RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	19CSC304A
Course Name	Operating Systems
Programme	B. Tech
Department	Computer Science and Engineering
Faculty	Engineering and Technology

Name of the Student	Deepak R
Reg. No	18ETCS002041
Semester/Year	5th/2020
Course Leader/s	Ms. Jishmi Jos Choondal

<table>
<tr><td colspan="5">Declaration Sheet</td></tr>
<tr><td>Student Name</td><td colspan="4">Deepak R</td></tr>
<tr><td>Reg. No</td><td colspan="4">18ETCS002041</td></tr>
<tr><td>Programme</td><td colspan="2">B. Tech</td><td>Semester/Year</td><td>5th/2020</td></tr>
<tr><td>Course Code</td><td colspan="4">19CSC304A</td></tr>
<tr><td>Course Title</td><td colspan="4">Operating Systems</td></tr>
<tr><td>Course Date</td><td></td><td>to</td><td colspan="2"></td></tr>
<tr><td>Course Leader</td><td colspan="4">Ms. Jishmi Jos Choondal</td></tr>
<tr><td colspan="5">Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.</td></tr>
<tr><td>Signature of the Student</td><td colspan="2"></td><td>Date</td><td>05/12/2020</td></tr>
<tr><td>Submission date stamp
(by Examination & Assessment Section)</td><td colspan="4"></td></tr>
<tr><td colspan="2">Signature of the Course Leader and date</td><td colspan="3">Signature of the Reviewer and date</td></tr>
<tr><td colspan="2"></td><td colspan="3"></td></tr>
</table>

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. in CSE
Semester/Batch	5th / 2020		
Course Code	19CSC304A	Course Title	Operating Systems
Course Leader(s)	Ms. Jishmi Jos Choondal/Ms. Naveeta		

Assignment			
Register No.	18ETCS002041	Name of Student	DeepakR

Sections		Marking Scheme	Max Marks	First Examiner Marks	Moderator Marks
Question 1	Q1.1	Introduction to multi-programming	01		
	Q1.2	Effect of multi-programming on CPU utilisation	04		
		Question 1 Max Marks	05		
Question 2	Q2.1	Design and implementation of the application using sequential approach with functions	04		
	Q2.2	Design and implementation of the application using multithreaded approach	04		
	Q2.3	Comparison of the execution time of the above two versions of the program and its analysis	02		
		Question 2 Max Marks	10		

Q u	Q3.1	Schedule of the processes using a Gantt chart	04		
	Q3.2	Average waiting time and average turnaround time experienced	04		
	Q3.3	Scheduling algorithm with better performance and its justification	02		
		Question 3 Max Marks	10		
		Total Assignment Marks	25		

Course Marks Tabulation				
Component- 1(B) Assignment	First Examiner	Remarks	Moderator	Remarks
Q1				
Q2				
Q3				
Marks (out of 25)				

Signature of First Examiner Signature of Second Examiner

Solution For Part A

The effect of multi-programming on CPU utilisation

Multiprogramming is one of the OS architecture that increases the CPU utilization by organizing jobs such that CPU always have one to execute.Multiprogrammed system provides environment for efficient utilization of resources but they do not provide user interaction with the computer system. Multitasking is the logical extension of multiprogramming in which CPU executes multiple jobs by switching but switches occur so frequently that users can interact with each program while it is running.

Example: Use a browser, play video, download apps and transfer data at the same time. In actual all process are working one at a time on the processor. Switching between process/program is so fast that we will never notice. our CPU efficiency is always measured in GHz. RAM is also required for switching. That's why we want a device with large RAM memory and high CPU GHz.

This increase the efficiency of devices.

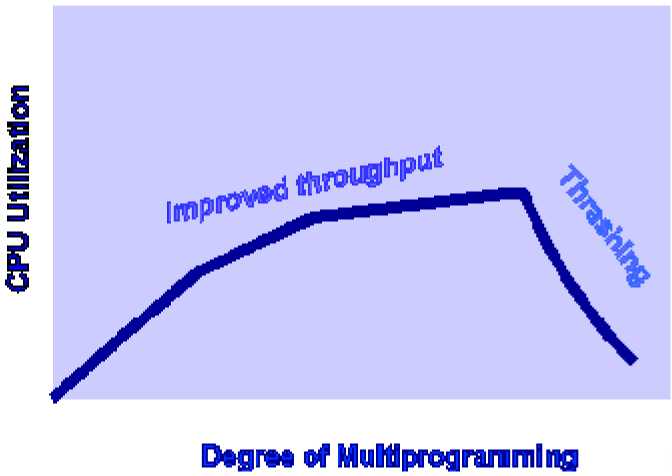

Fig Drawn Using Paint Tool

Programs in a multiprogrammed environment appear to run at the same time. Processes running in a multiprogrammed environment are called concurrent processes. In actuality, the CPU processes one instruction at a time, but can execute instructions from any active process.

The CPU utilization is (1 - P^N) where N is called the multiprogramming level (MPL) or the degree of multiprogramming. As N increases, the CPU utilization increases. While this equation indicates that a CPU continues to work more efficiently as more and more processes are added, logically, this cannot be true. Once the system passes the point of optimal CPU utilization, it thrashes.

In a single-core processor, the performance of the CPU is limited by the time taken to communicate with cache and RAM. Approximately 75% of CPU time is used waiting for memory access results. To improve the performance of their processors, manufacturers have been releasing more multi-core machines. A CPU that offers multiple cores may perform significantly better than a single-core CPU of the same speed.

As the above Graph shows, CPU utilization of a system can be improved by using multiprogramming.

Multiprogramming systems, an entire program was loaded into its own block of memory, called its *memory partition*. These early systems implemented *multiprogramming with fixed partitions*. As the size of programs grew, it became difficult to find partitions large enough to accomodate programs and still have the multiprogramming level high enough to produce a good CPU utilization.

In order to achieve a high CPU utilization with larger programs, multiprogrammed systems were combined with virtual memory. This combination allows the selection of a partition which is smaller than the address space of the program. It relies on a paging policy to manage the contents of the partition.

Solution for Question 2

Here is the code for Multi-threaded version:

```java
package multiThreaded;

/**
 *
 * @author Deepak R
 *
 * This class takes in a  integer array and adds it's contents. This
 * addition will be concurrent between several threads which will divide
 * the work of the array based on the threadID assigned to thread by the
 * programmer. Assume that the passed in  array to the constructor is a
 * matrix with each array in the main array having same length.
 */

public class ArraySum2D implements Runnable{

private int[][] arrayToSum;
private int threadID;
private int totalSum;

public ArraySum2D(int[][] arr, int threadID){
    this.arrayToSum = arr;
    this.threadID = threadID;
    this.setTotalSum(0);
}

@Override
public void run() {
    int arrayCol = arrayToSum[0].length;
    int arrayRow = arrayToSum.length;
    int colStart = (int)((threadID%2) * (arrayCol/2));
    int rowStart = (int)((int)(threadID/2) * (arrayRow/2));
    int colEnd = colStart + (int)(arrayCol/2);
    int rowEnd = rowStart + (int)(arrayRow/2);

    for(int i = colStart; i < colEnd; i++){
        for(int j = rowStart; j < rowEnd; j++){
            setTotalSum(getTotalSum() + arrayToSum[j][i]);
        }
    }
}

public int getTotalSum() {
    return totalSum;
}

public void setTotalSum(int totalSum) {
    this.totalSum = totalSum;
}

}
```

Next page code for Sequential Version

Here is the code for Sequential version:

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package sequentialNonMT;

/**
 *
 * @author Deepak R
 */
public class ArraySum2DNonMT {

    private int[][] arrayToSum;
    private int totalSum;

    public ArraySum2DNonMT(int[][] arr){
        this.arrayToSum = arr;
        this.setTotalSum(0);
    }

    public void runSequential(){
        for(int i = 0; i < arrayToSum[0].length; i++){
            for(int j = 0; j < arrayToSum.length; j++){
                setTotalSum(getTotalSum() + arrayToSum[j][i]);
            }
        }
    }

    public int getTotalSum() {
        return totalSum;
    }

    public void setTotalSum(int totalSum) {
        this.totalSum = totalSum;
    }

}
```

Next page code for Main Driver Class →

Here is the Code for Main Driver Class

```java
package controller;

import java.util.Random;

import multiThreaded.ArraySum2D;
import sequentialNonMT.ArraySum2DNonMT;

public class ControllerMain {

private final static int cols = 5000;
private final static int rows = 5000;
private static volatile int[][] arrayToAdd = new int[rows][cols];
private static Random rand = new Random();
private static ArraySum2D a0, a1, a2, a3;

public static void main(String[] args) throws InterruptedException{

    for(int j = 0; j < rows; j++){
        for(int i = 0; i < cols; i++){
            arrayToAdd[j][i] = rand.nextInt(100);
        }
    }

    ArraySum2DNonMT a = new ArraySum2DNonMT(arrayToAdd);

    long startTimeSequential = System.nanoTime();
    a.runSequential();
    long estimatedTimeSequential = System.nanoTime() - startTimeSequential;


    System.out.println("The total time taken by sequential program in nano Seconds is: " + estimatedTimeSequential);

    a0 = new ArraySum2D(arrayToAdd, 0);
    a1 = new ArraySum2D(arrayToAdd, 1);
    a2 = new ArraySum2D(arrayToAdd, 2);
    a3 = new ArraySum2D(arrayToAdd, 3);
    Thread t0 = new Thread(a0);
    Thread t1 = new Thread(a1);
    Thread t2 = new Thread(a2);
    Thread t3 = new Thread(a3);

    long startTimeMultiThreaded = System.nanoTime();
    t0.start();
    t1.start();
    t2.start();
    t3.start();

    t0.join();
    t1.join();
    t2.join();
    t3.join();
    int Sum = addThreadSum();
    long estimatedTimeMultiThreaded = System.nanoTime() - startTimeMultiThreaded;


    System.out.println("The total time taken by multi threaded program in nano seconds is: " + estimatedTimeMultiThreaded);
}

private static int addThreadSum(){
    return a0.getTotalSum() + a1.getTotalSum() + a2.getTotalSum() + a3.getTotalSum();
}

}
```

Test Cases

Test Case 1= 5000*5000

Output - controller (run)

```
run:
The total time taken by sequential program in nano Seconds is: 298614700
The total time taken by multi threaded program in nano seconds is: 135755200
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Case 2= 7000*7000

Output - controller (run)

```
run:
The total time taken by sequential program in nano Seconds is: 576813200
The total time taken by multi threaded program in nano seconds is: 236657500
BUILD SUCCESSFUL (total time: 1 second)
```

Test Case 3= 8000*8000

Output - controller (run)

```
run:
The total time taken by sequential program in nano Seconds is: 751047000
The total time taken by multi threaded program in nano seconds is: 359326900
BUILD SUCCESSFUL (total time: 2 seconds)
```

Test Case 4 = 7500*7500

Output - controller (run)

```
run:
The total time taken by sequential program in nano Seconds is: 640541300
The total time taken by multi threaded program in nano seconds is: 265634200
BUILD SUCCESSFUL (total time: 2 seconds)
```

Test Case 5 = 10000*10000

Output - controller (run)

```
run:
The total time taken by sequential program in nano Seconds is: 1324994000
The total time taken by multi threaded program in nano seconds is: 595365300
BUILD SUCCESSFUL (total time: 3 seconds)
```

	Sequential Version in ns	**Multithreaded Version in ns**
Test Case 1	**298614700**	**125755200**
Test Case 2	**576813200**	**236657500**
Test Case 3	**751047000**	**359326900**
Test Case 4	**640541300**	**265634200**
Test Case 5	**1324994000**	**595365300**
Average	**718402040**	**316547820**

Environment

Processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz

Installed RAM 8.00 GB (7.89 GB usable)

System type 64-bit operating system, x64-based processor

Runtime is O(nm), where *n* is the row size, and *m* is the column size.

Analysis

Runtime is O(nm), where *n* is the row size, and *m* is the column size.

When we go through the above table we will get to know that multithreaded Version take less time to Complete given task A multithreaded program can finish faster than a sequential one, because some of the work it does can proceed simultaneously. Multithreading allows placing certain tasks in different threads so that they don't interrupt each other. Because of that, tasks don't 'wait' for the access to the resources that they need, they share those resources. Moreover, this allows separating heavy operations (like data processing) from the main app tasks (like interface performance). Because of that, our interface can work faster

Solution for Question 3

Given

Processes	Burst Time (ns)	Arrival Time (ns)	Priority
P1	10	15	6
P2	15	20	8
P3	5	25	2
P4	12	10	4

Table 1

We Know that

3.1 Schedule of the processes using a Gantt chart
Preemptive Priority Scheduling

-	P4	P1	P2	P1	P4	P3
0	10 15	20	35	40	47	52

Non Preemptive Priority Scheduling

-	P4	P2	P1	P3
0	10 22	37	47	52

3.2 Average waiting time and average turnaround time experienced
Preemptive Priority Scheduling

Processes	Burst Time (ns)	Arrival Time (ns)	Priority	Competion time (ns)	Waiting Time(ns)	Turnaround time (ns)
P1	10	15	6	40	15	25
P2	15	20	8	35	0	15
P3	5	25	2	52	22	27
P4	12	10	4	47	25	37

Tr **is Turn around time = Waiting Time + Service time**

Waiting time = Turnaround time -Burst time
Waiting time for P1 = 25-10=15ns
Waiting time for P2 = 15-15=0ns
Waiting time for P3 = 27-5=22ns
Waiting time for P4 = 37-12=25ns

Average Waiting time = (Sum of Waiting time of P1+P2+P3+P4)/4
=(15+0+22+25)/4 = 15.5ns
So Average Waiting time is 15.5ns

Turn around time = Completion time - Arrival time
Turn around time for P1 = 40-15=25ns
Turn around time for P2 = 35-20=15ns
Turn around time for P3 = 52-25=27ns
Turn around time for P4 = 47-10=37ns
Average Turn around time = (Sum of Turnaround time of P1+P2+P3+P4)/4
=(25+15+27+37)/4 = 26.00ns
So Average Waiting time is 26.00ns

Processes	Burst Time (ns)	Arrival Time (ns)	Priority	Competion time (ns)	Waiting Time(ns)	Turnaround time (ns)
P1	10	15	6	47	22	32
P2	15	20	8	37	2	17
P3	5	25	2	52	22	27
P4	12	10	4	22	00	12

Waiting time = Turnaround time -Burst time
Waiting time for P1 = 32-10=22ns
Waiting time for P2 = 17-15=2ns
Waiting time for P3 = 27-5=22ns
Waiting time for P4 = 12-12=00ns
Average Waiting time = (Sum of Waiting time of P1+P2+P3+P4)/4
=(22+2+22+00)/4 = 11.5ns
So Average Waiting time is 11.5ns

Turn around time = Completion time - Arrival time
Turn around time for P1 = 47-15=32ns
Turn around time for P2 = 37-20=17ns
Turn around time for P3 = 52-25=27ns
Turn around time for P4 = 22-10=12ns
Average Turn around time = (Sum of Turnaround time of P1+P2+P3+P4)/4
=(32+17+27+12)/4 = 22.00ns
So Average Waiting time is 22.00ns

3.3 Scheduling algorithm with better performance and its justification

For Preemptive Scheduling Algorithm Tr/Ts Ratio =Relative delay experienced by a process

For P1 = 25/10 =2.50
For P2 = 15/15 =1.00
For P3 = 27/5 =5.40
For P4 = 37/12 =3.08

So By Taking Mean of all Processes Relative Delay (2.50+1.00+5.40+3.08)/4 = 11.98/4=2.995

So 2.995 is the Required Mean of Relative delays of all Processes in Preemptive Scheduling.

For Non-Preemptive Scheduling Algorithm Tr/Ts Ratio =Relative delay experienced by a process

For P1 = 32/10 =3.20
For P2 = 17/15 =1.13
For P3 = 27/5 =5.40
For P4 = 12/12 =1.00

So By Taking Mean of all Processes Relative Delay (3.20+1.13+5.40+1.00)/4 = 10.73/4= 2.682

So 2.682 is the Required Mean of Relative delays of all Processes in Non-Preemptive Scheduling.

Increasing values of Relative delay correspond to a decreasing level of service So in the Above Problem Non–Preemptive Scheduling Algorithm is More Efficient because of Comparably Less Relative Delay Ratio when Calculated By taking Mean of Each Processes.

References

Operating System Concepts (2012) by Silberschatz, Galvin and Gagne.