Monitors vs Semaphores

What is the difference between a monitor and semaphore?

The reason that semaphores and monitors are needed is because multi-threaded applications (like Microsoft Word, Excel, etc) must control how threads access shared resources. This is known as **thread synchronization** – which is absolutely necessary in a multi-threaded application to ensure that threads work well with each other. If applications do not control the threads then it may result in corruption of data and other problems.

Do I use a monitor or a semaphore?

Monitors and semaphores are both programming constructs used to accomplish thread

synchronization.
Whether you use a monitor or a semaphore depends on what your language or system supports.

What is a Monitor?

A monitor is a **set of multiple** routines which are protected by a mutual exclusion lock. None of the routines in the monitor can be executed by a thread until that thread acquires the lock. This means that only ONE thread can execute within the monitor at a time. Any other threads must wait for the thread that's currently executing to give up control of the lock.

However, a thread can actually suspend itself inside a monitor and then wait for an event to occur. If this happens, then another thread is given the opportunity to enter the monitor. The thread that was suspended will eventually be notified that the event it was waiting for has now occurred, which means it can wake up and reacquire the lock.

What is a Semaphore?

A semaphore is a simpler construct than a monitor because it's just a lock that protects a shared resource – and not a set of routines like a monitor. The application must acquire the lock before using that shared resource protected by a semaphore.

Example of a Semaphore – a Mutex

A mutex is the most basic type of semaphore, and mutex is short for mutual exclusion.

In a mutex, only one
thread can use the shared resource at a time. If another thread wants to use the shared resource, it must wait for the owning thread to release the lock.

Differences between Monitors and Semaphores

Both Monitors and Semaphores are used for the same purpose – thread synchronization. But, monitors are simpler to use than semaphores because they handle all of the details of lock acquisition and release. An application using semaphores has to release any locks a thread has acquired when the application terminates – this must be done by the application itself. If the application does not do this, then any other thread that needs the shared resource will not be able to proceed.

Another difference when using semaphores is that every routine accessing a shared resource has to explicitly acquire a a lock before using the resource. This can be easily forgotten when coding the routines dealing with multithreading . Monitors, unlike semaphores, automatically acquire the necessary locks.

Is there a cost to using a monitor or semaphore?

Yes, there is a cost associated with using synchronization constructs like monitors and semaphores. And, this cost is the time that is required to get the necessary locks whenever a shared resource is accessed.

0 Comments   Programmer Interview   1 Login

Recommend 6   Share   Sort by Best

Start the discussionâ€¦

Be the first to comment.

ALSO ON **PROGRAMMER INTERVIEW**

Javascript Interview Questions

1 comment • a year ago•

Drake Willmannson — As this website has more ads than content, a JS code to hide ads (including disqus), execute this in browser

What's the difference between a compiled and an interpreted language?

1 comment • a year ago•

shuyu — easy to understand

Find continuous sequence with largest sum

1 comment • a year ago•

Uba Chan — Its not the correct solution . try this input [21,2,-3,7,4,1]

Data Structure Interview Questions and Answers

3 comments • a year ago•

James — I did not miss the fact that the question said "middle". One of the things interviewers are looking for is your ability to

Mobile	Desktop