

Prof. Dr. Chr. Vogt

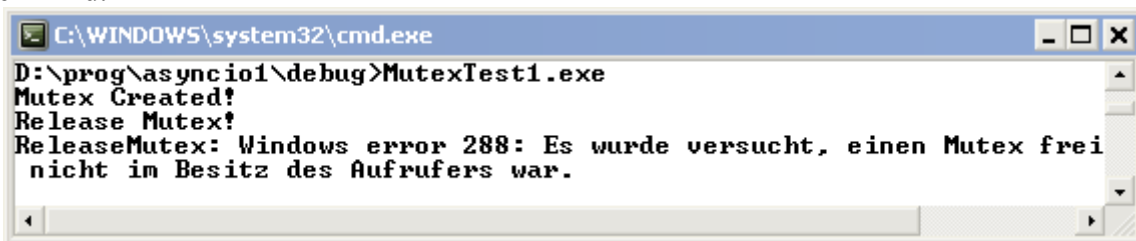
von Kulyk Nazar

(Matr.Nr.:258360030100)

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet.

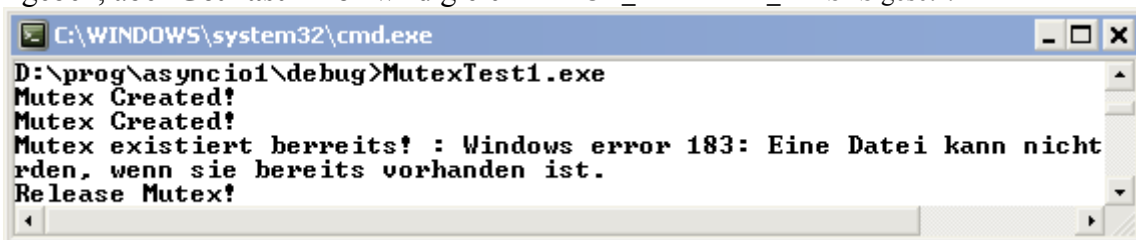
Aufgabe 1.

Wenn man einen Mutex freigeben will, den der Thread nicht hält bekommt man einen Fehler wie auf dem Bild:



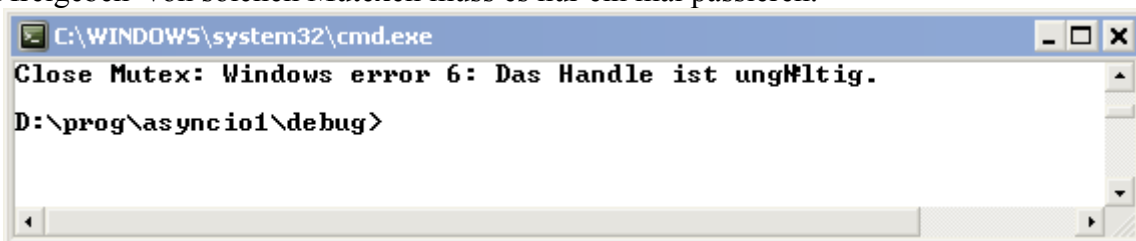
```
C:\WINDOWS\system32\cmd.exe
D:\prog\asynclol\debug>MutexTest1.exe
Mutex Created!
Release Mutex!
ReleaseMutex: Windows error 288: Es wurde versucht, einen Mutex frei
nicht im Besitz des Aufrufers war.
```

Man kann ein und den Selben Mutex in einem Thread zwei mal anfordern. Es wird keine Kritische Fehler geben, aber `GetLastError` wird gleich `ERROR_ALREADY_EXISTS` gesetzt:



```
C:\WINDOWS\system32\cmd.exe
D:\prog\asynclol\debug>MutexTest1.exe
Mutex Created!
Mutex Created!
Mutex existiert bereits! : Windows error 183: Eine Datei kann nicht
rden, wenn sie bereits vorhanden ist.
Release Mutex!
```

Beim freigeben von solchen Mutexen muss es nur ein mal passieren.



```
C:\WINDOWS\system32\cmd.exe
Close Mutex: Windows error 6: Das Handle ist ungültig.
D:\prog\asynclol\debug>
```

Bei Semaphoren ist es möglich Semaphor freizugeben in einem anderem Thread der den nie angefordert hat, falls die Rechte sind gesetzt auf dem Handle - `SEMAPHORE_MODIFY_STATE`.



```
C:\WINDOWS\system32\cmd.exe
D:\prog\asynclol\debug>SemaphorTest1.exe
Semaphor Created!
Release Semaphor!
D:\prog\asynclol\debug>
```

Aufgabe 2.

```
DWORD dwWaitResult;

dwWaitResult = WaitForSingleObject(
    hSemaphore,
    0L);

switch (dwWaitResult)
{
    case WAIT_OBJECT_0:
        // Semaphore war in signaled Status.
        break;

    case WAIT_TIMEOUT:
        // Semaphore war in nonsignaled Status.
        break;
}
```

Aufgabe 3.

Test Programm Beschreibung:

Das getestete Abschnitt weist einer Variable zu in 10000000 Zyklen.

```
for(i=0;i<=10000000;i++)
{
    j = i;
}
```

Dabei sind alle Optimierungen bei Kompilierung ausgeschaltet.
Getestet werden verschiedene Synchronisationsmechanismen.

a) Ohne jede Synchronisation läuft das Programm am schnellstem natürlich:



```
C:\WINDOWS\system32\cmd.exe
D:\prog\async io1\debug>CritSBench.exe
Its taked 00 Min 00 Sek and 31 Milliseconds
```

b) Es wird einen **InterlockedExchange(&j,i)**; statt **j = i;** verwendet.



```
C:\WINDOWS\system32\cmd.exe
D:\prog\async io1\debug>CritSBench.exe
Its taked 00 Min 00 Sek and 141 Milliseconds

D:\prog\async io1\debug>CritSBench.exe
Its taked 00 Min 00 Sek and 140 Milliseconds

D:\prog\async io1\debug>
```

c) Wir führen einen Mutex ein um den Zugriff auf die Variable **j** für zu schützen.



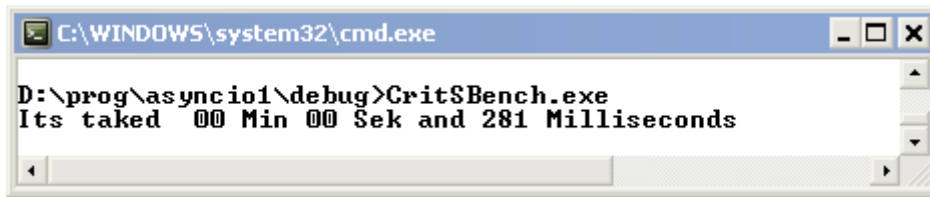
```
C:\WINDOWS\system32\cmd.exe
D:\prog\async io1\debug>CritSBench.exe
Its taked 00 Min 14 Sek and 16 Milliseconds

D:\prog\async io1\debug>
```

Das freigeben von Mutex verlangsamt den Ablauf dramatisch!

d) Jetzt verwenden wir eine CriticalSection um wieder **j** zu schützen. Es werden dabei

EnterCriticalSection and LeaveCriticalSection verwendet.



```
C:\WINDOWS\system32\cmd.exe
D:\prog\asynco1\debug>CritSBench.exe
Its taked 00 Min 00 Sek and 281 Milliseconds
```

Zusammenfassung. Wenn man einen geschütztes Zugriff auf eine Variable braucht ist es am besten eine Interlocked- Funktion zu verwenden.

Falls das nicht möglich ist kann man in den CriticalSections bestimmte Operationen machen. Das erhöht die Laufzeit ca. auf Doppelt.

Bei komplizierten Problemen kann man auch Semaphore Verwenden, dabei ist zu achten das die Laufzeiten mehr als Faktor 10 sich erhöhen.

Aufgabe 6.

1. Wenn es die Auto Reset Events nicht gäbe, könnte man CriticalSection verwenden. Dann wird jeder Reader Thread warten bis es lesen könnte auf die anderen Threads.
2. Das ist etwas schwer. Man muss alle 4 Leser Threads benachrichtigen das es einen Schreiben passieren soll und das atomar.
Man kann eventuell noch mal 4 Auto Reset Events machen um die Lese Threads zu benachrichtigen. Problem ist dann dass man nicht weiß wann das Schreiben zu ende ist. Also braucht man noch mal Events für ende des Schreibens.
Unterschied dabei ist das das ganze ist nicht Atomar und muss sehr vorsichtig gemacht werden. Es kann passieren dass man noch mal Schreiben will und der Lese Thread noch nicht im stand ist dieses Event abzufragen.
3. Es kann noch einen Manual Reset Event für jeden Lese Thread eingeführt wird. Im Schreiben Thread soll einen WaitForMultipleObject kommen, der wartet bis alle Lese Threads ihre Events auf „signaled“ setzen.