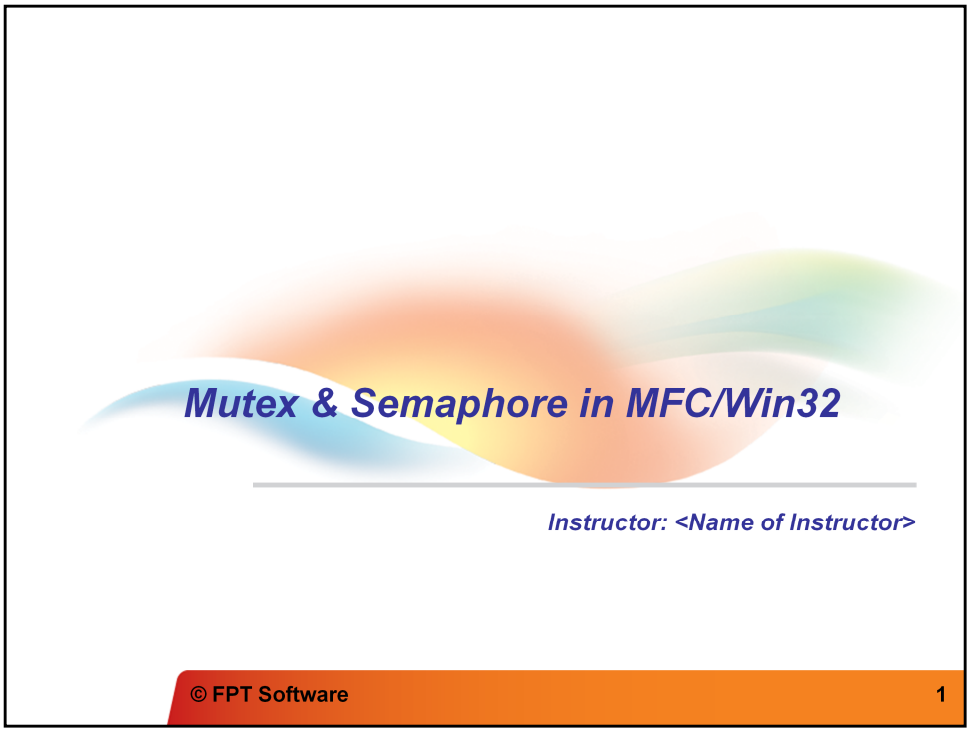


Latest updated by: HanhTT1

Agenda

- What is Thread Synchronization?
- Mutex
- Semaphore
- Mutex vs Semaphore

What is Thread Synchronization?

- Thread Synchronization is a technique that is used to synchronize resources between multiple threads and processes.
- There are two main Thread Synchronization methods:
  - Mutex
  - Semaphore

Mutex and Semaphore are used on different processes.

Mutex

- **Mutex** (Mutual Exclusion Lock) is locking mechanism used to synchronize access to a resource.
- The **Mutex** object allows accessing the resource single thread at a time.
- Only one task (thread or process) can acquire the mutex and release the mutex.
- **Mutex** has only 2 states : **locked** and **unlocked**

Mutex is the synchronization object used to synchronize the threads with more than one process.
Mutex can be released only by thread that had acquired it

Mutex - Win32

```
CreateMutex                                  //Create the mutex object
(
LPSECURITY_ATTRIBUTES lpMutexAttributes,     // Specifies a security descriptor for the new mutex
BOOL bInitialOwner,                          // State of mutex object
LPCTSTR lpName                               // Name of mutex object
)
```

```
OpenMutex                                    //Open an existing mutex object
(
DWORD dwDesiredAccess,                       // The access to the mutex object
BOOL bInheritHandle,                         // Inheritance state of current process
LPCTSTR lpName                               // The name of the mutex to be opened
)
```


If we create two or more Mutex objects on different processes, with the same name, when we call first time, the CreateMutex function creates the Mutex. The other CreateMutex function returns the handle of the previous Mutex object.

Mutex - Win32

```
WaitForSingleObject                    // Wait for a process (lock)
(
HANDLE hHandle,                        // The handle of mutex object
DWORD dwMilliseconds                   // Wait time (in millisecond)
)
```

```
ReleaseMutex                           // Release Mutex object (unlock)
(
HANDLE hMutex                          // The handle of mutex object
)
```

Mutex - Win32

- Win32 Mutex Object

```
HANDLE g_Mutex;
DWORD dwWaitResult;

// Create a mutex with TRUE status
g_Mutex = CreateMutex( NULL, TRUE, "MutexToProtectDatabase");

// Get mutex ownership
dwWaitResult = WaitForSingleObject( g_Mutex, 5000L);

// Check ownership
Bla…bla

// Release Mutex
ReleaseMutex(g_Mutex))
```


Mutex only have 2 value : true or false. Example : when status is true, the resource can be accessed, when false => the resource cannot be accessed

Mutex - MFC

- Mutex object ‘s notes (MFC)

Name	Description
CMutex	Constructs a Mutex object
CSingleLock	Controls resource access for one Mutex object
CMultiLock	Controls resource access for multiple Mutex objects
CSingleLock::Lock	Lock resource
CSingleLock::Unlock	Unlock resource

Mutex - MFC

- MFC Mutex Object

```
CMutex g_m;    //global mutex object
Int Thread1(LPVOID lParam) {
   // Create object for Single Lock
   CSingleLock lock(&g_m);
   lock.Lock();   //Lock resource
   // Unlock resource
   lock.Unlock();
   // Result
   return 0;
}
```


To control resource access for single Mutex object, use CSingleLock class. To control multiple Mutex objects, the CMultiLock is used to control the access to resources in multithreaded programming.

Semaphore

- **Semaphore** is signaling mechanism used to synchronize access to a resource.
- The **Semaphore** object allows accessing the resource for a count between zero and maximum number of thread.
- There are two **Semaphore** types:
  - **Binary Semaphore** : value of count number are 0 and 1
  - **Counting Semaphore** : value of count number >=0 and maximum thread number.

If the Thread enters the semaphore, the count is incremented
If the thread completed the work and is removed from the thread queue, the count is decremented.
When count = 0, semaphore object is non-singnaled.

Semaphore – Win32

```
CreateSemaphore                                  //Create the semaphore object
(
LPSECURITY_ATTRIBUTES lpSemaphoreAtt,   // Specifies a security descriptor for the new semaphore
LONG lInitialCount,                              // Initial count of semaphore object
LONG lMaximumCount,                              // Maximum count of semaphore object
LPCTSTR lpName                                   // Name of semaphore object
)
```

```
OpenSemaphore                                    //Open an existing semaphore object
(
DWORD dwDesiredAccess,                           // The access to the semaphore object
BOOL bInheritHandle,                             // Inheritance state of current process
LPCTSTR lpName                                   // The name of the semaphore to be opened
)
```

Semaphore – Win32

```
WaitForSingleObject                    //Wait for a process (lock)
(
HANDLE hHandle,                        // The handle of semaphore object
DWORD dwMilliseconds                   // Wait time (in millisecond)
)
```

```
ReleaseSemaphore                       //Release Semaphore object (unlock)
(
HANDLE hSemaphore,                     // The handle of semaphore object
LONG lReleaseCount,                    // The amount by which the semaphore object's
                                          current count is to be increased
LPLONG lpPreviousCount                 // The previous count for the semaphore
)
```

Semaphore – Win32

- Win32 Semaphore Object

```
HANDLE g_Semaphore;
DWORD dwWaitResult;
// Create a semaphore with initial and maximum thread number.
g_Semaphore = CreateSemaphore(  NULL,   4, 4,  NULL);
//Take ownership
dwWaitResult = WaitForSingleObject(   g_Semaphore, 0L);
// Check the ownership
Bla…bla
// Release Semaphore
ReleaseSemaphore(  g_Semaphore,  1,  NULL) ;
```


The count is never negative
OpenSemaphore function is used to open an existing handle to a semaphore object created within the process or another process.

Semaphore – MFC

- Semaphore object ‘s notes (MFC)

Name	Description
CSemaphore	Constructs a Semaphore object
CSingleLock	Controls resource access for one Semaphore object
CMultiLock	Controls resource access for multiple Semaphore objects
CSingleLock::Lock	Lock resource
CSingleLock::Unlock	Unlock resource

Semaphore – MFC

- MFC Semaphore Object

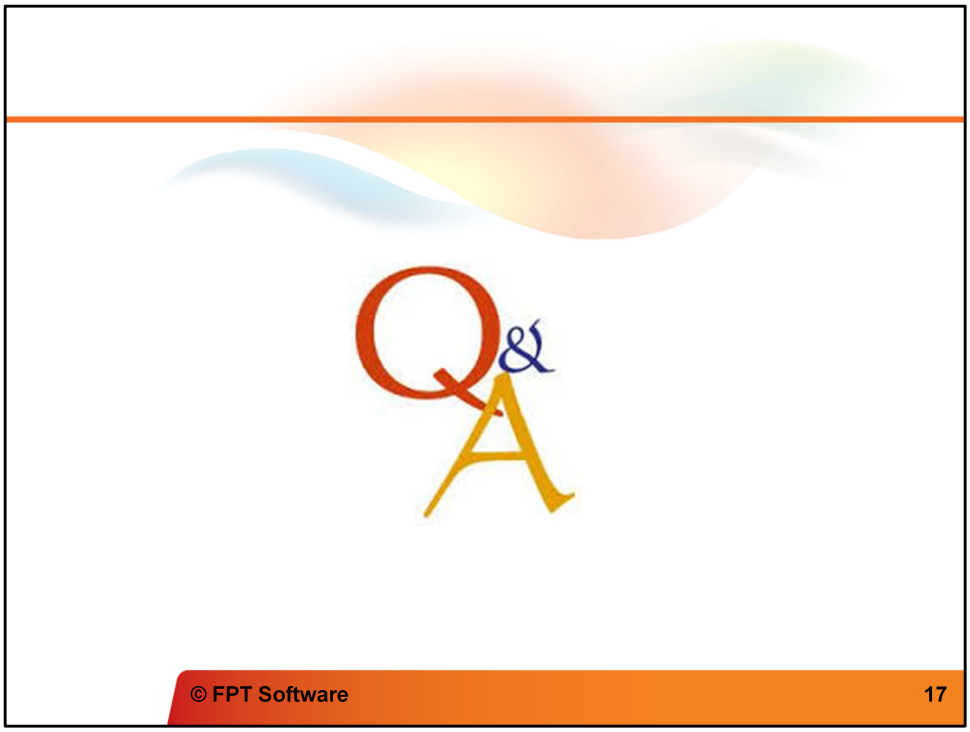
```
CSemaphore g_sm;        //global semaphore object
Int Thread1(LPVOID lParam)
{
   // Create object for Single Lock
   CSingleLock lock(&g_sm);
   lock.Lock();   //Lock resource
   // Unlock resource
   lock.Unlock();
   // Result
   return 0;
}
```

Mutex vs Semaphore

Mutex	Semaphore
- Only the process that locked the mutex can unlock it	- Any process can unlock a semaphore
- Mutex provides **priority inversion** safety (Priority inheritance)	- Semaphore does not provides **priority inversion** safety
- Mutex provides **deletion** safety	- Semaphore does not provide **deletion** safety
Mutex and Semaphore are the synchronization object can be used to synchronize the threads with more than one process	

The mutex knows its current owner, it is possible to promote the priority of the owner whenever a higher-priority task starts waiting on the mutex.
The process holding the mutex cannot be accidentally deleted

Priority inheritance : if a high priority task blocks while attempting to obtain a mutex (token) that is currently held by a lower priority task, then the priority of the task holding the token is temporarily raised to that of the blocking task. This mechanism is designed to ensure the higher priority task is kept in the blocked state for the shortest time possible, and in so doing minimise the 'priority inversion' that has already occurred.


Q&A
© FPT Software
17