Средства синхронизации потоков на примере Windows NT

(лекция №8)

1. Критическая секция

```
/*1)*/     #include <windows.h>
/*2)*/     int g_nIndex = 0;
/*3)*/     const int MAX_TIMES = 1000;
/*4)*/     DWORD g_dwTimes[MAX_TIMES];
/*5)*/     CRITICAL_SECTION g_CriticalSection;
/*6)*/
/*7)*/     static DWORD WINAPI FirstThread(void * pv)
/*8)*/     {
/*9)*/        BOOL fDone = FALSE;
/*10)*/       while (!fDone)
/*11)*/       {
/*12)*/             ::EnterCriticalSection(&g_CriticalSection);
/*13)*/             if (g_nIndex >= MAX_TIMES)
/*14)*/                   fDone = TRUE;
/*15)*/             else
/*16)*/                   g_dwTimes[g_nIndex++] = ::GetTickCount();
/*17)*/             ::LeaveCriticalSection(&g_CriticalSection);
/*18)*/       }
/*19)*/       return 0;
/*20)*/    }
```

```
/*21)*/
/*22)*/     static DWORD WINAPI SecondThread(void * pv)
/*23)*/     {
/*24)*/         BOOL fDone = FALSE;
/*25)*/         while (!fDone)
/*26)*/         {
/*27)*/               ::EnterCriticalSection(&g_CriticalSection);
/*28)*/               if (g_nIndex >= MAX_TIMES)
/*29)*/                     fDone = TRUE;
/*30)*/               else
/*31)*/                     g_dwTimes[++g_nIndex -1] = ::GetTickCount();
/*32)*/               ::LeaveCriticalSection(&g_CriticalSection);
/*33)*/         }
/*34)*/         return 0;
/*35)*/     }
/*36)*/
/*37)*/     int main(int argc, char* argv[])
/*38)*/     {
/*39)*/         HANDLE hThreads[2];
/*40)*/         ::InitializeCriticalSection(&g_CriticalSection);
/*41)*/
/*42)*/         DWORD dwThreadID;
/*43)*/        hThreads[0] = ::CreateThread(NULL, 0, FirstThread,
/*44)*/     &dwThreadID, 0, &dwThreadID);
/*45)*/         hThreads[1] = ::CreateThread(NULL, 0, SecondThread,
/*46)*/     &dwThreadID, 0, &dwThreadID);
/*47)*/     // ждем завершения обоих потоков
/*48)*/         ::WaitForMultipleObjects(2, hThreads, TRUE, INFINITE);
/*49)*/
/*50)*/         // закрываем описатели потоков
/*51)*/         ::CloseHandle(hThreads[0]);
/*52)*/         ::CloseHandle(hThreads[1]);
/*53)*/
/*54)*/         ::DeleteCriticalSection(&g_CriticalSection);
/*55)*/         return 0;
/*56)*/     }
```

2. Мютексы

```
/*57)*/     #include <windows.h>
/*58)*/     int g_nIndex = 0;
/*59)*/     const int MAX_TIMES = 1000;
/*60)*/     DWORD g_dwTimes[MAX_TIMES];
/*61)*/     HANDLE g_hMutex = NULL;
/*62)*/
/*63)*/     static DWORD WINAPI FirstThread(void * pv)
/*64)*/     {
/*65)*/        BOOL fDone = FALSE;
/*66)*/        while (!fDone)
/*67)*/        {
/*68)*/              ::WaitForSingleObject(&g_hMutex, INFINITE);
/*69)*/              if (g_nIndex >= MAX_TIMES)
/*70)*/                    fDone = TRUE;
/*71)*/              else
/*72)*/                    g_dwTimes[g_nIndex++] = ::GetTickCount();
/*73)*/              ::ReleaseMutex(&g_hMutex);
/*74)*/        }
/*75)*/        return 0;
/*76)*/     }
/*77)*/
/*78)*/     static DWORD WINAPI SecondThread(void * pv)
/*79)*/     {
/*80)*/        BOOL fDone = FALSE;
/*81)*/        while (!fDone)
/*82)*/        {
/*83)*/              ::WaitForSingleObject(&g_hMutex, INFINITE);
/*84)*/              if (g_nIndex >= MAX_TIMES)
/*85)*/                    fDone = TRUE;
/*86)*/              else
/*87)*/                    g_dwTimes[++g_nIndex -1] = ::GetTickCount();
/*88)*/              ::ReleaseMutex(&g_hMutex);
/*89)*/        }
/*90)*/        return 0;
/*91)*/     }
/*92)*/
/*93)*/     int main(int argc, char* argv[])
/*94)*/     {
/*95)*/
```

```
/*96)*/        HANDLE hThreads[2];
/*97)*/
/*98)*/        g_hMutex = ::CreateMutex(NULL, FALSE, NULL);
/*99)*/
/*100)*/       hThreads[0] = ::CreateThread(NULL, 0, FirstThread, NULL, 0,
/*101)*/    &dwThreadID);
/*102)*/       hThreads[1] = ::CreateThread(NULL, 0, SecondThread, NULL,
/*103)*/    0, &dwThreadID);
/*104)*/
/*105)*/       ::WaitForMultipleObjects(2, hThreads, TRUE, INFINITE);
/*106)*/
/*107)*/       ::CloseHandle(hThreads[0]);
/*108)*/       ::CloseHandle(hThreads[1]);
/*109)*/
/*110)*/       ::CloseHandle(&g_hMutex);
/*111)*/       return 0;
             }
```

3. Мютекс создан после создания потоков

```
/*112)*/     …
/*113)*/        hThreads[0] = ::CreateThread(NULL, 0, FirstThread, NULL,
/*114)*/     CREATE_SUSPENDED, &dwThreadID);
/*115)*/        hThreads[1] = ::CreateThread(NULL, 0, SecondThread, NULL,
/*116)*/     CREATE_SUSPENDED, &dwThreadID);
/*117)*/
/*118)*/        g_hMutex = ::CreateMutex(NULL, FALSE, NULL);
/*119)*/
/*120)*/        ::ResumeThread(hThreads[0]);
/*121)*/        ::ResumeThread(hThreads[1]);
/*122)*/     …
```

4. VOID Sleep(DWORD ***dwMilliseconds***);

5. Семафоры

BOOL ReleaseSemaphore(HANDLE *hSemaphore*, // handle to the semaphore object
LONG *lReleaseCount*, // amount to add to current count

```
LPLONG lpPreviousCount // address of previous count
);
```

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, // SD
  LONG lInitialCount,                          // initial count
  LONG lMaximumCount,                          // maximum count
  LPCTSTR lpName                               // object name
);
```

```
HANDLE OpenSemaphore(
  DWORD dwDesiredAccess,  // access
  BOOL bInheritHandle,    // inheritance option
  LPCTSTR lpName          // object name
);
```

```
BOOL ReleaseSemaphore(
  HANDLE hSemaphore,        // handle to semaphore
  LONG lReleaseCount,       // count increment amount
  LPLONG lpPreviousCount    // previous count
);
```