NAME – Utkrist Ark SECTION – K21CH

REGISTRATION NUMBER – 12105798 ROLL NO- 64

COURSE – Operating System COURSE CODE – CSE316

GitHub link --- https://github.com/iminsgineark/OS-Scheduling-Process

Question :

Design a scheduling program to implements a Queue with two levels:
Level 1 : Fixed priority preemptive Scheduling
Level 2 : Round Robin Scheduling
For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2.
All the processes in second level queue will complete their execution according to round robin scheduling.
Consider:

1. Queue 2 will be processed after Queue 1 becomes empty.
2. **Priority of Queue 2 has lower priority than in Queue 1.**

DESCRIPTION OF PROBLEM:-

This is a scheduling program to implement a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

For a Fixed priority pre-emptive scheduling if one process P1 is scheduled and running and another process P2 with higher priority comes. The New process with high priority process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2.

All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty. 2.

Priority of Queue 2 has lower priority than in Queue 1.

Fixed-priority preemptive scheduling :

Fixed-priority preemptive scheduling is a scheduling system commonly used in real-time systems. With fixed priority preemptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to execute.

Algorithm:

In this program algorithm for round robin scheduling and multilevel queue scheduling is used.

------ Algorithm For Multilevel Queue:

When a process starts executing then it first enters queue 1.
In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.

1. If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2
2. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit.In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
3. In the last queue, processes are scheduled in FCFS manner.
4. A process in lower priority queue can only execute only when higher priority queues are empty.
5. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.
6.

Round Robin Scheduling:

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.
It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
One of the most commonly used technique in CPU scheduling as a core.
It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
The disadvantage of it is more overhead of context switching.

------Algorithm for round robin scheduling:

How to compute below times in Round Robin using a program?

Completion Time: Time at which process completes its execution.
Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
Waiting Time(W.T): Time Difference between turn around time and burst time.
Waiting Time = Turn Around Time – Burst Time

Steps to find waiting times of all processes:

Create an array **rem_bt[]** to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)

1- Create another array **wt[]** to store

waiting times of processes. Initialize this

array as 0.

2- Initialize time : t = 0

3- Keep traversing the all processes while if it

is all processes are not done. Do following for

i'th process

not done yet.

a- If rem_bt[i] > quantum

(i) t = t + quantum

(ii) bt_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt_rem[i] = 0; // This process is over

Boundary Conditions:

• Level 1 : Fixed priority preemptive Scheduling
• Level 2 : Round Robin Scheduling
• Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.
• Consider 2. Priority of Queue 2 has lower priority than in Queue 1.

Test Cases:

- Time Quantum for Fixed priority queue- 2
- Time Quantum for Round Robin queue- 2

Process	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time
1	0	4	1	10	6
2	0	3	2	13	10
3	0	8	1	8	3
4	10	5	1	20	12

Process	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time
1	0	4	1	9	5

2	1	3	2	15	9
3	2	6	1	17	14
4	**4**	**6**	**1**	**15**	**9**

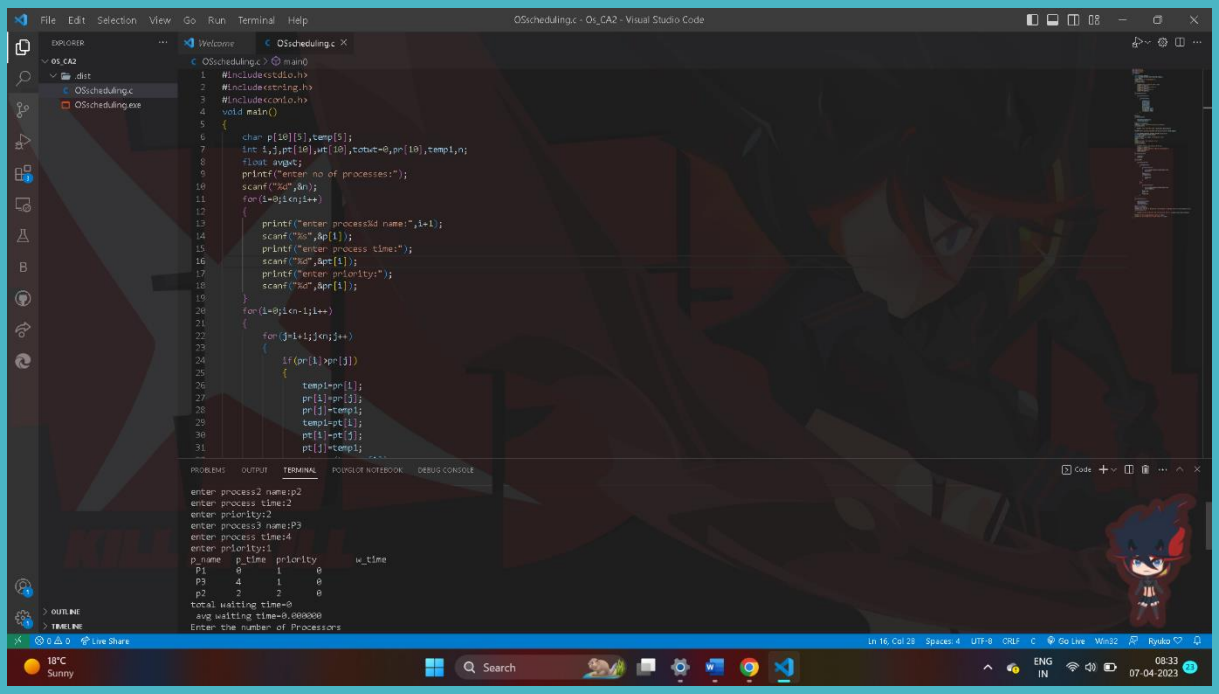
Code :-

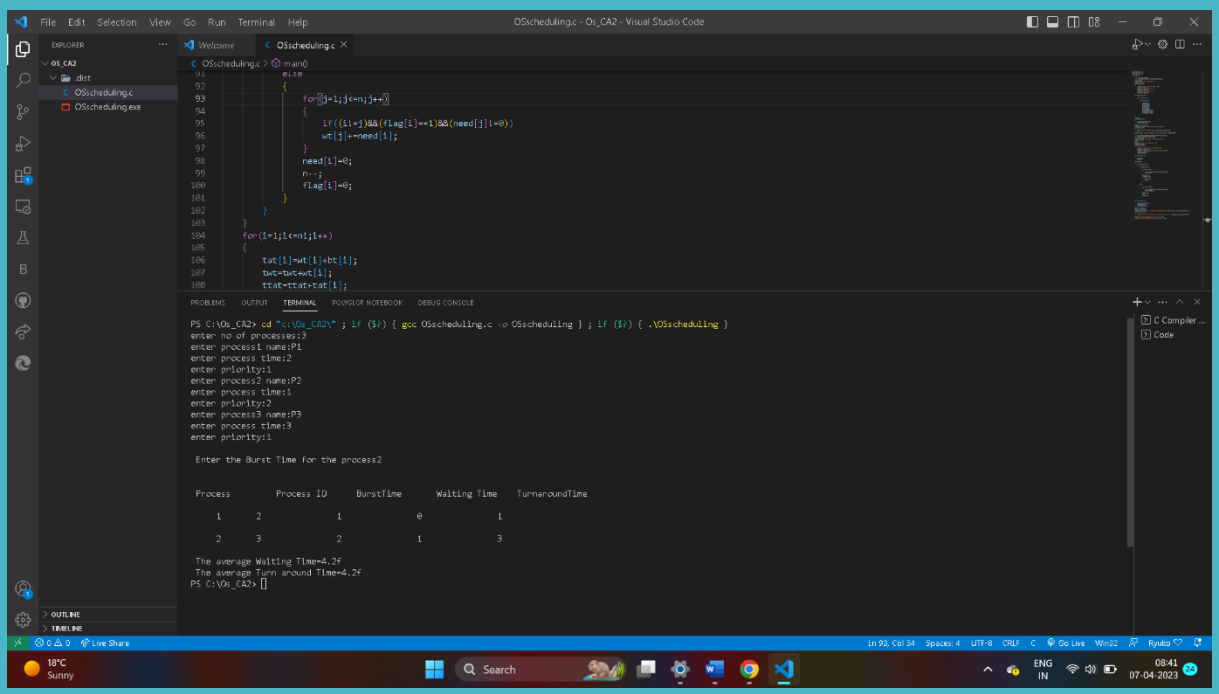
```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
    char p[10][5],temp[5];
    int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
    float avgwt;
    printf("enter no of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter process%d name:",i+1);
        scanf("%s",&p[i]);
        printf("enter process time:");
        scanf("%d",&pt[i]);
        printf("enter priority:");
        scanf("%d",&pr[i]);
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(pr[i]>pr[j])
            {
                temp1=pr[i];
                pr[i]=pr[j];
                pr[j]=temp1;
                temp1=pt[i];
                pt[i]=pt[j];
                pt[j]=temp1;
```

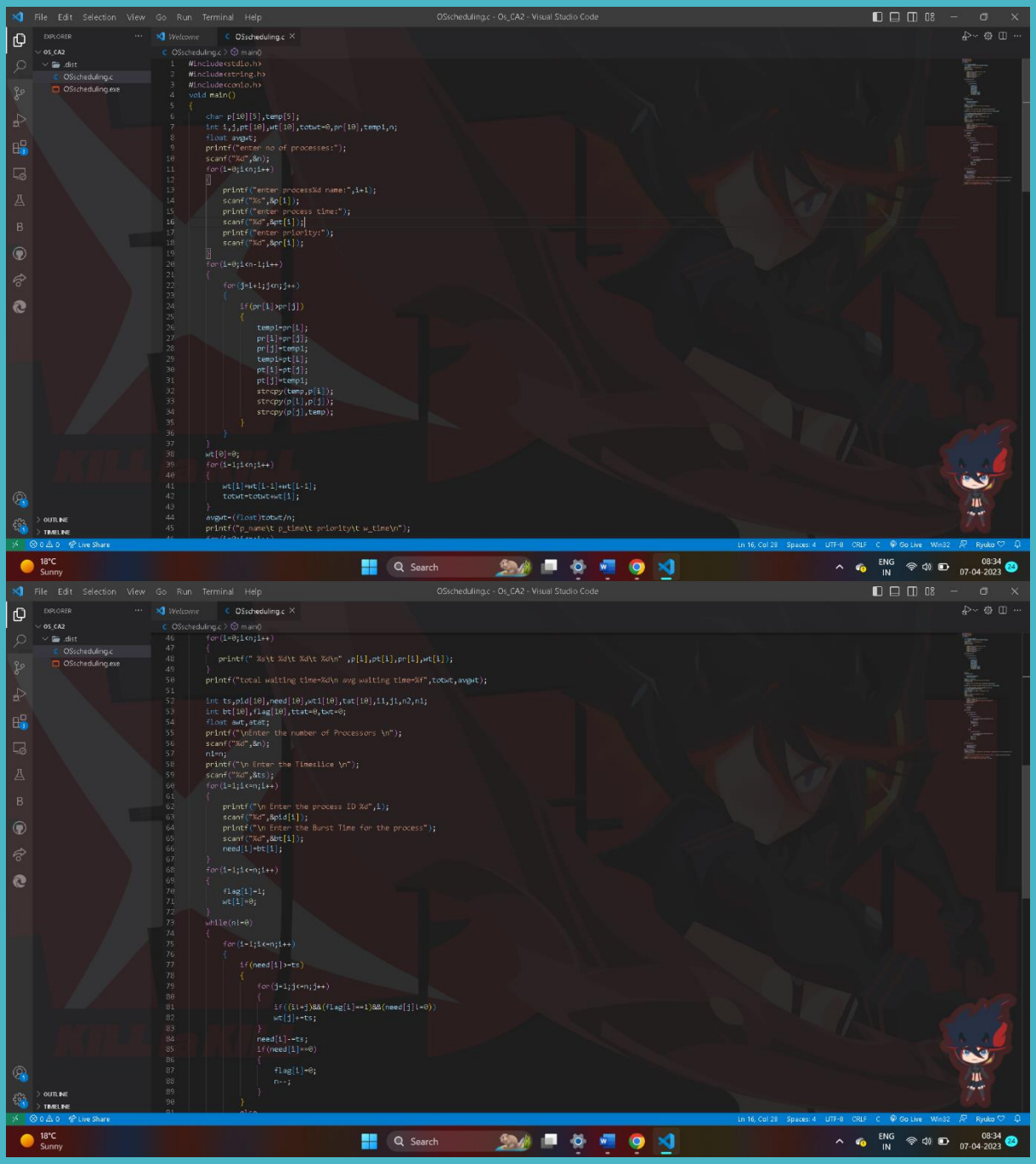
```
            strcpy(temp,p[i]);
            strcpy(p[i],p[j]);
            strcpy(p[j],temp);
          }
       }
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
       wt[i]=wt[i-1]+wt[i-1];
       totwt=totwt+wt[i];
    }
    avgwt=(float)totwt/n;
    printf("p_name\t p_time\t priority\t w_time\n");
    for(i=0;i<n;i++)
    {
      printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
    }
    printf("total waiting time=%d\n avg waiting
time=%f",totwt,avgwt);

    int ts,pid[10],need[10],wt1[10],tat[10],i1,j1,n2,n1;
    int bt[10],flag[10],ttat=0,twt=0;
    float awt,atat;
    printf("\nEnter the number of Processors \n");
    scanf("%d",&n);
    n1=n;
    printf("\n Enter the Timeslice \n");
    scanf("%d",&ts);
    for(i=1;i<=n;i++)
    {
       printf("\n Enter the process ID %d",i);
       scanf("%d",&pid[i]);
```

```
    printf("\n Enter the Burst Time for the process");
    scanf("%d",&bt[i]);
    need[i]=bt[i];
}
for(i=1;i<=n;i++)
{
    flag[i]=1;
    wt[i]=0;
}
while(n!=0)
{
    for(i=1;i<=n;i++)
    {
        if(need[i]>=ts)
        {
            for(j=1;j<=n;j++)
            {
                if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
                wt[j]+=ts;
            }
            need[i]-=ts;
            if(need[i]==0)
            {
                flag[i]=0;
                n--;
            }
        }
        else
        {
            for(j=1;j<=n;j++)
            {
                if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
                wt[j]+=need[i];
```

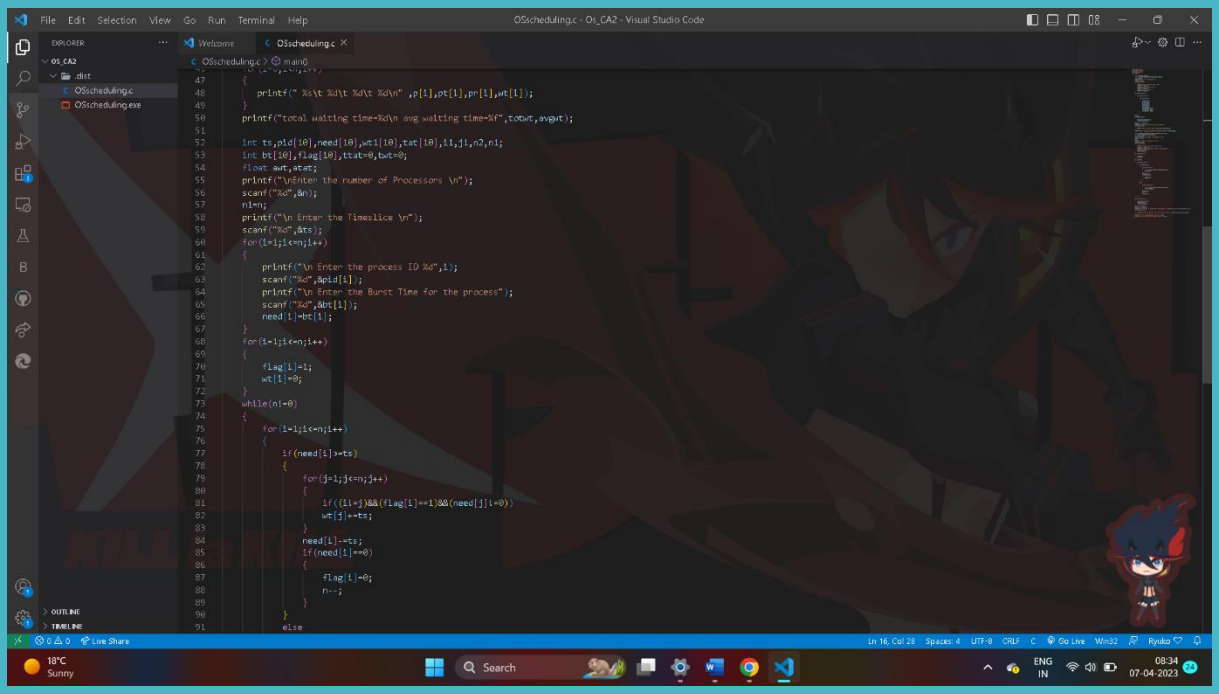
```
                }
                need[i]=0;
                n--;
                flag[i]=0;
            }
        }
    }
    for(i=1;i<=n1;i++)
    {
        tat[i]=wt[i]+bt[i];
        twt=twt+wt[i];
        ttat=ttat+tat[i];
    }
    awt=(float)twt/n1;
    atat=(float)ttat/n1;
    printf("\n\n Process \t Process ID  \t BurstTime \t Waiting
Time \t TurnaroundTime \n ");
    for(i=1;i<=n1;i++)
    {
        printf("\n %5d \t %5d \t\t %5d \t\t %5d \t\t %5d \n",
i,pid[i],bt[i],wt[i],tat[i]);
    }
    printf("\n The average Waiting Time=4.2f",awt);
    printf("\n The average Turn around Time=4.2f",atat);
}
```


File Edit Selection View Go Run Terminal Help
OSscheduling.c - Os_CA2 - Visual Studio Code
EXPLORER
OS_CA2
.dist
OSscheduling.c
OSscheduling.exe
Welcome
OSscheduling.c
OSscheduling.c > main()
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
char p[10][5],temp[5];
int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
float avgwt;
printf("enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process%d name:",i+1);
scanf("%s",&p[i]);
printf("enter process time:");
scanf("%d",&pt[i]);
printf("enter priority:");
scanf("%d",&pr[i]);
}
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(pr[i]>pr[j])
{
temp1=pr[i];
pr[i]=pr[j];
pr[j]=temp1;
temp1=pt[i];
pt[i]=pt[j];
pt[j]=temp1;
PROBLEMS OUTPUT TERMINAL POSTGLOT NOTEBOOK DEBUG CONSOLE
enter process2 name:p2
enter process time:2
enter priority:2
enter process3 name:P3
enter process time:4
enter priority:1
p_name p_time priority w_time
P1 0 1 0
P3 4 1 0
p2 2 2 0
total waiting time=0
avg waiting time=0.000000
Enter the number of Processors
OUTLINE
TIMELINE
Live Share
Ln 16, Col 28 Spaces: 4 UTF-8 CRLF C Go Live Win32 Ryuko
18°C Sunny
Search
ENG IN
08:33 07-04-2023


File Edit Selection View Go Run Terminal Help
OSscheduling.c - Os_CA2 - Visual Studio Code
EXPLORER
OS_CA2
.dist
OSscheduling.c
OSscheduling.exe
Welcome
OSscheduling.c
OSscheduling.c > main()
else
{
for(j=i;j<n;j++)
{
if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
wt[j]+=need[i];
}
need[i]=0;
n--;
flag[i]=0;
}
}
}
for(i=1;i<=n;i++)
{
tat[i]=wt[i]+bt[i];
twt=twt+wt[i];
ttat=ttat+tat[i];
PROBLEMS OUTPUT TERMINAL POSTGLOT NOTEBOOK DEBUG CONSOLE
PS C:\Os_CA2> cd "c:\Os_CA2\" ; if ($?) { gcc OSscheduling.c -o OSscheduling } ; if ($?) { .\OSscheduling }
enter no of processes:3
enter process1 name:P1
enter process time:2
enter priority:1
enter process2 name:P2
enter process time:1
enter priority:2
enter process3 name:P3
enter process time:3
enter priority:1
Enter the Burst Time for the process2
Process Process ID BurstTime Waiting Time TurnaroundTime
1 2 1 0 1
2 3 2 1 3
The average Waiting Time=4.2f
The average Turn around Time=4.2f
PS C:\Os_CA2>
C Compiler ...
Code
OUTLINE
TIMELINE
Live Share
Ln 93, Col 34 Spaces: 4 UTF-8 CRLF C Go Live Win32 Ryuko
18°C Sunny
Search
ENG IN
08:41 07-04-2023


File Edit Selection View Go Run Terminal Help
OSscheduling.c - Os_CA2 - Visual Studio Code
EXPLORER
OS_CA2
.dist
OSscheduling.c
OSscheduling.exe
Welcome
OSscheduling.c
OSscheduling.c > main()
#Includexstdio.h>
#Includexstring.h>
#Includexconio.h>
void main()
{
char p[10][5],temp[5];
int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
float avgwt;
printf("enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process%d name:",i+1);
scanf("%s",&p[i]);
printf("enter process time:");
scanf("%d",&pt[i]);
printf("enter priority:");
scanf("%d",&pr[i]);
}
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(pr[i]>pr[j])
{
temp1=pr[i];
pr[i]=pr[j];
pr[j]=temp1;
temp1=pt[i];
pt[i]=pt[j];
pt[j]=temp1;
strcpy(temp,p[i]);
strcpy(p[i],p[j]);
strcpy(p[j],temp);
}
}
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+wt[i-1];
totwt=totwt+wt[i];
}
avgwt=(float)totwt/n;
printf("p_name\t p_time\t priority\t w_time\n");
OUTLINE
TIMELINE
Ln 16, Col 28 Spaces: 4 UTF-8 CRLF C Go Live Win32 Ryuko
18°C Sunny
Search
ENG IN
08:34 07-04-2023

File Edit Selection View Go Run Terminal Help
OSscheduling.c - Os_CA2 - Visual Studio Code
EXPLORER
OS_CA2
.dist
OSscheduling.c
OSscheduling.exe
Welcome
OSscheduling.c
OSscheduling.c > main()
for(i=0;i<n;i++)
{
printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
}
printf("total waiting time=%d\n avg waiting time=%f",totwt,avgwt);
int ts,pid[10],need[10],wt1[10],tat[10],i1,j1,n2,n1;
int bt[10],flag[10],ttat=0,twt=0;
float awt,atat;
printf("\nEnter the number of Processors \n");
scanf("%d",&n);
n1=n;
printf("\n Enter the Timeslice \n");
scanf("%d",&ts);
for(i=1;i<=n;i++)
{
printf("\n Enter the process ID %d",i);
scanf("%d",&pid[i]);
printf("\n Enter the Burst Time for the process");
scanf("%d",&bt[i]);
need[i]=bt[i];
}
for(i=1;i<=n;i++)
{
flag[i]=1;
wt[i]=0;
}
while(n1=0)
{
for(i=1;i<=n;i++)
{
if(need[i]>=ts)
{
for(j=1;j<=n;j++)
{
if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
wt[j]+=ts;
}
need[i]-=ts;
if(need[i]==0)
{
flag[i]=0;
n--;
}
}
OUTLINE
TIMELINE
Ln 16, Col 28 Spaces: 4 UTF-8 CRLF C Go Live Win32 Ryuko
18°C Sunny
Search
ENG IN
08:34 07-04-2023


File Edit Selection View Go Run Terminal Help
OSscheduling.c - Os_CA2 - Visual Studio Code
EXPLORER
OS_CA2
.dist
OSscheduling.c
OSscheduling.exe
Welcome
OSscheduling.c
OSscheduling.c > main()
printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
printf("total waiting time=%d\n avg waiting time=%f",totwt,avgwt);
int ts,pid[10],need[10],wt1[10],tat[10],i1,j1,n2,n1;
int bt[10],flag[10],ttat=0,twt=0;
float awt,atat;
printf("\nEnter the number of Processors \n");
scanf("%d",&n);
n1=n;
printf("\n Enter the Timeslice \n");
scanf("%d",&ts);
for(i=1;i<=n;i++)
printf("\n Enter the process ID %d",i);
scanf("%d",&pid[i]);
printf("\n Enter the Burst Time for the process");
scanf("%d",&bt[i]);
need[i]=bt[i];
for(i=1;i<=n;i++)
flag[i]=1;
wt[i]=0;
while(n!=0)
for(i=1;i<=n;i++)
if(need[i]>=ts)
for(j=1;j<=n;j++)
if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
wt[j]+=ts;
need[i]-=ts;
if(need[i]==0)
flag[i]=0;
n--;
else
OUTLINE
TIMELINE
Ln 16, Col 28 Spaces: 4 UTF-8 CRLF C Go Live Win32 Ryuko
18°C
Sunny
Search
ENG
IN
08:34
07-04-2023