Name:V Narender Reddy

Sec:k1622

Group:2

Roll.No:51

Q.NO:15……………………

#include<stdio.h>

typedef struct{

int process\_id, arrival\_time, burst\_time, priority;

int q, ready;

int queue;

}process\_structure;

void findWaitingTime(int processes[], int n,int bt[], int wt[], int quantum){

int rem\_bt[n],i;

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

rem\_bt[i] = bt[i];

int t = 0;

while (1)

{

int done = 1;

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

{

if (rem\_bt[i] > 0)

{

done = 0;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == 1)

break;

}

}

void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[]){

int i;

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

tat[i] = bt[i] + wt[i];

}

void findavgTime(int processes[], int n, int bt[],int quantum){

int wt[n], tat[n], total\_wt = 0, total\_tat = 0,i;

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst time Waiting time Turn around time\n");

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d\t\t%d\t\t%d\t\t%d\n", i+1 ,bt[i] ,wt[i],tat[i]);

}

printf("Average waiting time = %f\nAverage turn around time =%f" ,(float)total\_wt / (float)n,(float)total\_tat / (float)n);

}

int Queue(int t1){

if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3){

return 1;

}

else{

return 2;

}

}

void RoundRobin(process\_structure process[],int processes[],int size){

int BurstTime[size],pid[size];

int quantum,i;

printf("Enter time quantum:");

scanf("%d",&quantum);

for(i=0;i<size;i++){

pid[i] = process[processes[i]].process\_id;

BurstTime[i] = process[processes[i]].burst\_time;

}

findavgTime(pid,size,BurstTime,quantum);

}

int main(){

int limit, count, temp\_process, time, j, y,ch;

process\_structure temp;

printf("Enter Total Number of Processes:\t");

scanf("%d", &limit);

int RoundQueue[limit],RoundSize=0;

int PriorityQueue[limit],PrioritySize=0;

process\_structure process[limit];

for(count = 0; count < limit; count++){

printf("\nProcess ID:\t");

scanf("%d", &process[count].process\_id);

printf("Arrival Time:\t");

scanf("%d", &process[count].arrival\_time);

printf("Burst Time:\t");

scanf("%d", &process[count].burst\_time);

printf("Process Belongs to which Queue:\n1.Priority Queue\n2.Round Robin\n");

scanf("%d",&process[count].queue);

if(process[count].queue == 1){

printf("Process Priority:\t");

scanf("%d", &process[count].priority);

temp\_process = process[count].priority;

process[count].q = Queue(temp\_process);

process[count].ready = 0;

PriorityQueue[PrioritySize++] = count;

}

else{

RoundQueue[RoundSize++] = count;

}

}

time = process[PriorityQueue[0]].burst\_time;

for(y = 0; y < PrioritySize; y++){

for(count = y; count < PrioritySize; count++){

if(process[PriorityQueue[count]].arrival\_time < time) {

process[PriorityQueue[count]].ready = 1;

}

}

for(count = y; count < PrioritySize - 1; count++){

for(j = count + 1; j < PrioritySize; j++){

if(process[PriorityQueue[count]].ready == 1 && process[PriorityQueue[j]].ready == 1){

if(process[PriorityQueue[count]].q == 2 && process[PriorityQueue[j]].q == 1){

temp = process[PriorityQueue[count]];

process[PriorityQueue[count]] = process[PriorityQueue[j]];

process[PriorityQueue[j]] = temp;

}

}

}

}

for(count = y; count < PrioritySize - 1; count++){

for(j = count + 1; j < PrioritySize; j++)

{

if(process[PriorityQueue[count]].ready == 1 && process[PriorityQueue[j]].ready == 1)

{

if(process[PriorityQueue[count]].q == 1 && process[PriorityQueue[j]].q == 1)

{

if(process[PriorityQueue[count]].burst\_time > process[PriorityQueue[j]].burst\_time)

{

temp = process[PriorityQueue[count]];

process[PriorityQueue[count]] = process[PriorityQueue[j]];

process[PriorityQueue[j]] = temp;

}

else{

break;

}

}

}

}

}

printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[PriorityQueue[y]].process\_id, time, time + process[PriorityQueue[y]].burst\_time);

time = time + process[PriorityQueue[y]].burst\_time;

for(count = y; count < PrioritySize; count++){

if(process[PriorityQueue[count]].ready == 1){

process[PriorityQueue[count]].ready = 0;

}

}

}

RoundRobin(process,RoundQueue,RoundSize);

}

2……………..........Q.No 12………………..

#include<stdio.h>

struct process

{

char process\_name;

int arrival\_time, burst\_time, ct, waiting\_time, turnaround\_time, priority;

int status;

}process\_queue[10];

int limit;

void Arrival\_Time\_Sorting()

{

struct process temp;

int i, j;

for(i = 0; i < limit - 1; i++)

{

for(j = i + 1; j < limit; j++)

{

if(process\_queue[i].arrival\_time > process\_queue[j].arrival\_time)

{

temp = process\_queue[i];

process\_queue[i] = process\_queue[j];

process\_queue[j] = temp;

}

}

}

}

int main()

{

int i, time = 0, burst\_time = 0, largest;

char c;

float wait\_time = 0, turnaround\_time = 0, average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &limit);

for(i = 0, c = 'A'; i < limit; i++, c++)

{

process\_queue[i].process\_name = c;

printf("\nEnter Details For Process[%C]:\n", process\_queue[i].process\_name);

printf("Enter Arrival Time:\t");

scanf("%d", &process\_queue[i].arrival\_time );

printf("Enter Burst Time:\t");

scanf("%d", &process\_queue[i].burst\_time);

printf("Enter Priority:\t");

scanf("%d", &process\_queue[i].priority);

process\_queue[i].status = 0;

burst\_time = burst\_time + process\_queue[i].burst\_time;

}

Arrival\_Time\_Sorting();

process\_queue[9].priority = -9999;

printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");

for(time = process\_queue[0].arrival\_time; time < burst\_time;)

{

largest = 9;

for(i = 0; i < limit; i++)

{

if(process\_queue[i].arrival\_time <= time && process\_queue[i].status != 1 && process\_queue[i].priority > process\_queue[largest].priority)

{

largest = i;

}

}

time = time + process\_queue[largest].burst\_time;

process\_queue[largest].ct = time;

process\_queue[largest].waiting\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time - process\_queue[largest].burst\_time;

process\_queue[largest].turnaround\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time;

process\_queue[largest].status = 1;

wait\_time = wait\_time + process\_queue[largest].waiting\_time;

turnaround\_time = turnaround\_time + process\_queue[largest].turnaround\_time;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", process\_queue[largest].process\_name, process\_queue[largest].arrival\_time, process\_queue[largest].burst\_time, process\_queue[largest].priority, process\_queue[largest].waiting\_time);

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\n\nAverage waiting time:\t%f\n", average\_waiting\_time);

printf("Average Turnaround Time:\t%f\n", average\_turnaround\_time);

}