Лекция 16
Синхронизация потоков с помощью объектов ядра.

- Объекты ядра
- Дескрипторы объектов ядра
- Создание и удаление объектов
- Свободное (signaled) и занятое (non-signaled) состояния
- События
- Мьютексы
- Семафоры

Объект	Описание объекта
Process	Представляет процесс.
Thread	**Представляет поток.**
File	Представляет открытый файл.
FileMapping	Представляет отображаемый в память файл.
Pipe	Используется для обмена данными между процессами.
Event	**Представляет объект синхронизации потоков, сигнализирующим о завершении операции.**
Mutex	**Представляет объект синхронизации потоков, который может использоваться несколькими процессами.**
Semaphore	**Используется для учета ресурсов. Сигнализирует потоку о доступности ресурса на данный момент.**

Примеры функций, создающих объект ядра:

HANDLE **CreateFile**(LPCTSTR *lpFileName*, // имя файла DWORD *dwDesiredAccess*, // GENERIC_READ, … DWORD *dwShareMode*, // FILE_SHARE_WRITE, … LPSECURITY_ATTRIBUTES *lpSecurityAttributes*, //NULL – дескриптор файла не наследуется DWORD *dwCreationDisposition*, // OPEN_EXISTING DWORD *dwFlagsAndAttributes*, //FILE_ATTRIBUTE_HIDDEN HANDLE *hTemplateFile* // Копирование аттрибутов шаблона);

SECURITY_ATTRIBUTES sa;
sa.nLength = sizeof(sa);
sa.lpSecuntyDescriptor = NULL;
sa.bInheritHandle =- TRUE; //делаем возвращаемый //дескриптор наследуемым

BOOL **CreateProcess**(LPCTSTR *lpApplicationName*, // имя исполняемого модуля LPTSTR *lpCommandLine*, // командная строка LPSECURITY_ATTRIBUTES *lpProcessAttributes*, //наследование дескриптора процесса LPSECURITY_ATTRIBUTES *lpThreadAttributes*, BOOL *bInheritHandles*, //наследование дескрипторов //объектов, открытых в родительском процессе DWORD *dwCreationFlags*, // флаги создания LPVOID *lpEnvironment*, // среда исполнения LPCTSTR *lpCurrentDirectory*, // текущая директория LPSTARTUPINFO *lpStartupInfo*, // вид окна LPPROCESS_INFORMATION *lpProcessInformation* // информация о процессе);

```
HANDLE CreateThread (
LPSECURITY_ATTRIBUTES  SecurityAttributes,
ULONG StackSize,
SEC_THREAD_START StartFunction,          PVOID
ThreadParameter,                         ULONG
CreationFlags,                          PULONG
ThreadId                                      );
```

```
DWORD WINAPI ThreadProc(
LPVOID lpParameter // данные );
```

```
HANDLE WINAPI CreateEvent(
LPSECURITY_ATTRIBUTES lpEventAttributes,
BOOL bManualReset,// SetEvent, ResetEvent
BOOL bInitialState,//TRUE - открыт
LPCTSTR lpName
);
```

```
BOOL SetEvent(
HANDLE hEvent );
```

```
HANDLE WINAPI CreateMutex(
LPSECURITY_ATTRIBUTES   lpMutexAttributes,
//NULL - дескриптор безопасности по умолчанию
BOOL bInitialOwner,
//FALSE (начальный владелец не определен)
LPCTSTR lpName  //NULL - создается без имени
);
```

```
HANDLE WINAPI CreateSemaphore(
LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
LONG lInitialCount, //начальное значение счетчика
LONG lMaximumCount, //максимальное значение
LPCTSTR lpName
);
```

Освобождение объекта: **BOOL**
CloseHandle(HANDLE hObj);

Возврат функции WaitForSingleObject происходит, когда объект находится в *свободном состоянии* (*сигнальном состоянии*) или когда истекает время ожидания:

```
DWORD WINAPI WaitForSingleObject(
 HANDLE hHandle, //Дескриптор объекта
 DWORD dwMilliseconds // Время ожидания
);
```

Перед возвратом функция WaitForSingleObject может менять состояние ожидаемого объекта (увеличивает счетчик семафора, переводит событие и мьютекс в занятое состояние):

```
//  > cl /MT /D "_X86_" ev2.c
 #include <windows.h>
 #include <process.h>
 #include <stdio.h>
 HANDLE hEvent1,hEvent2;
 char sh[6];
 void Thread( void* p);
 int main( void ){
 hEvent1=CreateEvent(NULL,FALSE,TRUE,NULL);
 hEvent2=CreateEvent(NULL,FALSE,FALSE,NULL);
 _beginthread( Thread, 0, NULL );
      while( 1 ){
    WaitForSingleObject(hEvent1, INFINITE);
         printf("%s\n",sh);
    SetEvent(hEvent2);
       }
return 0;
 }
```

```
void Thread( void* pParams ) {
   int counter = 0;
   while ( 1 ){
      WaitForSingleObject(hEvent2, INFINITE);
      if(counter%2){
         sh[0]='H';sh[1]='e';sh[2]='l';sh[3]='l';sh[4]='o';sh[5]='\0';
      }
      else{
         sh[0]='B';sh[1]='y';sh[2]='e';sh[3]='_';sh[4]='u';sh[5]='\0';
      }
     SetEvent(hEvent1);
     counter++;
   }
 }
```

```
//  > cl /MT /D "_X86_" mu2.c
 #include <windows.h>
 #include <process.h>
 #include <stdio.h>
 HANDLE hMutex;
 char sh[6];
 void Thread( void* pParams );

 int main( void ) {
    hMutex=CreateMutex(NULL,FALSE,NULL);
    _beginthread( Thread, 0, NULL );
    while( 1 ){
         WaitForSingleObject(hMutex, INFINITE);//захват
         printf("%s\n",sh);
         ReleaseMutex(hMutex);//освобождение
    }
    return 0;
 }
```

```
void Thread( void* pParams ){
  int counter = 0;
  while ( 1 ){
     WaitForSingleObject(hMutex, INFINITE);//захват мьютекса
      if(counter%2){
         sh[0]='H';sh[1]='e';sh[2]='l';sh[3]='l';sh[4]='o';sh[5]='\0';
      }
      else{
         sh[0]='B';sh[1]='y';sh[2]='e';sh[3]='_';sh[4]='u';sh[5]='\0';
      }
      ReleaseMutex(hMutex); //освобождение мьютекса
      counter++;
   }
 }
```

```
//  > cl /MT /D "_X86_" se2.c
 #include <windows.h>
 #include <process.h>
 #include <stdio.h>
 HANDLE hSemaphore;
 char sh[6];
 void Thread( void* pParams );
 int main( void ) {
    hSemaphore=CreateSemaphore(NULL,1,1,NULL);
    _beginthread( Thread, 0, NULL );
    while( 1 ){
         WaitForSingleObject(hSemaphore, INFINITE);
         printf("%s\n",sh);
         ReleaseSemaphore(hSemaphore,1,NULL);
    }
    return 0;
 }
```

```
void Thread( void* pParams ){
   int counter = 0;
   while ( 1 ){
    WaitForSingleObject(hSemaphore, INFINITE);
    if(counter%2){
        sh[0]='H';sh[1]='e';sh[2]='l';sh[3]='l';sh[4]='o';sh[5]='\0';
    }
    else{
        sh[0]='B';sh[1]='y';sh[2]='e';sh[3]='_';sh[4]='u';sh[5]='\0';
    }
    ReleaseSemaphore(hSemaphore,1,NULL);
         counter++;
    }
 }
```

Упражнение1: протестировать события, мьютексы и семафоры при синхронизации потоков одного процесса.

Упражнение2: синхронизируйте доступ к разделяемой разными процессами переменной (упражнение 2, лекция 9).