

DAY -3

CSA0465 – OPERATING SYSTEMS FOR HANDLING DEADLOCKS

LAB EXPERIMENTS – Slot B

Name :- Aswini .P

Reg no :- 192011399

11. Round Robin :-

Program :-

```
#include<stdio.h>

int main()
{
    int count,j,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d",&n);
    remain=n;
    for(count=0;count<n;count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for(time=0,count=0;remain!=0;)
    {
        if(rt[count]<=time_quantum && rt[count]>0)
        {
            time+=rt[count];
```

```

rt[count]=0;
flag=1;
}
else if(rt[count]>0)
{
rt[count]-=time_quantum;
time+=time_quantum;
}
if(rt[count]==0 && flag==1)
{
remain--;
printf("P[%d]\t\t%d\t\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
wait_time+=time-at[count]-bt[count];
turnaround_time+=time-at[count];
flag=0;
}
if(count==n-1)
count=0;
else if(at[count+1]<=time)
count++;
else
count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
return 0;
}

```

Output :-

The screenshot shows a C++ IDE with a file named "11. Round robin.c". The code implements a Round Robin scheduling algorithm. It takes the number of processes, arrival times, burst times, and a time quantum as input. It then calculates the turnaround and waiting times for each process. The output window shows the following results:

```

Enter Total Process: 4
Enter Arrival Time and Burst Time for Process Process Number 1 :3
5
Enter Arrival Time and Burst Time for Process Process Number 2 :2
4
Enter Arrival Time and Burst Time for Process Process Number 3 :5
3
Enter Arrival Time and Burst Time for Process Process Number 4 :2
5
Enter Time Quantum: 2

Process |Turnaround Time|Waiting Time
P[2]    |      6      |      2
P[1]    |     10      |      5
P[3]    |      9      |      6
P[4]    |     15      |     10

Average Waiting Time= 5.750000
Avg Turnaround Time = 10.000000
Process returned 0 (0x0)   execution time : 32.900 s
Press any key to continue.

```

12. Inter Process Communication :-

Program :-

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int mutex = 1;
```

```
int full = 0;
```

```
int empty = 10, x = 0;
```

```
void producer()
```

```
{
    --mutex;

    ++full;

    --empty;

    x++;

    printf("\nProducer produces"
           "item %d",
           x);

    ++mutex;
```

```
}
```

```
void consumer()
```

```
{
```

```

        --mutex;

        --full;

        ++empty;

        printf("\nConsumer consumes "
               "item %d",
               x);

        x--;

        ++mutex;
    }

int main()
{
    int n, i;

    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");

#pragma omp critical
    for (i = 1; i > 0; i++) {
        printf("\nEnter your choice:");

        scanf("%d", &n);

        switch (n) {
            case 1:
                if ((mutex == 1)
                    && (empty != 0)) {
                    producer();
                }
                else {
                    printf("Buffer is full!");
                }

                break;

            case 2:

```

```

        if ((mutex == 1)
            && (full != 0)) {
            consumer();
        }
    else {
        printf("Buffer is empty!");
    }
    break;
case 3:
    exit(0);
    break;
}
}
}

```

Output :-

The screenshot shows a code editor with a C program for the Producer-Consumer problem and a terminal window displaying its output.

Code Editor (12. Inter Process Communication.c):

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  int mutex = 1;
4  int full = 0;
5  int empty = 10, x = 0;
6  void producer()
7  {
8      --mutex;
9      ++full;
10     --empty;
11     x++;
12     printf("\nProducer produces "
13           "item %d",
14           x);
15     ++mutex;
16 }
17 void consumer()
18 {
19     --mutex;
20     --full;
21     ++empty;
22     printf("\nConsumer consumes "
23           "item %d",
24           x);
25     x--;
26     ++mutex;
27 }
28 int main()
29 {
30     int n, i;
31     printf("\n1. Press 1 for Producer"
32           "\n2. Press 2 for Consumer"
33           "\n3. Press 3 for Exit");
34     #pragma omp critical
35     for (i = 1; i > 0; i++) {
36         printf("\nEnter your choice:");
37         scanf("%d", &n);

```

Terminal Window Output:

```

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1
Producer produces item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:1
Producer produces item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:3
Process returned 0 (0x0)   execution time : 36.247 s
Press any key to continue.

```

13. Dinning Philosopher :-

Program :-

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
    int i,a[5];
    pthread_t tid[5];

    sem_init(&room,0,4);

    for(i=0;i<5;i++)
        sem_init(&chopstick[i],0,1);

    for(i=0;i<5;i++){
        a[i]=i;
        pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
    }
    for(i=0;i<5;i++)
        pthread_join(tid[i],NULL);
}
```

```
void * philosopher(void * num)
{
    int phil=*(int *)num;

    sem_wait(&room);
    printf("\nPhilosopher %d has entered room",phil);
    sem_wait(&chopstick[phil]);
    sem_wait(&chopstick[(phil+1)%5]);

    eat(phil);
    sleep(2);
    printf("\nPhilosopher %d has finished eating",phil);

    sem_post(&chopstick[(phil+1)%5]);
    sem_post(&chopstick[phil]);
    sem_post(&room);
}
```

```
void eat(int phil)
{
    printf("\nPhilosopher %d is eating",phil);
}
```

Output :-

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<pthread.h>
4  #include<semaphore.h>
5  #include<unistd.h>
6
7  sem_t room;
8  sem_t chopstick[5];
9
10 void * philosopher(void *);
11 void eat(int);
12 int main()
13 {
14     int i,a[5];
15     pthread_t tid[5];
16
17     sem_init(&room,0,4);
18
19     for(i=0;i<5;i++)
20         sem_init(&chopstick[i],0,1);
21
22     for(i=0;i<5;i++){
23         a[i]=i;
24         pthread_create(&tid[i],NULL,philosopher,(void *) &a[i]);
25     }
26     for(i=0;i<5;i++)
27         pthread_join(tid[i],NULL);
28 }
29
30 void * philosopher(void * num)
31 {
32     int phil=(int *) num;
33
34     sem_wait(&room);
35     printf("\nPhilosopher %d has entered room\n", phil);

```

```

Philosopher 0 has entered room
Philosopher 2 has entered room
Philosopher 2 is eating
Philosopher 1 has entered room
Philosopher 3 has entered room
Philosopher 0 is eating
Philosopher 0 has finished eating
Philosopher 2 has finished eating
Philosopher 1 is eating
Philosopher 4 has entered room
Philosopher 3 is eating
Philosopher 1 has finished eating
Philosopher 3 has finished eating
Philosopher 4 is eating
Philosopher 4 has finished eating
Process returned 0 (0x0)   execution time : 6.081 s
Press any key to continue.

```

14. Banker's Algorithm :-

Program :-

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int n,r,i,j,k,p,u=0,s=0,m;
```

```
    int block[10],run[10],active[10],newreq[10];
```

```
    int max[10][10],resalloc[10][10],resreq[10][10];
```

```
    int totalloc[10],totext[10],simalloc[10];
```

```
    //clrscr();
```

```
    printf("Enter the no of processes:");
```

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

```
    printf("Enter the no of resource classes:");
```

```
    scanf("%d",&r);
```

```
    printf("Enter the total existed resource in each class:");
```

```
    for(k=1; k<=r; k++)
```

```
        scanf("%d",&totext[k]);
```



```

printf("Enter the allocated resources:");
for(i=1; i<=n; i++)
    for(k=1; k<=r; k++)
        scanf("%d",&resalloc);
printf("Enter the process making the new request:");
scanf("%d",&p);
printf("Enter the requested resource:");
for(k=1; k<=r; k++)
    scanf("%d",&newreq[k]);
printf("Enter the process which are n blocked or running:");
for(i=1; i<=n; i++)
{
    if(i!=p)
    {
        printf("process %d:\n",i+1);
        scanf("%d%d",&block[i],&run[i]);
    }
}
block[p]=0;
run[p]=0;
for(k=1; k<=r; k++)
{

    j=0;
    for(i=1; i<=n; i++)
    {
        totalloc[k]=j+resalloc[i][k];
        j=totalloc[k];
    }
}

```

```

for(i=1; i<=n; i++)
{
    if(block[i]==1||run[i]==1)
        active[i]=1;
    else
        active[i]=0;
}
for(k=1; k<=r; k++)
{
    resalloc[p][k]+=newreq[k];
    totalloc[k]+=newreq[k];
}
for(k=1; k<=r; k++)
{
    if(totext[k]-totalloc[k]<0)
    {
        u=1;
        break;
    }
}
if(u==0)
{
    for(k=1; k<=r; k++)
        simalloc[k]=totalloc[k];
    for(s=1; s<=n; s++)
        for(i=1; i<=n; i++)
        {
            if(active[i]==1)
            {
                j=0;

```

```

    for(k=1; k<=r; k++)
    {
        if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
        {
            j=1;
            break;
        }
    }
    if(j==0)

    {
        active[i]=0;
        for(k=1; k<=r; k++)
            simalloc[k]=resalloc[i][k];
    }
    }
m=0;
for(k=1; k<=r; k++)
    resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
}
else
{
    for(k=1; k<=r; k++)
    {
        resalloc[p][k]=newreq[k];
        totalloc[k]=newreq[k];
    }
    printf("Deadlock will occur");
}

```

```
}
```

```
}
```

Output :-

The screenshot shows a C program for the Banker's Algorithm in a code editor and its execution output in a terminal window.

Code Editor (15. Multi Threading.c):

```
1 #include<stdio.h>
2
3 int main()
4 {
5     int n,r,i,j,k,p,u=0,s=0,m;
6     int block[10],run[10],active[10],newreq[10];
7     int max[10][10],resalloc[10][10],resreq[10][10];
8     int totalloc[10],totext[10],simalloc[10];
9     //GLASSER();
10    printf("Enter the no of processes:");
11    scanf("%d",&n);
12    printf("Enter the no of resource classes:");
13    scanf("%d",&r);
14    printf("Enter the total existed resource in each class:");
15    for(k=1; k<=r; k++)
16        scanf("%d",&totext[k]);
17    printf("Enter the allocated resources:");
18    for(i=1; i<=n; i++)
19        for(k=1; k<=r; k++)
20            scanf("%d",&resalloc[k]);
21    printf("Enter the process making the new request:");
22    scanf("%d",&p);
23    printf("Enter the requested resource:");
24    for(k=1; k<=r; k++)
25        scanf("%d",&newreq[k]);
26    printf("Enter the process which are n blocked or running:");
27    for(i=1; i<=n; i++)
28    {
29        if(i!=p)
30        {
31            printf("process %d:\n",i+1);
32            scanf("%d%d",&block[i],&run[i]);
33        }
34    }
35    block[p]=0;
36    run[p]=0;
```

Terminal Output:

```
Enter the no of processes:4
Enter the no of resource classes:4
Enter the total existed resource in each class:0 0 1 2
Enter the allocated resources:0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
Enter the process making the new request:3
Enter the requested resource:2
1 0 0 0
Enter the process which are n blocked or running:process 2:
1 3 5 4
process 3:
process 5:
1 0 8 4
Deadlock will occur
Process returned 0 (0x0)   execution time : 144.167 s
Press any key to continue.
```

15. Multi threading :-

Program :-

```
#include<string.h>
```

```
#include<stdlib.h>
```

```
#include<stdio.h>
```

```
struct
```

```
{
```

```
char dname[10],fname[10][10];
```

```
int fcnt;
```

```
}dir[10];
```

```
int main()
```

```
{
```

```
int i,ch,dcnt,k;
```

```
char f[30], d[30];
```

```

dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);

```

```

for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");

```

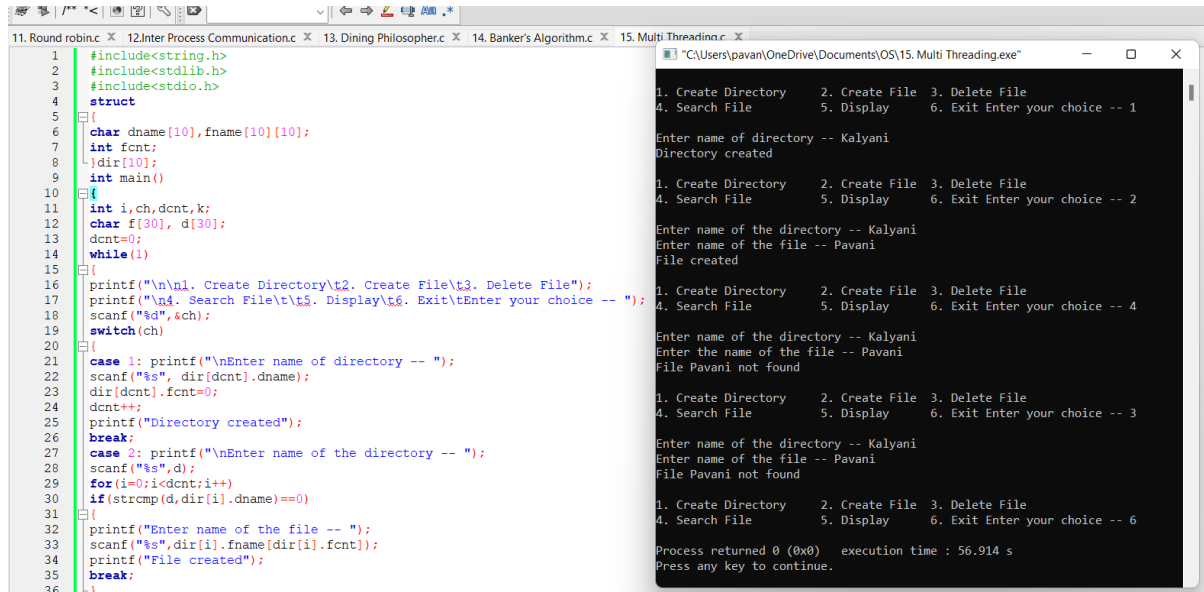
```

scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}

```

}

Output :-



The image shows a C++ IDE with multiple tabs. The active tab is '15. Multi Threading.c', which contains the following code:

```
1 #include<string.h>
2 #include<stdlib.h>
3 #include<stdio.h>
4 struct
5 {
6     char dname[10],fname[10][10];
7     int fcnt;
8 }dir[10];
9 int main()
10 {
11     int i,ch,dcnt,k;
12     char f[30],d[30];
13     dcnt=0;
14     while(1)
15     {
16         printf("\n1. Create Directory\t2. Create File\t3. Delete File");
17         printf("\n4. Search File\t5. Display\t6. Exit\tEnter your choice -- ");
18         scanf("%d",&ch);
19         switch(ch)
20         {
21             case 1: printf("\nEnter name of directory -- ");
22                     scanf("%s", dir[dcnt].dname);
23                     dir[dcnt].fcnt=0;
24                     dcnt++;
25                     printf("Directory created");
26                     break;
27             case 2: printf("\nEnter name of the directory -- ");
28                     scanf("%s",d);
29                     for(i=0;i<dcnt;i++)
30                     if(strcmp(d,dir[i].dname)==0)
31                     {
32                         printf("Enter name of the file -- ");
33                         scanf("%s",dir[i].fname[dir[i].fcnt]);
34                         printf("File created");
35                         break;
36                     }
```

The output window shows the execution of the program. It displays a menu with six options: 1. Create Directory, 2. Create File, 3. Delete File, 4. Search File, 5. Display, and 6. Exit. The user enters '1' to create a directory, then 'Kalyani' as the directory name. The program prints 'Directory created'. The user then enters '2' to create a file, then 'Pavani' as the file name. The program prints 'File created'. The user then enters '4' to search for a file, then 'Pavani' as the file name. The program prints 'File Pavani not found'. The user then enters '6' to exit. The program prints 'Process returned 0 (0x0) execution time : 56.914 s' and 'Press any key to continue.'.