

Лекция 9

Синхронизация потоков

- барьерная синхронизация;
- состязательная ситуация;
- критические области;
- алгоритм Петерсона;
- синхронизация с использованием объектов ядра.

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

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

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

```
void thread1();
```

```
void thread2();
```

```
int sh=0;
```

```
int flag[2]={1,1};
```

```
int main() {
```

```
    _beginthread((void*)thread0,0,NULL);
```

```
    _beginthread((void*)thread1,0,NULL);
```

```
    while(flag[0] || flag[1]);
```

```
    printf("%i\n",sh);
```

```
    return 0;
```

```
}
```

Барьерная синхронизация

```
void thread0( ){  
    int i=0;  
    for(;i<100;i++, sh++)Sleep(10);  
  
    flag[0]=0;  
}
```

```
void thread1(){  
    int i=0;  
    for(;i<100;i++, sh+=2)Sleep(10);  
  
    flag[1]=0;  
}
```

```
#include <windows.h>
#include <stdio.h>
#include <process.h>
void thread0();
void thread1();
```

**Барьерная синхронизация
средствами Windows API**

```
int sh = 0;
//int flag[2] = { 1,1 };
int main() {
    HANDLE h[2];
    h[0] = (HANDLE)_beginthread((void*)thread0, 0, NULL);
    h[1] = (HANDLE)_beginthread((void*)thread1, 0, NULL);
    //while (flag[0] || flag[1]);
    //WaitForSingleObject(h[1], INFINITE);
    WaitForMultipleObjects(2,h,TRUE,INFINITE);

    printf("%i\n", sh);
    return 0;
}
```

```
void thread0( ){  
    int i=0;  
    for(;i<100;i++, sh++)Sleep(10);  
  
    //flag[0]=0;  
}
```

```
void thread1(){  
    int i=0;  
    for(;i<100;i++, sh+=2)Sleep(10);  
  
    //flag[1]=0;  
}
```

```
int turn=0;
void thread0( ){
    int i=0;
    for(; i < 100; i++){
        while(turn);
        sh++; //критическая область
        turn=1;
        Sleep(10); //некритическая область
    }
    flag[0]=0;
}
void thread1(){
    int i=0;
    for(;i<100;i++){
        while(!turn);
        sh+=2; //критическая область
        turn=0;
        Sleep(10); //Sleep(10000); //некритическая область
    }
    flag[1]=0;
}
```

**Состязательная ситуация
(состояние гонки)**

Конфликт запись-запись

```
int turn=0, flagReady[2]={1,1};
void thread0( ){
    int i=0;
    for(; i < 100; i++){
        flagReady[0]=1;
        while(turn && flagReady[1]);
        sh++; //критическая область
        turn=1; flagReady[0]=0;
        Sleep(10); //некритическая область
    }
    flag[0]=0;
}
void thread1(){
    int i=0;
    for(;i<100;i++){
        flagReady[1]=1;
        while(!turn && flagReady[0]);
        sh+=2; //критическая область
        turn=0; flagReady[1]=0;
        Sleep(10); //Sleep(10000); //некритическая область
    }
    flag[1]=0;
}
```

**Состязательная ситуация
(состояние гонки)**

Конфликт запись-запись

Алгоритм Петерсона

```
bool readyFlags[2];  
int turn;  
  
void EnterCriticalRegion(int threadId)  
{  
    readyFlags[threadId] = true;    // Флаг текущего потока  
    turn = 1 - threadId;           // Флаг очереди исполнения  
    while (turn == 1 - threadId && readyFlags[1 - threadId]);  
}  
  
void LeaveCriticalRegion(int threadId)  
{  
    readyFlags[threadId] = false; // Сброс флага текущего потока  
}
```



```
while(TRUE)
    EnterCriticalRegion(0)
    critical_region();
    LeaveCriticalRegion(0);
    non_critical_region();
}
```

```
while(TRUE)
    EnterCriticalRegion(1)
    critical_region();
    LeaveCriticalRegion(1);
    non_critical_region();
}
```

```
int turn=0;
void thread1( ){
    int i=0;
    for(; i < 100; i++){
        EnterCriticalRegion(0);//while(turn);
        sh++; //критическая область
        LeaveCriticalRegion(0);//turn=1;
        Sleep(10); //некритическая область
    }
    flag[0]=0;
}
void thread2(){
    int i=0;
    for(;i<100;i++){
        EnterCriticalRegion(1);//while(!turn);
        sh+=2; //критическая область
        LeaveCriticalRegion(1);//turn=0;
        Sleep(10); //Sleep(10000); //некритическая область
    }
    flag[1]=0;
}
```

Реализации механизмов синхронизации Win32 API

Критические секции

[//http://www.i2r.ru/static/374/out-15679.shtml](http://www.i2r.ru/static/374/out-15679.shtml) *//остроумная иллюстрация*

```
// > cl /MT /D "_X86_" th2.c
```

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

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

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

```
CRITICAL_SECTION cs;
```

```
char sh[6];
```

```
void Thread( void* pParams ){
    int counter = 0;
    while ( 1 ){
        //EnterCriticalSection( &cs );
        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';
        }
        //LeaveCriticalSection( &cs );
        counter++;
    }
}
```

```
int main( void ){  
    // InitializeCriticalSection( &cs );  
    _beginthread( Thread, 0, NULL );  
    while( 1 ){  
        //EnterCriticalSection( &cs );  
        printf("%s\n",sh);  
        //LeaveCriticalSection( &cs );  
    }  
    return 0;  
}
```

Объявление переменной `CRITICAL_SECTION` выделяет память необходимую для критической секции.

Перед использованием критической секции её надо инициализировать.

```
void WINAPI InitializeCriticalSection(  
    LPCRITICAL_SECTION lpCriticalSection  
);
```

Чтобы завладеть критической секцией в потоке вызывается функция

```
void WINAPI EnterCriticalSection(  
    LPCRITICAL_SECTION lpCriticalSection);
```

Чтобы освободить критическую секцию в потоке вызывается функция

```
void WINAPI LeaveCriticalSection(  
    LPCRITICAL_SECTION lpCriticalSection);
```

Чтобы исключить взаимоблокировку система генерирует исключение **EXCEPTION_POSSIBLE_DEADLOCK**

Предельное время ожидания устанавливается в реестре, ключ:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\  
Session Manager\CriticalSectionTimeout
```