**Student Name: Anil Singh**

**Student ID: 11803558**

**Roll No.: K18BY22**

**Ques. No. Assigned: 6**

**Email Address: singhanil2716@gmail.com**

**GitHub Link:** <https://github.com/Anil123singh/operating-system>

1. **Explain the problem in terms of operating system concept?**

**Ans :** In the given problem of 6 Each process is assigned a numerical priority, with a

higher number indicating a higher relative priority and this problem is used a premtive priority scheduling to indicating a higher priority and round robin algorithm is used to each higher priority is use to implement in cyclic way and assign a each process in fixed execution time . In this problem, the scheduler selects the tasks to work as per the priority.Priority depends upon memory requirements, time requirements, etc.In Preemptive Scheduling, the tasks are mostly assigned with their priorities. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.And in these process is used a round robin algorithm is used to each process use to implement in cyclic way, which is mostly used to multitasking.In the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P\_idle). This task has priority 0 and is scheduled whenever the system has no other available processes to run.

According to priority Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

**2.Write the algorithm for proposed solution of the assigned problem.**

**Ans In the given problem to solve with priority scheduling and round robin algorithm to implement.**

**/\*Priority Scheduling** algorithm is a method of scheduling processes that is based on priority. In this problem the scheduler selects the tasks to work as per the priority.\*/

**#include<thread.h>**

**#include<stdio.h>**

**#include<unistd.h>**

**#include<sys/wait.h>**

**#include<stdlib.h>**

**#define NEW 0**

**#define READY 1**

**#define WAITING 2**

**#define RUNNING 3**

**#define EXIT 4**

**int currenttime=0,runningtime=0,ALLEXIT=0,TimeQuantum,CT=1,n;**

**float totalext=0;**

**struct Proc**

**{**

**int pid;**

**int state;**

**int timeleft;**

**int priority;**

**int at;**

**int wt,tat,ct,exect;**

**struct Proc \*prev;**

**struct Proc \*next;**

**} \*pa;**

**int chpp;**

**struct Queue**

**{**

**struct Proc \*front ,\*rear;**

**}\*ReadyQueue;**

**void enqueue\_p(struct Proc \*p)**

**{**

**if(ReadyQueue->front==NULL)**

**{**

**ReadyQueue->front=p;**

**ReadyQueue->rear=p;**

**p->next=NULL;**

**}else**

**{ if(p->priority>ReadyQueue->front->priority)**

**{ p->next=ReadyQueue->front;**

**ReadyQueue->front->prev=p;**

**ReadyQueue->front=p; }**

**else if(p->priority==ReadyQueue->front->priority)**

**{ p->next=ReadyQueue->front->next;**

**p->prev=ReadyQueue->front;**

**ReadyQueue->front->next=p;**

**if(p->next!=NULL){**

**p->next->prev=p; } }**

**else if(p->priority<ReadyQueue->rear->priority)**

**{ p->next=NULL;**

**ReadyQueue->rear->next=p;**

**p->prev=ReadyQueue->rear;**

**ReadyQueue->rear=p; }**

**else**

**{ struct Proc \*start=ReadyQueue->front->next;**

**while(start->priority>p->priority)**

**{ start=start->next; }**

**if(start!=NULL&& p->priority==start->priority)**

**{ p->next=start->next;**

**start->next=p;**

**p->prev=start**

**} else**

**{ (start->prev)->next=p;**

**p->next=start;**

**p->prev=start->prev;**

**start->prev=p; } } } }**

**void enqueue(struct Proc \*p)**

**{ if(ReadyQueue->front==NULL)**

**{ p->prev=NULL;**

**p->next=NULL;**

**ReadyQueue->front=p;**

**ReadyQueue->rear=p;**

**return; }**

**p->prev=ReadyQueue->rear;**

**ReadyQueue->rear->next=p;**

**ReadyQueue->rear=p;**

**}**

**struct Proc \* deQueue()**

**{ if(ReadyQueue->front==NULL)**

**{ return NULL; }**

**struct Proc \* temp=ReadyQueue->front;**

**ReadyQueue->front=ReadyQueue->front->next;**

**temp->next=NULL;**

**if(ReadyQueue->front==NULL)**

**{ ReadyQueue->rear=NULL; }**

**return temp; }**

**void checkqueue()**

**{**

**int count=0;**

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

**{ if(pa[i].state == NEW && currenttime>=pa[i].at)**

**{ enqueue\_p(&pa[i]);**

**pa[i].state=READY; }**

**if(pa[i].state==EXIT)**

**{ count++ ; } }**

**if(count==n)**

**{ ALLEXIT=1; }**

**}**

**int main()**

**{ ReadyQueue =(struct Queue\*) malloc(sizeof(struct Queue));**

**printf("Please enter No of processes to schedule");**

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

**printf("\n Please Enter Time qunatum");**

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

**pa=(struct Proc \*)malloc(sizeof(struct Proc)\*n);**

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

**{ printf("\n\n Enter Process Id For %d Process",(i+1));**

**scanf("%d",&(pa[i].pid));**

**printf("\n Enter arrival time For %d Process",(i+1));**

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

**printf("\n Enter Execution time For %d Process",(i+1));**

**scanf("%d",&(pa[i].timeleft));**

**printf("\n Enter Priority for %d Process",(i+1));**

**scanf("%d",&(pa[i].priority));**

**pa[i].exect=pa[i].timeleft;**

**totalext+=pa[i].timeleft;**

**pa[i].state=NEW; }**

**struct Proc key;**

**int i,j;**

**for(i = 1; i < n; i++)**

**{ key = pa[i];**

**j = i - 1;**

**while (j >= 0 && pa[j].at > key.at) {**

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

**j = j - 1; }**

**pa[j + 1] = key;**

**}**

**struct Proc \*pr;**

**printf("Gannt Chart\n====================================================================================================================\n");**

/\* round robin algorithm is used to given a fixed excecute time in time quantum with help of higher priority process. If the no process of the system also has an idle task (which consumes no CPU resources and is identified as P\_idle)\*/

**while(1)**

**{ checkqueue();**

**if(ALLEXIT==1)**

**{ break; }**

**if(ReadyQueue->front!=NULL && CT==1)**

**{ runningtime=1;**

**pr=deQueue();**

**printf(" %d| Process Id:%d |",currenttime,pr->pid);**

**pr->state=RUNNING;**

**pr->timeleft--;**

**currenttime++;**

**if(runningtime==pr->exect)**

**{**

**CT=1;**

**pr->state=EXIT;**

**pr->ct=currenttime;**

**pr->tat=currenttime-pr->at;**

**pr->wt=pr->tat-pr->exect;**

**}**

**else**

**{**

**CT=0;**

**}**

**}**

**else if(CT==0&&pr!=NULL && pr->state==RUNNING )**

**{**

**if(pr->timeleft==0)**

**{**

**CT=1;**

**pr->state=EXIT;**

**pr->ct=currenttime;**

**pr->tat=currenttime-(pr->at);**

**pr->wt=(pr->tat)-(pr->exect);**

**continue;**

**}**

**else if(runningtime==TimeQuantum&&ReadyQueue->front !=NULL)**

**if(pr->priority>ReadyQueue->front->priority)**

**{**

**pr->state=READY;**

**enqueue(pr);**

**CT=1;**

**continue;**

**}**

**}**

**runningtime++;**

**pr->timeleft--;**

**currenttime++;**

**if(pr->timeleft==0)**

**{**

**CT=1;**

**pr->state=EXIT;**

**pr->ct=currenttime;**

**pr->tat=currenttime-(pr->at);**

**pr->wt=(pr->tat)-(pr->exect);**

**continue;**

**}**

**else if(runningtime==Time Quantum&&ReadyQueue->front !=NULL)**

**{ if(pr->priority>ReadyQueue->front->priority)**

**{**

**pr->state=READY;**

**enqueue(pr);**

**CT=1;**

**}**

**else**

**{**

**CT=0;**

**}**

**}**

**else**

**{**

**CT=0;**

**}**

**}**

**else**

**{**

**printf(" %d| Process Id: 0|",currenttime);**

**currenttime++;**

**}**

**}**

**}printf("%d|\n",currenttime);**

**printf("============================================================================================================\n");**

**int sumwt=0,sumtat=0;**

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

**{**

**printf("\n\nprocess pid=%d\nct=%d\ntat=%d\nwt=%d",pa[i].pid,pa[i].ct,pa[i].tat,pa[i].wt);**

**sumwt+=pa[i].wt;**

**sumtat+=pa[i].tat;**

**}**

**printf("\n\n Avergae TAT=%f \n Average WT=%f\n CPU Utilization=%f\n",(sumtat/(n\*1.0)),(sumwt/(n\*1.0)),(totalext/(currenttime\*1.0)));**

**}**

**2.Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity).**

Ans **Worst and Average Case Time Complexity:**O(n\*n). Worst case occurs when arrival time is reverse sorted.

**Best Case Time Complexity:** O(n). Best case occurs when arrival time is already sorted.

For a priority scheduling algo we need to find the element of highest priority.This problem is reduced to a search problem of an unsorted queue.

The worst case complexity of such a problem is O(n).

In the round robin algorithm worst case complexity is O(n\*burst time).  
In the round robin algorithm worst case complexity is O(n\*burst time)

**4. Explain all the constraints given in the problem. Attach the code snippet of the**

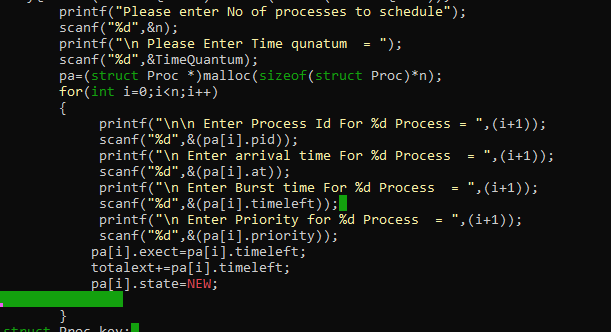
**implemented constraint.**

In this problems, we are given constraint of n process,I.e 1<=process<n

Whereas I applied a constraint of priority of process, i.e. 1 <= priority <150

Whereas I applied a constraint of arrival time of process, i.e. 0 <= arrival <150

Whereas I applied a constraint of Burst time of process, i.e. 1 <= priority <100



**5. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.**

**(Bubble Sort)**

Bubble sort is (provably) the fastest sort available under a very specific circumstance. It originally became well known primarily because it was one of the first algorithms (of any kind) that was rigorously analyzed, and the proof was found that it was optimal under its limited circumstance.

I have used **Bubble Sort** algorithm in Problem to sort the processes in descending order based on the arrival time of processes and updated there priority and sorted the order of process based on there current priority.

**struct Proc key;**

**int i,j;**

**for (i = 1; i < n; i++) {**

**key = pa[i];**

**j = i - 1;**

**while (j >= 0 && pa[j].at > key.at) {**

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

**j = j - 1;**

**}**

**pa[j + 1] = key;**

**}**

1. **Explain the boundary conditions of the implemented code.**

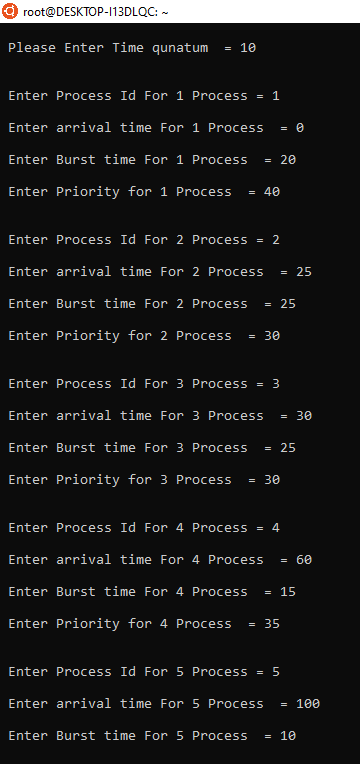
**Boundary condition:-**

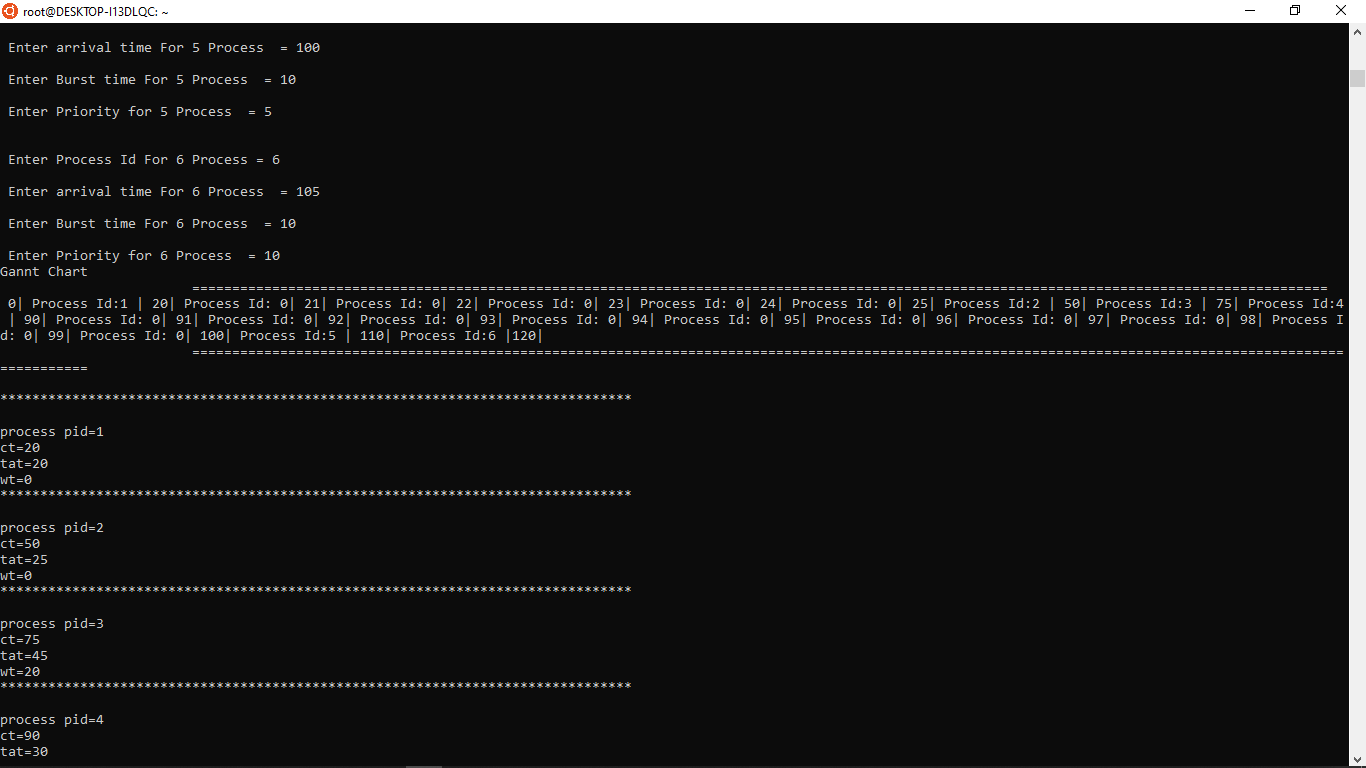
Bubble sort takes minimum time (Order of n) when elements are already sorted.

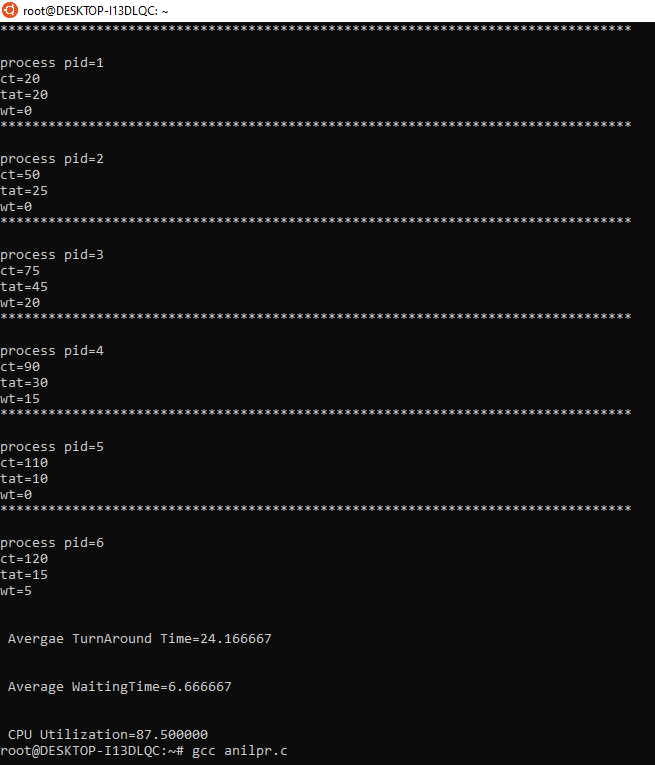
1. **Explain all the test cases applied on the solution of assigned problem**

This problem doesn’t require any additional test case. We’ll get a different output everytime when we execute the code of this problem. The screenshots of output are attached below

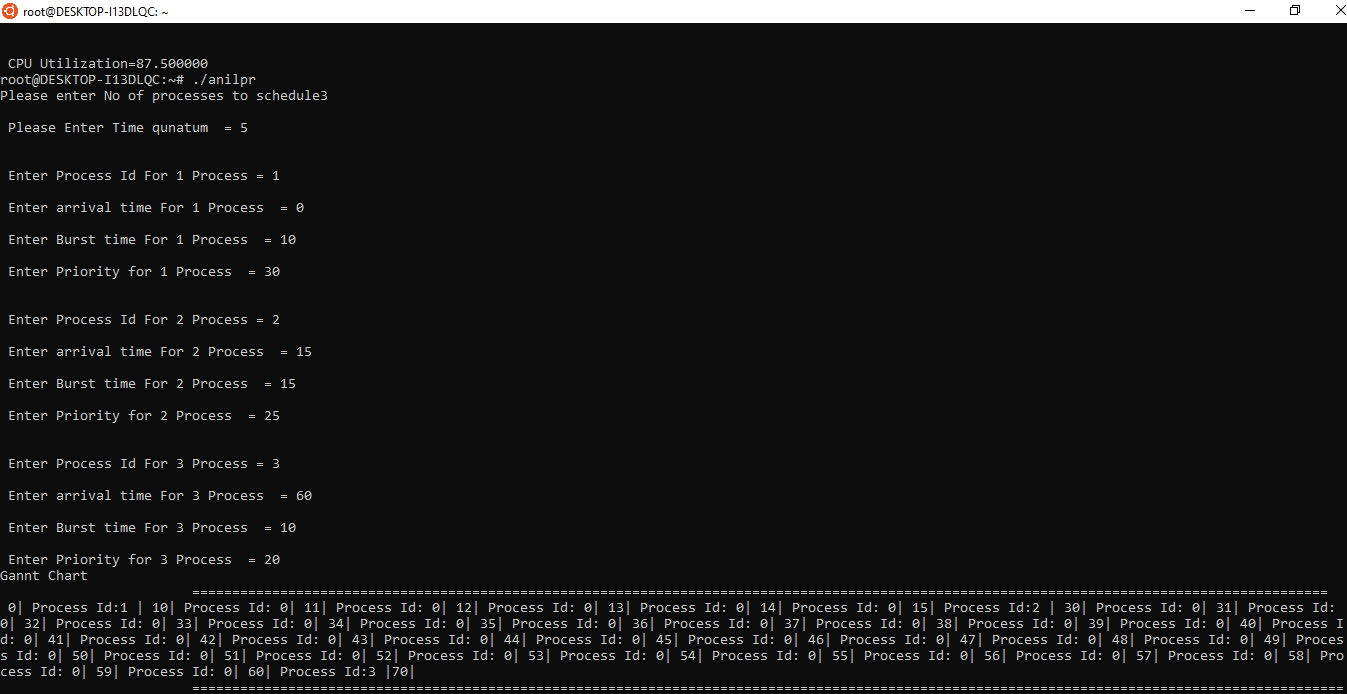
**EXECUTION 1:=**

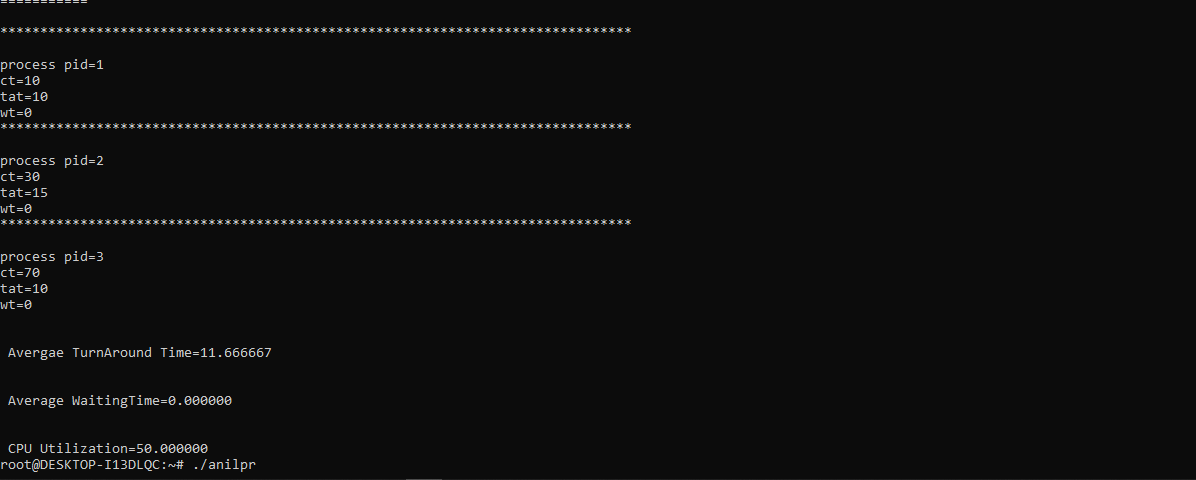






**EXECUTION 2 :=**

****

****

**8. Have you made minimum 5 revisions of solution on GitHub?**

**GitHub Link:** <https://github.com/Anil123singh/operating-system>