**NAME: ABHISEK RAUTARAY**

**REGISTRATION NO.: 11802710**

1. **TECH CSE**

**EMAIL ID - abhisekrautaray905@gmail.com**

**OPERATING SYSTEM ASSIGNMENT**

**GitHub link: [https://github.com/abhisekgudu/OS-Assignment](https://github.com/puru531/OS-Assignment)**

**Q.21 :**

**SOLUTION CODE :-**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

// global variable for better optimaization

int NumBowls[15],num=0 ,arr[15];

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);

}

**Algorithm behind solution :-**

Firstly, 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,position of Cat A.

->y: an integer, position of Cat B.

->z: an integer, position of Mouse C.

Input Format

The first line contains a single integer, t, 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).

**Boundary Condition (Constraints):-**

1. 1<=t<=100
2. 1<=x,y,z<=100

Test Case 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:, for sample input 1 only

Cat B

1. ----(1)-----(2)------(3)-------(4)--------->

Cat A Mouse C

Cat B will catch the mouse first, so we print Cat B on a new line., for sample input 2

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

Cat B

1. ---------(1)---------(2)-----------(3)--------(4)-------->

Cat A Mouse C

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

**Q.6:**

**Solution Code:-**

#include <stdio.h>

#include <stdlib.h>

#define M 100 //c Marcos for optimization

// for defining data type of Proccess

typedef struct

{

int pid, burst, wait, turntime;

} Process;

void print\_table(Process p[], int n);// for printning table of proccess

void print\_gantt\_chart(Process p[], int n); // for printing ghant chart

int main()

{

Process p[M];

int i, j, n;

int sum\_wait = 0, sum\_turntime;

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);

p[i].wait = p[i].turntime = 0;

}

// calculate waiting time and turnaround time

p[0].turntime = p[0].burst;

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

p[i].wait = p[i-1].wait + p[i-1].burst;

p[i].turntime = p[i].wait + p[i].burst;

}

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

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

sum\_wait += p[i].wait;

sum\_turntime += p[i].turntime;

}

// print table

puts(""); // Empty line

print\_table(p, n);

puts(""); // Empty Line

printf("Waiting Time is : %-2d\n", sum\_wait);

printf("Average Waiting Time is : %-2.2lf\n", (double)sum\_wait / (double) n);

printf("Total Turnaround Time is : %-2d\n", sum\_turntime);

printf("Average Turnaround Time : %-2.2lf\n", (double)sum\_turntime / (double) n);

// print Gantt chart

puts(""); // Empty line

//puts(" GANTT CHART ");

puts(" -:GANTT CHART :- ");

puts(" ");

print\_gantt\_chart(p, n);

return 0;

}

void print\_table(Process p[], int n)

{

int i;

puts("|\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|");

// 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, p[i].wait, p[i].turntime );

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; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

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

for(j=0; j<p[i].burst - 1; j++) printf(" ");

printf("P%d", p[i].pid);

for(j=0; j<p[i].burst - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

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

for(j=0; j<p[i].burst; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

printf("0");

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

for(j=0; j<p[i].burst; j++) printf(" ");

if(p[i].turntime > 9) printf("\b"); // backspace : remove 1 space

printf("%d", p[i].turntime);

}

printf("\n");

}

**Algorithm:-**

Let take an example of proccess with arrival time and brust time

Process Arrival Time Burst Time

P1 0.0 8

P2 0.4 4

P3 1.0 1

Part A. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

First Come First Serve Gantt Chart

Proccess: 1 2 3

|---------------|-------|----|

Time: 0 8 12 13

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

B. What is the average turnaround time for these processes with the SJF scheduling algorithm?

Proccess: 1 3 2

|-------------|--|--------| (sjf gant chart)

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

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.

Proccess: 0 3 2 1

|--|--|---------|-----------------|

Time: 0 1 2 6 14

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

**Boundary Condition (constraints):**-

P -> Proccess

0 <P<=100000

**Test Case 0:-**

Enter total number of process: 2

Enter burst time for each process:

P[1] : 12

P[2] : 13

**Sample Output:-**

|\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

| PID | Burst Time | Waiting Time | Turnaround Time |

|-----|------------|--------------|-----------------|

| 1 | 12 | 0 | 12 |

|-----|------------|--------------|-----------------|

| 2 | 13 | 12 | 25 |

|-----|------------|--------------|-----------------|

Waiting Time is : 12

Average Waiting Time is : 6.00

Total Turnaround Time is : 37

Average Turnaround Time : 18.50

**Github Revision :**

**I had 3 revision in github in this project.**