OPERATING SYSTEMS – CSE316

Assignment based on Multithreading and Page Fault

Student Name: Sasikiran Gandepalli

Student Registration ID – 11807372

Email Id – sasi.gandepalli@gmail.com

Github – https://github.com/Ricky-1999/sasiassignment

Section – K18GT

Roll No – 45

1.Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
*P*1	0.0	8
*P*2	0.4	4
*P*3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?
b. What is the average turnaround time for these processes with the SJF scheduling algorithm?
c. Compute what average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P*1 and *P*2 are waiting during this idle time, so their waiting time may increase.

ANS) A . FCFS Gantt Chart

```
Proc:   1              2       3

Time:   0              8       12 13
```

Average Turnaround Time: ((8-0)+(12-0.4)+(13-1.0)) / 3 = 10.53

B.) SJF Gantt Chart

```
Proc:   1              3  2

Time:   0              8  9       13
```

Average Turnaround Time: ((8-0)+(13-0.4)+(9-1.0)) / 3 = 9.53

C.) Proc: x 3 2   1

```
Time:   0  1  2        6                 14
```

Average Turnaround Time = ((14-0)+(6-0.4)+(2-1.0)) / 3 = 6.87

1. C Program for the FCFS ans SJF

```c
#include <stdio.h>

#include <stdlib.h>


#define MAX 100


typedef struct

{

    int pid;

    int burst_time;

    int waiting_time;

    int turnaround_time;

} Process;


void print_table(Process p[], int n);

void print_gantt_chart(Process p[], int n);


int main()

{

    Process p[MAX];
```

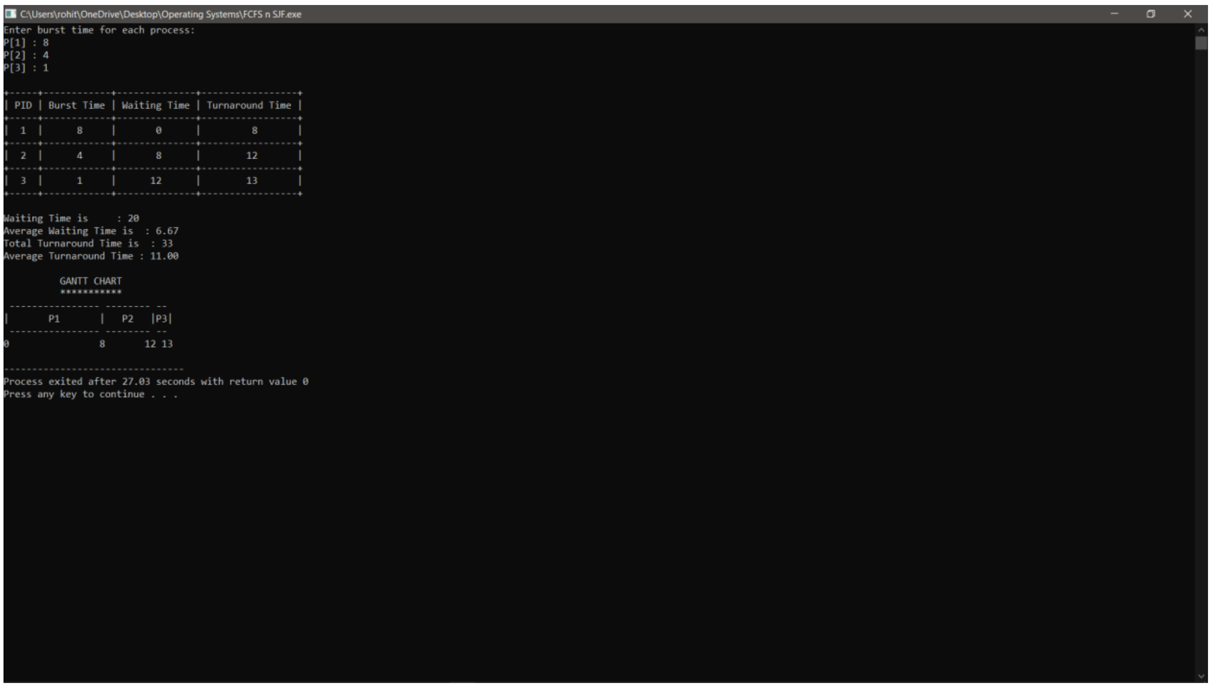
```
int i, j, n;

int sum_waiting_time = 0, sum_turnaround_time;

printf("Enter total number of process: ");

scanf("%d", &n);

printf("Enter burst time for each process:\n");

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

   p[i].pid = i+1;

   printf("P[%d] : ", i+1);

   scanf("%d", &p[i].burst_time);

   p[i].waiting_time = p[i].turnaround_time = 0;

}


// calculate waiting time and turnaround time

p[0].turnaround_time = p[0].burst_time;


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

   p[i].waiting_time = p[i-1].waiting_time + p[i-1].burst_time;

   p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;

}


// calculate sum of waiting time and sum of turnaround time

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

      sum_waiting_time += p[i].waiting_time;

      sum_turnaround_time += p[i].turnaround_time;
```

```
    }

    // print table
    puts(""); // Empty line
    print_table(p, n);
    puts(""); // Empty Line
    printf("Total Waiting Time      : %-2d\n", sum_waiting_time);
    printf("Average Waiting Time    : %-2.2lf\n", (double)sum_waiting_time / (double) n);
    printf("Total Turnaround Time   : %-2d\n", sum_turnaround_time);
    printf("Average Turnaround Time : %-2.2lf\n", (double)sum_turnaround_time / (double)
n);

    // print Gantt chart
    puts(""); // Empty line
    puts("          GANTT CHART          ");
    puts("          ***********          ");
    print_gantt_chart(p, n);
    return 0;
}


void print_table(Process p[], int n)
{
    int i;
    puts("+-----+-----------+-------------+----------------+");
    puts("| PID | Burst Time | Waiting Time | Turnaround Time |");
    puts("+-----+-----------+-------------+----------------+");

    for(i=0; i<n; i++) {
        printf("| %2d  |    %2d     |      %2d      |        %2d       |\n"
               , p[i].pid, p[i].burst_time, p[i].waiting_time, p[i].turnaround_time );
        puts("+-----+-----------+-------------+----------------+");
    }

}


void print_gantt_chart(Process p[], int n)
{
    int i, j;
    // print top bar
    printf(" ");
    for(i=0; i<n; i++) {
        for(j=0; j<p[i].burst_time; j++) printf("--");
        printf(" ");
    }
    printf("\n|");
```

```
// printing process id in the middle
for(i=0; i<n; i++) {
    for(j=0; j<p[i].burst_time - 1; j++) printf(" ");
    printf("P%d", p[i].pid);
    for(j=0; j<p[i].burst_time - 1; j++) printf(" ");
    printf("|");
}
printf("\n ");
// printing bottom bar
for(i=0; i<n; i++) {
    for(j=0; j<p[i].burst_time; j++) printf("--");
    printf(" ");
}
printf("\n");


// printing the time line
printf("0");
for(i=0; i<n; i++) {
    for(j=0; j<p[i].burst_time; j++) printf("  ");
    if(p[i].turnaround_time > 9) printf("\b"); // backspace : remove 1 space
    printf("%d", p[i].turnaround_time);


}
printf("\n");
```

}

Runtime Execution:



```
C:\Users\rohit\OneDrive\Desktop\Operating Systems\FCFS n SJF.exe
Enter burst time for each process:
P[1] : 8
P[2] : 4
P[3] : 1

+-----+------------+-------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+-------------+-----------------+
|  1  |     8      |      0      |        8        |
+-----+------------+-------------+-----------------+
|  2  |     4      |      8      |       12        |
+-----+------------+-------------+-----------------+
|  3  |     1      |     12      |       13        |
+-----+------------+-------------+-----------------+

Waiting Time is     : 20
Average Waiting Time is  : 6.67
Total Turnaround Time is  : 33
Average Turnaround Time : 11.00

          GANTT CHART
          ***********
 ---------------- -------- --
|       P1       |   P2   |P3|
 ---------------- -------- --
0                8       12 13

--------------------------------
Process exited after 27.03 seconds with return value 0
Press any key to continue . . .
```

Q)21. A number of cats and mice inhabit a house. The cats and mice have worked out a deal where the mice can steal pieces of the cats' food, so long as the cats never see the mice actually doing so. If the cats see the mice, then the cats must eat the mice (or else lose face with all of their cat friends). There are NumBowls cat food dishes, NumCats cats, and NumMice mice. Your job is to synchronize the cats and mice so that the following requirements are satisfied:No mouse should ever get eaten. You should assume that if a cat is eating at a food dish, any mouse attempting to eat from that dish or any other food dish will be seen and eaten. When cats aren't eating, they will not see mice eating. In other words, this requirement states that if a cat is eating from any bowl,then no mouse should be eating from any bowl. Only one mouse or one cat may eat from a given dish at any one time. Neither cats nor mice should starve. A cat or mouse that wants to eat should eventually be able to eat. For example, a synchronization solution that permanently prevents all mice from eating would be unacceptable. When we actually test your solution, each simulated cat and mouse will only eat a finite number of times; however, even if the simulation were allowed to run forever, neither cats nor mice should starve.

<u>**Solution →**</u>

In a rectangular field of size n by m squares there is a mouse and two cats. The mouse is the first to make a move, then each of the cats makes a move, then again its the mouse's turn, and so on. In each move both the mouse and the cats can move exactly one square vertically or horizontally. If the mouse is standing at the edge of the field then in its next move it can jump off the field and is saved from the cats. If in the next move one of the cats moves to the field with the mouse then there is no escape for the mouse

Function Description

Complete the *catAndMouse* function in the editor below. It should return one of the three strings as described.

catAndMouse has the following parameter(s):

- *x*: an integer, Cat A's position
- *y*: an integer, Cat B's position
- *z*: an integer, Mouse C's position

Input Format

The first line contains a single integer, q, denoting the number of queries.
Each of the q subsequent lines contains three space-separated integers describing the respective values of x (cat A's location),y (cat B's location), and z(mouse C's location).

Constraints

- 1<=q<=100
- 1<=x,y,z<=100

Output Format

For each query, return Cat A if cat A catches the mouse first, Cat B if cat B catches the mouse first, or Mouse C if the mouse escapes.

Sample Input 0

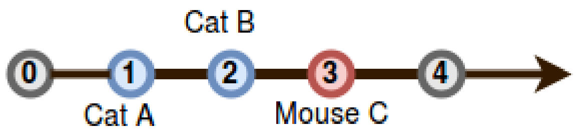
```
2
1 2 3
1 3 2
```

Sample Output 0

```
Cat B
Mouse C
```

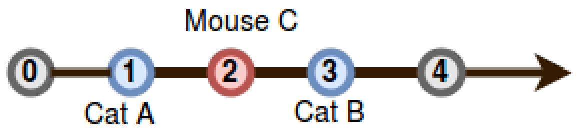
Explanation 0

Query 0: The positions of the cats and mouse are shown below:



Cat B will catch the mouse first, so we print Cat B on a new line.

Query 1: In this query, cats A and B reach mouse C at the exact same time:



Because the mouse escapes, we print Mouse C on a new line.

2. **C Program for the Cats and Mice problem.**

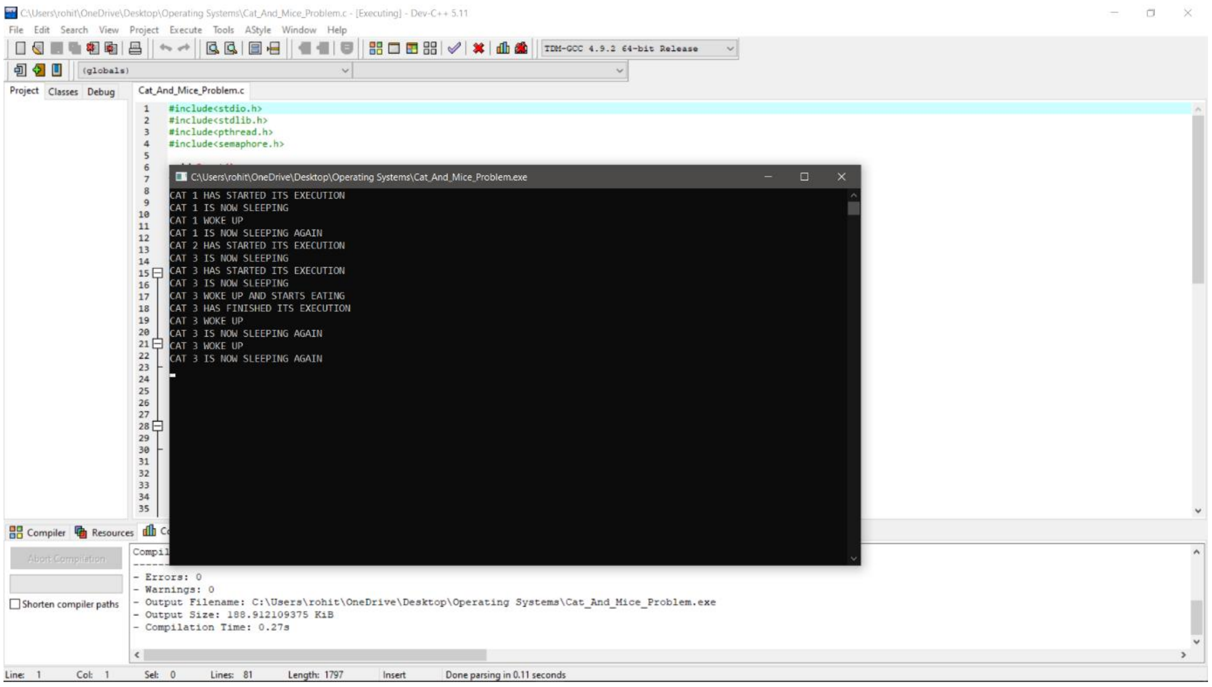
```
#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>


void * cat();

void * mice();

int NumBowls[20],num=0 ,arr[20];

int NumCats=0,NumMice=0;

sem_t numberOfCats,numberOfMice;

pthread_t thread1,thread2,thread3,thread4,thread5;

pthread_mutex_t mutex,catmutex,micemutex;


void * mice()

{

                NumMice=NumMice+1;

                arr[NumMice]=NumMice;
```

```
		int i=NumMice;


	sem_wait(&numberOfMice);
	if(NumMice==1){
				pthread_mutex_lock(&micemutex);
	}
	printf("MOUSE %d IS EATING \n",NumMice);
	printf("MOUSE %d IS SLEEPING \n",NumMice);
	sleep(5);
	if(i!=arr[i])
	{
		return;
	}
	printf("MOUSE %d WOKE UP AND STARTS EATING \n",NumMice);
	sleep(5);


	printf("MOUSE %d HAS EXECUTED\n",NumMice);


	pthread_mutex_unlock(&micemutex);
}


void * cat()
{
	pthread_mutex_lock(&mutex);
```

```
        NumCats=NumCats+1;

        num=num+1;

        printf("CAT %d HAS STARTED ITS EXECUTION \n",NumCats);

        printf("CAT %d IS NOW SLEEPING \n",NumCats);

        sleep(5);


        printf("CAT %d WOKE UP \n",NumCats);

        while(NumMice>0)

        {

        sem_destroy(&numberOfMice);

        printf("MOUSE %d IS DEAD %d \n",NumMice);

        arr[NumMice]=-1;

        NumMice=NumMice-1;

        }

        printf("CAT %d IS NOW SLEEPING AGAIN\n",NumCats);

        sleep(5);


        printf("CAT %d WOKE UP AND STARTS EATING\n",NumCats);

        NumBowls[num]=num;

        printf("CAT %d HAS FINISHED ITS EXECUTION \n",NumCats);

        pthread_mutex_unlock(&mutex);

}


int main()
```

```
{   int num=5,x;

        sem_init(&numberOfCats,0,5);

        sem_init(&numberOfMice,0,5);

        pthread_create(&thread1,NULL,cat,NULL);

        sleep(10);

        pthread_create(&thread2,NULL,cat,NULL);

        pthread_create(&thread3,NULL,cat,NULL);

        sleep(10);

        pthread_create(&thread4,NULL,cat,NULL);

        pthread_create(&thread5,NULL,mice,NULL);

        pthread_join(thread1,NULL);

        pthread_join(thread2,NULL);

        pthread_join(thread3,NULL);

        pthread_join(thread4,NULL);

        pthread_join(thread5,NULL);

        }
```

Output Runtime Code:


C:\Users\rohit\OneDrive\Desktop\Operating Systems\Cat_And_Mice_Problem.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Cat_And_Mice_Problem.c
Compiler Resources
Abort Compilation
Shorten compiler paths

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

C:\Users\rohit\OneDrive\Desktop\Operating Systems\Cat_And_Mice_Problem.exe

```
CAT 1 HAS STARTED ITS EXECUTION
CAT 1 IS NOW SLEEPING
CAT 1 WOKE UP
CAT 1 IS NOW SLEEPING AGAIN
CAT 2 HAS STARTED ITS EXECUTION
CAT 3 IS NOW SLEEPING
CAT 3 HAS STARTED ITS EXECUTION
CAT 3 IS NOW SLEEPING
CAT 3 WOKE UP AND STARTS EATING
CAT 3 HAS FINISHED ITS EXECUTION
CAT 3 WOKE UP
CAT 3 IS NOW SLEEPING AGAIN
CAT 3 WOKE UP
CAT 3 IS NOW SLEEPING AGAIN
```

```
Compil
------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\rohit\OneDrive\Desktop\Operating Systems\Cat_And_Mice_Problem.exe
- Output Size: 188.912109375 KiB
- Compilation Time: 0.27s
```

Line: 1 Col: 1 Sel: 0 Lines: 81 Length: 1797 Insert Done parsing in 0.11 seconds