MULTI-THREAD ASSIGNMENT

1. Concurrency vs. Parallelism:

Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

For example: You are doing one thing, another thing comes.

If before finishing the first thing, you cannot deal with another thing. That means that you do not support Concurrency or Parallelism.

If you continue to finish another thing then get back to finish the first thing. That means that you support Concurrency.

If you are doing both things together. That means that you support Parallelism.

2. Process vs. Thread

Process means any program is in execution. Process control block controls the operation of any process. Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc. A process can creates other processes which are known as Child Processes. Process takes more time to terminate and it is isolated means it does not share memory with any other process. The process can have the following states like new, ready, running, waiting, terminated, suspended.

Thread: Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread has 3 states: running, ready, and blocked. Thread takes less time to terminate as compared to process and like process threads do not isolate.

3. Synchronization and race

The locking of an object so that others cannot access it, called synchronization.

A “race condition” arises if two threads try to read and write the same data.

4. Mutual exclusion critical section and lock

Mutex (English: Mutual Exclusion, abbreviation Mutex) is for multi-threaded programming in preventing two threads at the same time on the same public resources (such as global variables mechanism) read and write. This object is sliced into a code of a critical region (critical section) reached. Critical region refers to a piece of code that accesses public resources, not a mechanism or algorithm. A program, process, or thread can have multiple critical regions, but mutual exclusion locks are not necessarily applied.

Examples of resources that require this mechanism are: flags, queues, counters, interrupt handlers, and other resources used to transfer data, synchronization status, etc. between multiple parallel running codes. Maintaining the synchronization, consistency and integrity of these resources is very difficult, because a thread may be suspended (sleep) or resume (awakened) at any time.

For example: a piece of code (A) is modifying a piece of data step by step. At this time, another thread (B) was awakened for some reason. If B goes to read the data that A is modifying at this time, and A happens to have not completed the entire modification process, the state of this piece of data at this time is in a very uncertain state, and of course the read data is also problematic. of.

A more serious situation is that B also writes data to this place. In this way, the consequences will become uncontrollable. Therefore, data shared between multiple threads must be protected. The way to achieve this is to ensure that only one critical area is in operation at the same time, and other critical areas, whether they are reading or writing, must be suspended and cannot get the opportunity to operate.

5. Condition synchronization

Condition Synchronization (or merely synchronization) is any mechanism that protects areas of memory from being modified by two different threads at the same time.

6. Atomicity

Atomic operations are those operations that ALWAYS execute together. Either all of them execute together, or none of them executes. If an operation is atomic, then it cannot be partially complete, either it will be complete, or not start at all, but will not be incomplete.

i=0; This is an atomic operation since this will either happen all together, assigning 0 to i, or it won't happen at all.

i=i+1; it is not an atomic operation. This one-liner actually consists of three operations.

Read operation, where the value of i is read.

Modify operation, where a new value is being calculated (i + 1).

Write operation, where the new value is written to the variable i.

If we call this method on two threads at the same time, then thread A and B will both read the value at the same time and update the value at the same time as well.

The most commonly used atomic classes are – AtomicInt, AtomicLong, AtomicBoolean, and AtomicReference. All of these provide atomic operations for the respective classes. AtomicReference can be used for just any type of object.

What incrementAndGet() does is atomically increment the value by 1, and then returns the updated value. If the program is now run in a multi-threaded environment, supposing with 2 threads, then the end result will always be i being equal to 2. This is because no matter which thread gets to the incrementAndGet() method first, since it is an atomic operation, the thread will update the value of i to 1, and only then another thread will be able to access or update it, which will make the value of i to 2, thus giving us the correct result.

7. Dead Lock

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.

8. Producer and Consumer model

A baker produces breads and puts them on the shelf, like a queue.

Customers take them off the shelf.

¨ Threads A: produce loaves of bread and put them in the queue

¨ Threads B: consume loaves by taking them off the queue

This is the producer/consumer model.