Back to tab

Started on	Wednesday, 22 November 2023, 11:37 AM
State	Finished
Completed on	Wednesday, 22 November 2023, 1:31 PM
Time taken	1 hour 54 mins
Grade	**9.5** out of 10.0 (**95**%)

Question 1

Complete

Mark 1.0 out of 1.0

Explain what it means when a thread **join**s an another thread.

```
Joining with one thread with another means waiting for the other thread to finish its work and the
the next instructions. Here the thread that is joining has to wait for the thread that is doing wo
its work. This is used to achieve synchronization between threads when doing more than one thing a
```

Comment:

Question 2

Complete

Mark 1.0 out of 1.0

1. Explain programming concepts of **Concurrent** and **Parallel**.
2. What is the difference between these concepts?

```
Concurrent programming is a technique of writing programs that can handle multiple tasks at the sa
using threads. Threads are lightweight units of execution that can run independently. These progra
single processor or CPU core by switching between threads very quickly and giving the impression/i
simultaneously. For example,

Parallel programming is the technique to write programs which actually run multiple tasks simultar
CPU cores or processors. This can improve the speed of the program drastically when used correctly
situation, the requirement here is that one task can not depend on another. For example, at an air
the boarding pass of passengers if there is more than one line to check boarding pass then those w
checking parallelly as they do not depend on each other.

The main difference between concurrent and parallel programming is that concurrent programming car
core, but parallel programming requires multiple cores. Parallel programming can give more perform
concurrent programming is more flexible(can used in a wide range of environments).
```

Comment:

Question 3

Complete

Mark 1.0 out of 1.0

Explain what is a **Critical Section**.

A critical section is a piece of code that accesses a part of memory that is shared across multipl
or processes. When more than one process/thread accesses and modifies the same part of memory the
calculations becomes unpredictable and that is called data race.

Comment:

Question 4

Complete

Mark 1.0 out of 1.0

Explain what is a **Race Condition**.

When more than one process/thread accesses and modifies the same part of memory the outcome of the
calculations become unpredictable and that is called a data race.

Comment:

Question 5

Complete

Mark 1.0 out of 1.0

Programming in general, what techniques are available to protect the **critical section** from **parallel access**?

```
There are many techniques available in general to prevent race conditions, some of the techniques
Mutex, Semaphores, Condition Variables, Compare and Swap and atomics.
```

Comment:

Question 6

Complete

Mark 1.0 out of 1.0

Write a pseudocode that will cause a **Deadlock**. Explain the deadlock situation in code comments.

```
mutex 1;
mutex 2;

Thread_1 {
    1.lock();
    sleep(300);
    2.lock();
}

Thread_2 {
    2.lock();
    sleep(300);
    1.lock();
}
```

Comment:

Question 7

Complete

Mark 0.5 out of 1.0

Explain what is an **Atomic operation**.

```
An atomic operation is an instantaneous operation that is acknowledged when it is asked to happen.
The requested operation happens without any interruption from other concurrent processes/threads.
basic operations are allowed but complex operations can be done through simple instructions.
```

Comment: Not necessarily instantaneous, but always indivisible.

Question 8

Complete

Mark 1.0 out of 1.0

1. Explain the concept of **lock-free** programming.
2. What are the pros and cons of lock free programming?

```
Lock-free programming means to not use locks when trying to synchronize concurrent programs. This
unnecessary waiting between tasks. This can be achieved using compare and swap and atomics.

Pros:
- Less overhead/waiting
- Improved speed
- Improved scalability

Cons:
- Harder to implement and debug
- Can cause memory leaks
- This can handle simple data structures and operations better
- Not all platforms supports the semantics
```

Comment:

Question 9

Complete

Mark 1.0 out of 1.0

The data structure below stores two integers X and Y in a way that
it is possible to increment both variables in one function call. Struct Coordinate is a sequential implementation of the data structure:

```
struct Coordinate
{
   int X;
   int Y;
   int getX() { return X; }
   int getY() { return Y; }
   public void incXY()
   {
      // increment X and Y
      X = getX() + 1;
      Y = getY() + 1;
   }
}
```

Questions:
* Explain why the above implementation of Coordinate is not thread safe.
* Describe a concrete scenario where race conditions may occur.

```
The above implementation of Coordinate is not thread-safe since the read and write can happen at t
To be more specific write(incXY) operations can be done from multiple threads at the same time.

Race conditions will occur when one thread is reading data and more than one thread is writing dat
The result will be inconsistent because the more than one thread is increasing the XY will read th
value assume it's 5, then they will make it 6 but in reality, it should have been more than that,
incrementing it should have been 7.
```

Comment:

Question 10

Complete

Mark 1.0 out of 1.0

Consider traditional producer-consumer problem:

producer	**consumer**
loop forever	loop forever
1. product = produce()	1. product = warehouse.get()
2. warehouse.put(product)	2. consume(d)

Implement (in pseudocode) the **warehouse** with methods **get** and **put**. Implementation must take into an account that there can be multiple parallel producers and consumers running.

What synchronization method you chose? Why?

```
number_of_producers = x;
number_of_consumers = y;
warehouse_mtx: mutex;
warehouse_cv: condition_variable;
warehouse: queue = [];
shipping_house_mtx: mutex;
shipping_house: queue = [];


Producer{
    while(true){
        product = await produce() // waits until the product is produced
        {
            lock_guard(warehouse_mtx)
            warehouse.put(product)
        }
        warehouse_cv.notify_one()
    }
}
```

Comment:

◄ Course Feedback (Peppi)

Jump to...