**Student Name: Sri Harsha Morla**

**Student ID:11805408.**

**Code 1 ( Question 6):**

// C program for implementation of FCFS

#include<stdio.h>

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

for (int  i = 1; i < n ; i++ )

wt[i] =  bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

for (int  i = 0; i < n ; i++)

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

}

//Function to calculate average time

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

{

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

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

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

//Display processes along with all details

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

// Calculate total waiting time and total turn

// around time

for (int  i=0; i<n; i++)

{

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

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

printf("   %d ",(i+1));

printf("       %d ", bt[i] );

printf("       %d",wt[i] );

printf("       %d\n",tat[i] );

}

int s=(float)total\_wt / (float)n;

int t=(float)total\_tat / (float)n;

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int  burst\_time[] = {10, 5, 8};

findavgTime(processes, n,  burst\_time);

return 0;

}

**Code 2( Question 6):**

#include <bits/stdc++.h>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Function to find the waiting time for all

// processes

void findWaitingTime(Process proc[], int n,

int wt[])

{

int rt[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

for (int j = 0; j < n; j++) {

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate turn around time

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

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

}

// Function to calculate average time

void findavgTime(Process proc[], int n)

{

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

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

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

// Calculate total waiting time and

// total turnaround time

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

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

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

printf("   %d ",(i+1));

printf("       %d ", bt[i] );

printf("       %d",wt[i] );

printf("       %d\n",tat[i]

}

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

// Driver code

int main()

{

Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },

{ 3, 7, 2 }, { 4, 3, 3 } };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

**Code 3( Question 21):**

#include <types.h>

#include <lib.h>

#include <test.h>

#include <thread.h>

#include <synch.h>

extern int initialize\_bowls(unsigned int bowlcount);

extern void cat\_eat(unsigned int bowlnumber);

extern void mouse\_eat(unsigned int bowlnumber);

extern void cat\_sleep(void);

extern void mouse\_sleep(void);

int NumBowls; // number of food bowls

int NumCats; // number of cats

int NumMice; // number of mice

int NumLoops; // number of times each cat and mouse should eat

struct semaphore \*CatMouseWait;

static void cat\_simulation(void \* unusedpointer, unsigned long catnumber)

{

int i;

unsigned int bowl;

(void) unusedpointer;

(void) catnumber;

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

cat\_sleep();

bowl = ((unsigned int)random() % NumBowls) + 1;

cat\_eat(bowl);

}

V(CatMouseWait);

}

static void mouse\_simulation(void \* unusedpointer, unsigned long mousenumber)

{

int i;

unsigned int bowl;

(void) unusedpointer;

(void) mousenumber;

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

mouse\_sleep();

bowl = ((unsigned int)random() % NumBowls) + 1;

mouse\_eat(bowl);

}

V(CatMouseWait);

}

int catmouse(int nargs, char \*\* args)

{

int index, error;

int i;

if (nargs != 5) {

kprintf("Usage: <command> NUM\_BOWLS NUM\_CATS NUM\_MIC NUM\_LOOPS\n");

return 1; // return failure indication

}

NumBowls = atoi(args[1]);

if (NumBowls <= 0) {

kprintf("catmouse: invalid number of bowls: %d\n",NumBowls);

return 1;

}

NumCats = atoi(args[2]);

if (NumCats < 0) {

kprintf("catmouse: invalid number of cats: %d\n",NumCats);

return 1;

}

NumMice = atoi(args[3]);

if (NumMice < 0) {

kprintf("catmouse: invalid number of mice: %d\n",NumMice);

return 1;

}

NumLoops = atoi(args[4]);

if (NumLoops <= 0) {

kprintf("catmouse: invalid number of loops: %d\n",NumLoops);

return 1;

}

kprintf("Using %d bowls, %d cats, and %d mice. Looping %d times.\n",

NumBowls,NumCats,NumMice,NumLoops);

CatMouseWait = sem\_create("CatMouseWait",0);

if (CatMouseWait == NULL) {

panic("catmouse: could not create semaphore\n");

}

if (initialize\_bowls(NumBowls)) {

panic("catmouse: error initializing bowls.\n");

}

for (index = 0; index < NumCats; index++) {

error = thread\_fork("cat\_simulation thread",NULL,index,cat\_simulation,NULL);

if (error) {

panic("cat\_simulation: thread\_fork failed: %s\n", strerror(error));

}

}

for (index = 0; index < NumMice; index++) {

error = thread\_fork("mouse\_simulation thread",NULL,index,mouse\_simulation,NULL);

if (error) {

panic("mouse\_simulation: thread\_fork failed: %s\n",strerror(error));

}

}

for(i=0;i<(NumCats+NumMice);i++) {

P(CatMouseWait);

}

sem\_destroy(CatMouseWait);

return 0;

}