Лекция 10

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
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.**

События

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

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

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

Мьютексы

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

Семафоры

```
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++;
    }
 }
```

```
//#include <windows.h>
//#include <process.h>
#include <pthread.h>
#include <stdio.h>
//HANDLE hMutex;
pthread_mutex_t Mutex;//=PTHREAD_MUTEX_INITIALIZER;
char sh[6];
void* Thread( void* pParams );

int main( void ) {
  pthread_t   thread_id;
  //hMutex=CreateMutex(NULL,FALSE,NULL);
  pthread_mutex_init(&Mutex, NULL);

  //_beginthread( Thread, 0, NULL );
  pthread_create(&thread_id, NULL, &Thread, NULL);
```

```
while( 1 ){
    //WaitForSingleObject(hMutex, INFINITE);//захват
    pthread_mutex_lock(&Mutex);
    printf("%s\n",sh);
    //ReleaseMutex(hMutex);//освобождение
    pthread_mutex_unlock(&Mutex);
  }
     return 0;
  }
```

```
void* Thread( void* pParams ){
  int counter = 0;
  while ( 1 ){
  //WaitForSingleObject(hMutex, INFINITE);//захват мьютекса
     pthread_mutex_lock(&Mutex);
     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
     pthread_mutex_unlock(&Mutex);
     counter++;
    }
    return NULL;
  }
```

Взаимоблокировка

deadlock.c

```
#include <windows.h>
#include <process.h>
#include <stdio.h>

HANDLE hMutex1, hMutex2;
double sh1=0.0;
int sh2=0;
void Thread( void* pParams );

int main( void ) {
  hMutex1=CreateMutex(NULL,FALSE,NULL);
  hMutex2=CreateMutex(NULL,FALSE,NULL);

  _beginthread( Thread, 0, NULL );

  while( 1 ){
    WaitForSingleObject(hMutex1, INFINITE);//захват
        printf("%g\n",sh1);
    WaitForSingleObject(hMutex2, INFINITE);//захват
        printf("%d\n",sh2);
    ReleaseMutex(hMutex2);//освобождение
    ReleaseMutex(hMutex1);//освобождение
  }
     return 0;
  }
```

```
void Thread( void* pParams ){
  while ( 1 ){
    WaitForSingleObject(hMutex2, INFINITE);//захват мьютекса
      sh2++;
    WaitForSingleObject(hMutex1, INFINITE);//захват мьютекса
      sh1+=0.1;
    ReleaseMutex(hMutex1); //освобождение мьютекса
    ReleaseMutex(hMutex2); //освобождение мьютекса
    }
  }
```

Упражнение 1: реализуйте алгоритм Петерсона и протестируйте его.

Упражнение 2: протестируйте события, мьютексы и семафоры при синхронизации потоков одного процесса.

Упражнение 3: измените программу *deadlock.c* так, чтобы избежать взаимоблокировки.