**CS 3343 Operating Systems Assignment 4 14 points**

**Due February 26 at 5pm**

**One homework assignment submission per student. Microsoft Word format only. No AI or GPT use. Cite all references.**

**Email your answers to me at** [**harringp@nsuok.edu**](mailto:harringp@nsuok.edu)

**Send emails exclusively via** [**https://mail.google.com/**](https://mail.google.com/)

**Sign in with your NSU email and password**

**Chapter 4:**

**Part 1 : Use the slides or textbook definitions to answer the following (7 points)**

1. What are the four benefits of multithreaded programming? **(Book: 162)**

Responsiveness, Resource Sharing, Economy, and Scalability are the four benefits of multithreaded programming.

Responsiveness – multithreading may allow a program to continue running even if it is blocked or performing a lengthy operation. This increases responsiveness for the user.

Resource Sharing – threads allow for the sharing of memory and resources by default.

Economy – because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.

Scalability – the benefits can be even greater in a multiprocessor architecture, where threads may be running in parallel on different cores.

1. What is a thread library?  Name two ways of implementing a thread library. **(Book: 168)**

A thread library provides the programmer with an API for creating and managing threads.

The two primary ways include:

* Providing a library entirely in user space with no kernel support. All code and data structures exist in the user space.
* Implement a kernel-level library supported directly by the operating system. In this case, code and data structures exist in kernel space.

1. What are kernel threads and user threads? **(Book: 166)**

Support for threads may be provided either at the user level or by the kernel.

* Kernel Threads are supported and managed directly by the operating system.
* User Threads are supported above the kernel and are managed without kernel support.

1. Describe the three multithreaded models. **(Book: 166-168)**

Many-to-One – Many user-level threads get mapped to one kernel thread. In this system however, the entire process will block if a thread makes a blocking system call. Also, multiple threads are unable to run in parallel on multicore systems.

One-to-One – each user thread is mapped to a kernel thread. This provides more concurrency by allowing another thread to run when a thread makes a blocking system call. It also allows multiple threads to run in parallel on multiprocessors. Linux, along with the Windows OS, implement the one-to-one model.

Many-to-Many – multiplexes many user-level threads to a smaller or equal number of kernel threads. These kernel threads may be specific to either a particular application or a particular machine. Although this appears to be the most flexible, it is difficult to implement.

1. What is thread cancellation of a target thread?  In what two scenarios may this occur? **(Book: 190)**

A thread that is to be cancelled is often referred to as “the target thread”. The two scenarios in which this may occur is asynchronous cancellation and deferred cancellation.

Asynchronous – one thread immediately terminates the target thread

Deferred – target thread periodically checks whether it should terminate, allowing the opportunity to terminate itself.

1. What is a signal in the Unix OS and in what way can this be delivered to threads within a process? **(Book: 188)**

A signal is used to notify a process that a particular event has occurred. In a single threaded program, signals are always delivered to a process. For multithreaded applications however, it becomes more difficult. There are four options according to the book. These include delivering the signal to the thread to which the signal applies, delivering the signal to every thread in the process, delivering the signal to certain threads in the process, or assigning a specific thread to receive all signals for the process.

1. What is a lightweight process and how is it utilized in a multi-threaded OS? **(Book: 193)**

A lightweight process is an intermediate data structure between the user and kernel threads. This is done in many systems implementing the many-to-many or the two-level model.   
  
To the user-thread library, the lightweight process appears to be a virtual processor on which the application can schedule a user thread to run. Each lightweight process is attached to a kernel thread, of which it is the kernel threads that the OS schedules to run on physical processors. If a kernel thread blocks, the lightweight process blocks which in turn blocks the user-thread attached to the lightweight process. Typically, a lightweight process is required for each concurrent blocking system call.

**Part 2: Java programming (7 points):**

Modify and combine the two java files in “cs 3343 java files assignment 4.zip”: MultiThreadedAddition.java and TPExample.java

**I’m not sure if I am on the right track or not, but from what I gathered, both the MultiThreadedAddition and TPExample files had similar makeups. I’m still not entirely sure how the MultiThreadedAddition manages each thread, but I do know that in TPExample it uses a pool. I wasn’t quite sure how to implement the pool into the MultitThreadedAddition, but I tried multiple methods. In the end, I decided to just pull bits and pieces from the TPExample file to get what I now have in the MultiThreadedAddition file.   
  
I would appreciate feedback about the pool system, it seems really interesting and I would like to know more about it. I would’ve liked to implement it in the file, but I couldn’t get it to work properly. There is probably just a few lines I am missing somewhere along the line to get the answer I wanted in the email I sent.**

import java.util.Scanner;

public class MultiThreadedAddition

{

    public static void main(String[] args)

    {

        Summation[] worker;

        Sum[] sumValue;

        Scanner sc = new Scanner(System.in);

        Integer total = 0;

        Integer maximum;

        int number;  //number of threads

        int lower,upper;

        System.out.print("Enter an integer value: ");

        maximum = sc.nextInt();

        System.out.print("Enter the number threads: ");

        number = sc.nextInt();

        System.out.println();

        sc.close();

        if(number < maximum && number > 0)

        {

            worker = new Summation[number]; //create threads

            sumValue = new Sum[number];

            lower = 1;

            int increment = maximum/number;

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

            {

               upper = lower + increment;

               if(upper > maximum)

                  upper = maximum;

               sumValue[i] = new Sum();

               worker[i] = new Summation(lower, upper,sumValue[i]);

               Integer name = i+1;

               worker[i].setName(name.toString());

               worker[i].start();

               lower = upper + 1;

            }

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

            {

                try

                {

                    worker[i].join();

                }

                catch(InterruptedException e){System.out.println(e.toString());}

                //FROM FOR LOOP 3

                //I took away the 3rd for loop because I deemed it unnessecary.

                total += sumValue[i].get();

            }

            System.out.println("SUM OF NUMBERS BETWEEN 1 AND " + maximum.toString() + " = " + total.toString());

            Integer checkSum;

            checkSum = maximum\*(maximum+1)/2;

            System.out.println("CHECK SUM = " + checkSum.toString());

        }

        else

        {

            System.out.println("Number of threads must be less than maximum integer value and greater than zero.");

        }

    }

}

//I moved the classes down here, because I prefer the main to be up top and easily visible

class Sum

{

    private int value;

    public int get()

    {

        return value;

    }

    public void set(int sum)

    {

        this.value = sum;

    }

}

class Summation extends Thread

{

    private int lower,upper;

    private int sum;

    private Sum sumValue;

    //I changed the variables here in the Summation class.

    //I got rid of all instances of "this" that you mentioned in class on Wednesday

    public Summation(int lowerNum,int upperNum,Sum sumValueNum)

    {

        lower = lowerNum;

        upper = upperNum;

        sumValue = sumValueNum;

    }

    public int getSum()

    {

        return sum;

    }

    public void run()

    {

        int sum = 0;

        for (int i = lower; i <= upper; i++)

            sum += i;

        sumValue.set(sum);

        //This is the data I pulled from the TPExample java file.

        //I modified it a bit to flow and read better in the console.

        Thread me;

        me = Thread.currentThread();

        System.out.println("-- I am working on a task--" + "\n[Thread] = " + me.getName() + "\t[Sum] = " + sum + "\n");

    }

}

**Run 1: Run 2:**

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated