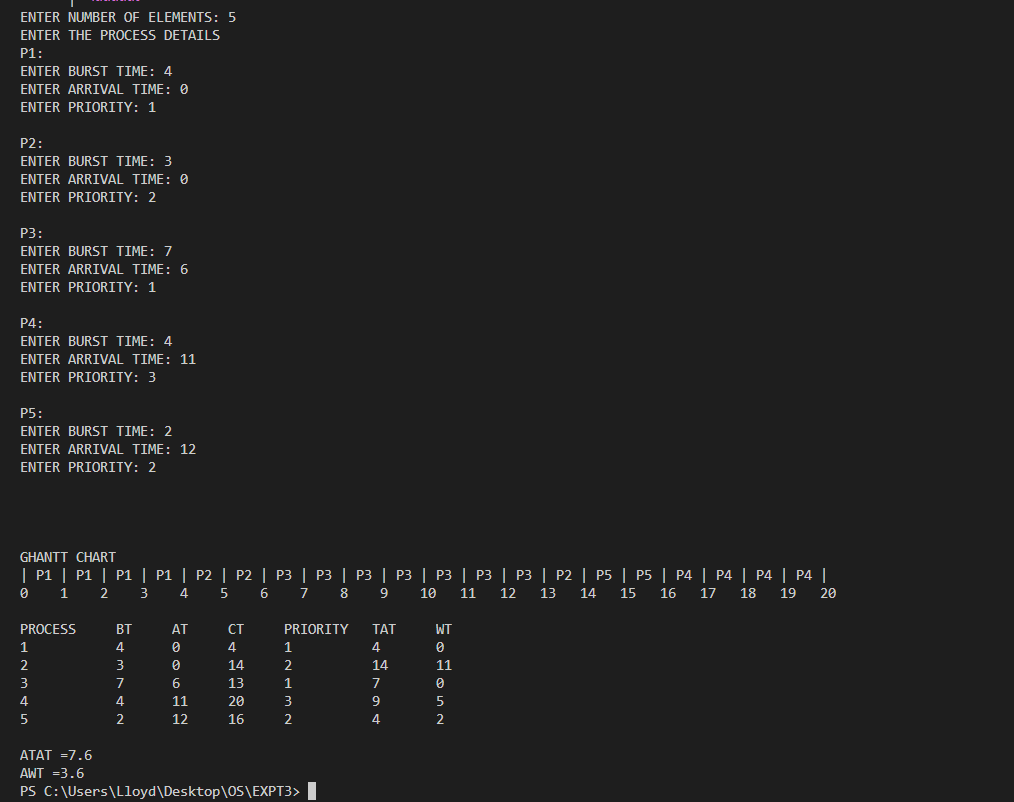
**processArrange(Table);**

**cout<<endl<<endl;**

**PRIORITY\_SCHEDULING(Table);**

**ProcessInfo(Table);**

**}**

**OUTPUT**

**CONCLUSION**

The pre-emptive priority scheduling algorithm was comprehended and was successfully implemented.