**CS 3343 Operating Systems Assignment 6 12 points**

**Due April 3 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 6**

**Part 1 : Answer the following (7 points)**

1. What are the three parts of the solution to the critical section problem? **(Book 260-261)**
2. Mutual Exclusion – if a process P, is executing in its critical section, then no other processes can be executing in their critical sections.
3. Progress – If no process is executing in its critical section and some processes wish to enter critical sections, then only the processes that are not executing in their remainder sections can participate in deciding which will enter critical sections next.
4. Bounded Waiting – For this, there exists a bound/limit on the number of times that other processes are allowed to enter critical sections after a process has made a request to enter its critical section and before that request is granted.
5. What are the two types of semaphores and how do they provide a solution to the critical section problem? **(Book 273-274)**

We have counting semaphores and binary semaphores. Semaphores can be used to solve various synchronization problems. They can also be used to control access to a given resource consisting of a finite number of instances. Both of these things can help with the critical section problem.

1. Define deadlock and starvation. **(Book 283 & Ch6 Slides 23)**

Deadlock – a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.

Starvation – Indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.

1. Describe the bounded-buffer problem. **(Ch6 Slides 24-25)**

It is a classic problem of synchronization and it occurs when according to the slides…

1. N buffers, each can hold one item
2. Semaphore mutex initialized to the value 1
3. Semaphore full initialized to the value 0
4. Semaphore empty initialized to the value N
5. Describe the readers-writers problem. **(Ch6 Slides 28)**

* Readers can only read the data set. They do not perform any updates.
* Writers can both read and write.

The problem that can occur though is when multiple readers are allowed to read at the same time. Only one single writer can access the shared data at the same time.

The shared data on the slides includes…

1. Data set
2. Semaphore mutex initialized to 1
3. Semaphore wrt initialized to 1
4. Integer readcount initialized to 0
5. Describe the Windows 64 bit API function calls. (<https://learn.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-170>)

I’m not entirely sure if I am on the right track for this question, but I found this page from the Microsoft docs on the x64 calling convention. It says that the x64 ABI uses a four-register fast-call convention by default. It also states that there is a strict one-to-one correspondence between a function call’s arguments and the registers used for those arguments. There is a ton more data about parameter passing, return values, function pointers, etc. included in the page as well.

1. Compare and contrast the two-phase locking protocol and the timestamp protocol. (Ch6 Slides 62-63)

The two-phase locking protocol can ensure conflict serializability. It does not prevent deadlock, and each transaction issues lock and unlock requests in two phases.

* These two phases being Growing and Shrinking
* Growing involves obtaining locks and shrinking involves releasing locks

Timestamp locking protocols on the other hand will select the order in advance. This is known as timestamp-ordering. Timestamps will determine the serializability order. The timestamp locking protocol ensures that any conflicting read and write is executed in timestamp order.

* The algorithm according to the slides ensures conflict serializability and gives freedom from deadlock.

**Part 2: Java programming (5 points):**

Change the java code below to handle interrupted exceptions, and also to input the integer to multiply by sleeptime. Include screenshots of your running program.

public class SleepUtilities

{

/\*\* \* Nap between zero and NAP\_TIME seconds. \*/

public static void nap() {

nap(NAP\_TIME);

}

/\*\* \* Nap between zero and duration seconds. \*/

public static void nap(int duration) {

int sleeptime = (int) (NAP\_TIME \* Math.random() );

try { Thread.sleep(sleeptime\*1000); }

catch (InterruptedException e) {}

}

private static final int NAP\_TIME = 5;

}

This was the first coding assignment I had real trouble with, I gave it my best shot, but it is all over the place. I can’t seem to get the Thread.sleep working the way I want it to. I assume we need to find a way to print out the Thread.sleep, but I can’t get it to work. I keep running into errors. I will provide what I have with the comments in the code, but I’m just not sure where to go or what to do.

import java.util.Scanner;

public class Assignment6

{

//I tried to get it to run, but I think I have some of my setup wrong. I know I'm missing something, because all this code does that I setup is run the initial nap() function which just includes nap(NAP\_TIME).

public static void main(String[] args)

{

nap();

}

//I'm assuming this allows for the process to nap from zero to the NAP\_TIME seconds which is set to 5 down below.

/\*\* \* Nap between zero and NAP\_TIME seconds. \*/

public static void nap()

{

nap(NAP\_TIME);

}

//This one I'm assuming allows the process to nap from zero to the duration that is given. I am just not entirely sure on how to set it up properly

/\*\* \* Nap between zero and duration seconds. \*/

public static void nap(int duration)

{

int toMultiply;

Scanner in = new Scanner(System.in);

System.out.print("Please enter a number to multiply by sleeptime: ");

toMultiply = in.nextInt();

int sleeptime = (int) (NAP\_TIME \* Math.random());

try

{

//I set it up like the instructions stated, I'm just not sure how to print or show this above. When I use System.out.println(Thread.sleep(sleeptime \* toMultiply)); it runs into errors that I'm not sure how to handle.

Thread.sleep(sleeptime \* toMultiply);

}

//I'm not really sure we needed to change much here for the exception. I may be wrong.

catch (InterruptedException e)

{

System.out.println("Error Occurred: " + e);

}

in.close();

}

private static final int NAP\_TIME = 5;

}