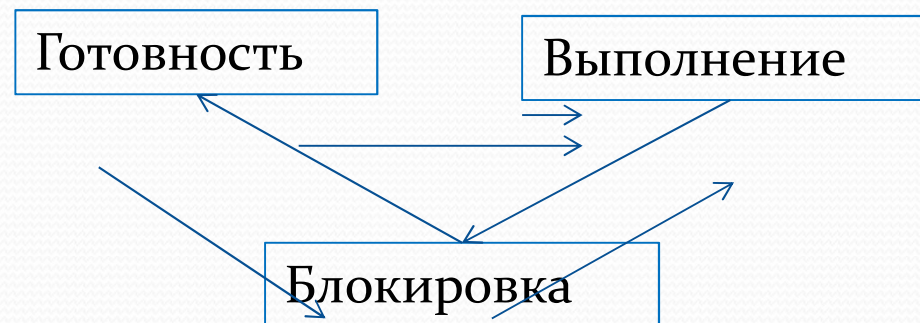


# Многопоточное программирование

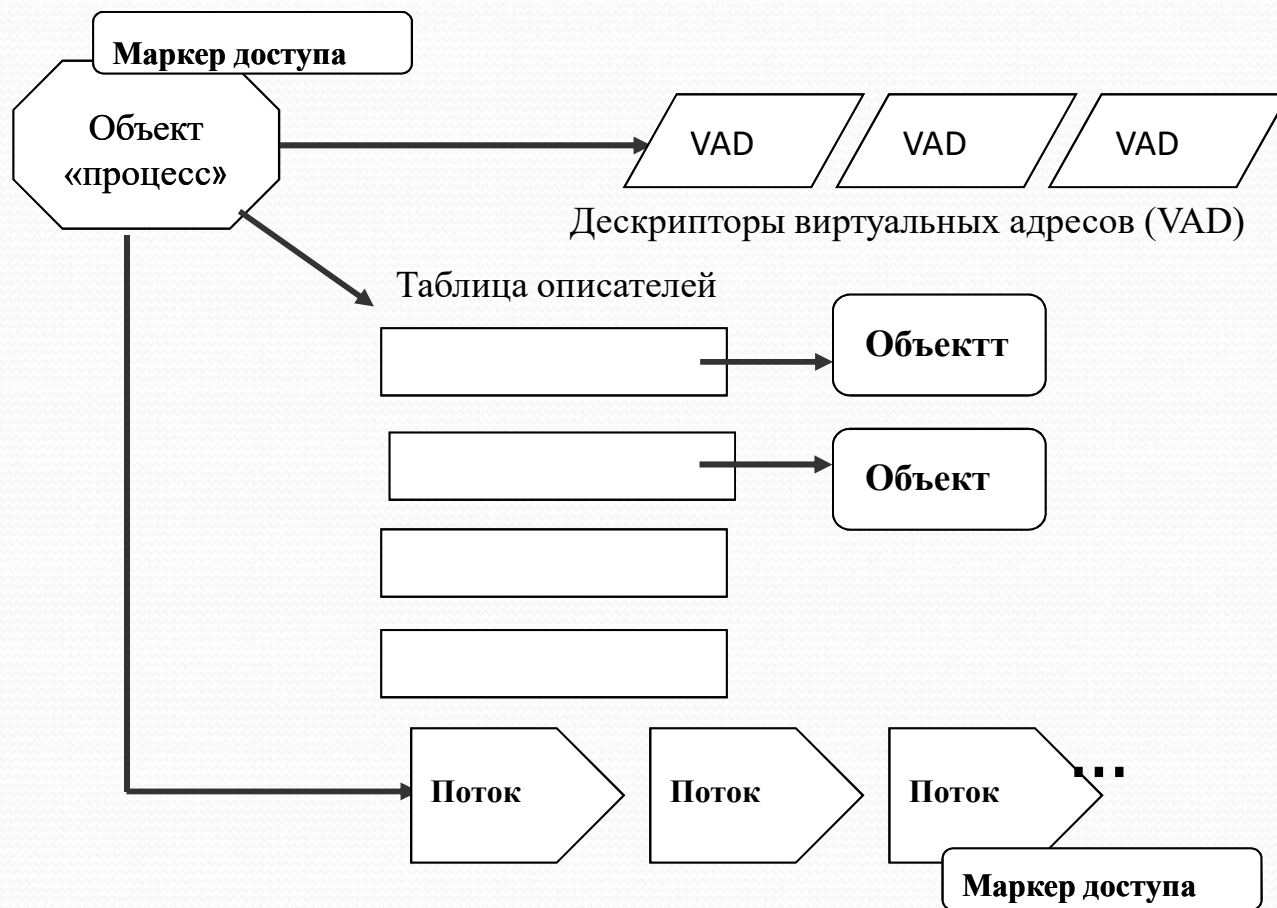
# ПРОЦЕСС

**Процесс** (или по-другому, задача) – абстракция, описывающая выполняющуюся программу, представляет собой единицу работы, заявку на потребление системных ресурсов.

Состояния процесса



# Структура процесса





# Поток (thread)

ПОТОК– это объект ядра ОС, получающий процессорное время для выполнения

## Структура потока

- содержимое набора регистров процессора, отражающих состояние процессора;
- два стека;
- закрытая область памяти (thread-local storage, TLS) ;
- уникальный идентификатор потока.

# Создание и работа с потоками

Входная функция потока

```
DWORD WINAPI ThreadProc (VOID * pParam);  
{ ...  
    return 0;  
}
```

При завершении потока система выполняет следующие действия:

- Останавливает поток
- Освобождает стек
- Счетчик пользователей для объекта ядра потока уменьшится на 1.



# Создание потока

HANDLE CreateThread(

    psa, //указатель на структуру атрибутов безопасности

    cbStack, //размер стека потока

    pfnStartAddr, //указатель на потоковую функцию

    pvParam, //параметр потоковой функции

    tdwCreate, //CREATE\_SUSPENDED или 0

    pdwThreadId //идентификатор потока

);

# Завершение потока

Поток может завершиться в следующих случаях:

- Самоуничтожается с помощью вызова **ExitThread** (не рекомендуется)
- Функция потока возвращает управление (рекомендуемый способ)
- Один из потоков данного или стороннего процесса вызывает функцию **TerminateThread**(нежелательный способ)
- Завершается процесс, содержащий данный поток (тоже нежелательно).

Функцию потока следует проектировать так, чтобы поток завершался только после того, как она возвращает управление. При этом:

- любые C++-объекты, созданные данным потоком, уничтожаются соответствующими деструкторами;
- система корректно освобождает память, которую занимал стек потока;
- система устанавливает код завершения данного потока (поддерживаемый объектом ядра "поток»);
- счетчик пользователей данного объекта ядра "поток" уменьшается на 1.



# Совместимость потоков и стандартной библиотеки

В библиотеках времени выполнения MSVCRT C предоставляются следующие функции создания  
`_beginthread(start_address, stack_size, *arglist);`  
и завершения потоков: `_endthread ();`



# Синхронизация потоков ПРИМЕР1

```
volatile bool  bReadyForProcessing = false;
volatile bool  bTerminate = false;
int  iResult = 0;
DWORD WINAPI ThreadProc(PVOID pParam)
{
    while(!bTerminate)
    {if (bReadyForProcessing)
        {iResult = iResult * 100;
         bReadyForProcessing = false;
        }
        else
        {Sleep(1); }
    }
    return 0;
}

int _main( )
{ DWORD dwID;
  HANDLE hThread = CreateThread(NULL, 0, ThreadProc,
    for(int i = 0; i < 1000; i++)
    {   iResult = 100;
        bReadyForProcessing = true;
        while(bReadyForProcessing)
            Sleep(1);
        if(10000 != iResult)
            {std::cout << "error" << std::endl;}
        std::cout << i << iResult << std::endl;
    }
    bTerminate = true;
    return 0;
}
```

cmd.exe - iResult \* 100.  
H:\WINDOWS\system32\cmd.exe

```
881 10000
882 10000
883 10000
884 10000
885 10000
886 10000
887 10000
888 10000
889 10000
890 10000
891 10000
892 10000
893 10000
894 10000
895 10000
896 10000
897 10000
898 10000
899 10000
900 10000
901 10000
902 10000
903 10000
904 10000
```

!= iResult)

:cout << "error" << std::endl;

t << i << " " << iResult << std::endl; //печатаем результат

# Как можно приостановить работу потока?

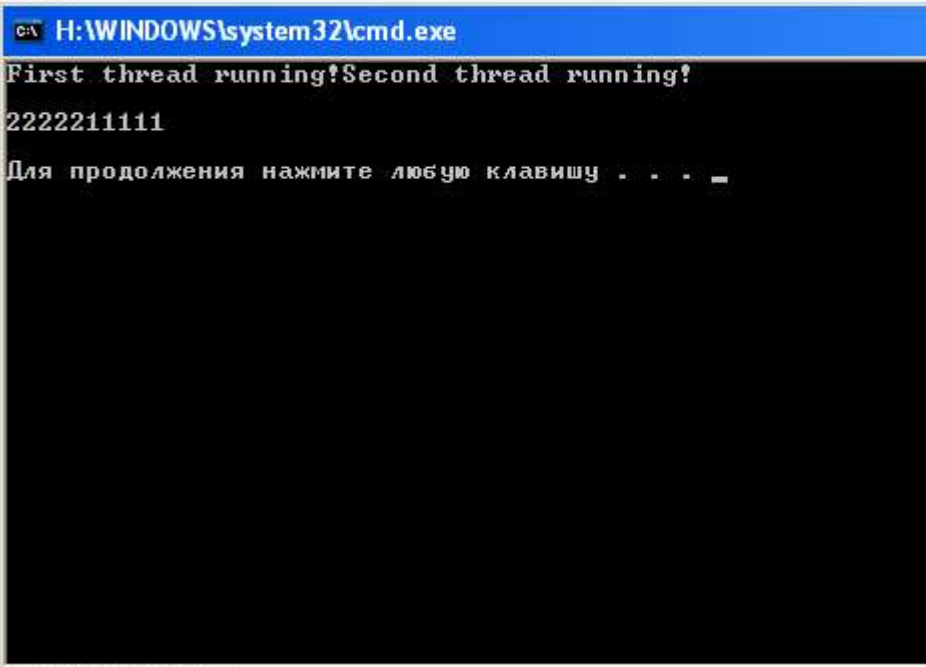
- Sleep (dwMilliseconds) ;
- WaitForSingleObject  
(hHandle, dwMilliseconds) ;
- WaitForMultipleObjects (nCount,\*lpHandles,  
bWaitAll,dwMilliseconds);

*dwMilliseconds={0..N. INFINITE}*



## ПРИМЕР 2

```
#include <process.h>
#include <iostream>
void ThreadFunc( void * arg)
{
    char **str = (char**)arg;
    std::cout <<str[0]<<std::endl;
    std::cout<<str[1]<< std::endl;
    _endthread( );
    return ;
};
int main(int argc, char* argv[])
{
    char * InitStr1[2] = {"First thread running!","1"};
    char * InitStr2[2] = {"Second thread running!","2"};
    HANDLE hThreads[2];
    hThreads[0] = (HANDLE)_beginthread( ThreadFunc,0, InitStr1);
    hThreads[1] = (HANDLE)_beginthread( ThreadFunc,0, InitStr2);
    WaitForMultipleObjects(2, hThreads, TRUE, INFINITE );
    CloseHandle( hThreads[0] );
    CloseHandle( hThreads[1] );
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "H:\WINDOWS\system32\cmd.exe". The output of the program is displayed as follows:

```
First thread running!Second thread running!
2222211111
Для продолжения нажмите любую клавишу . . .
```

# Объекты синхронизации

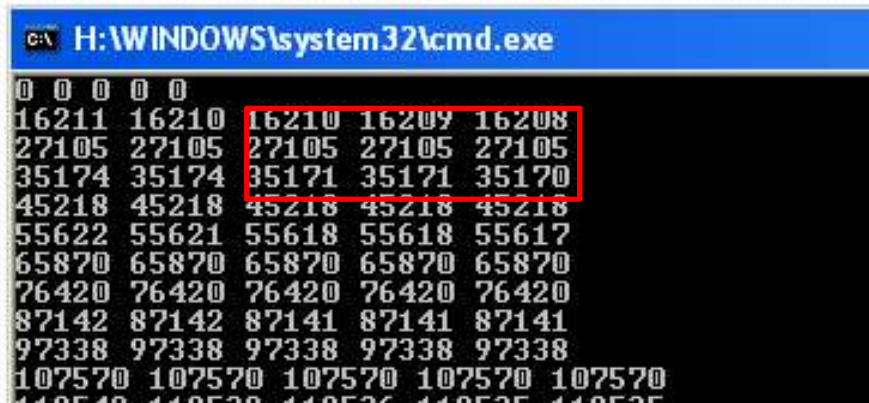
- Мьюекс (Mutex)
- Критическая секция (Criticalsection)
- Событие ( Event)
- Семафор (Semaphore)



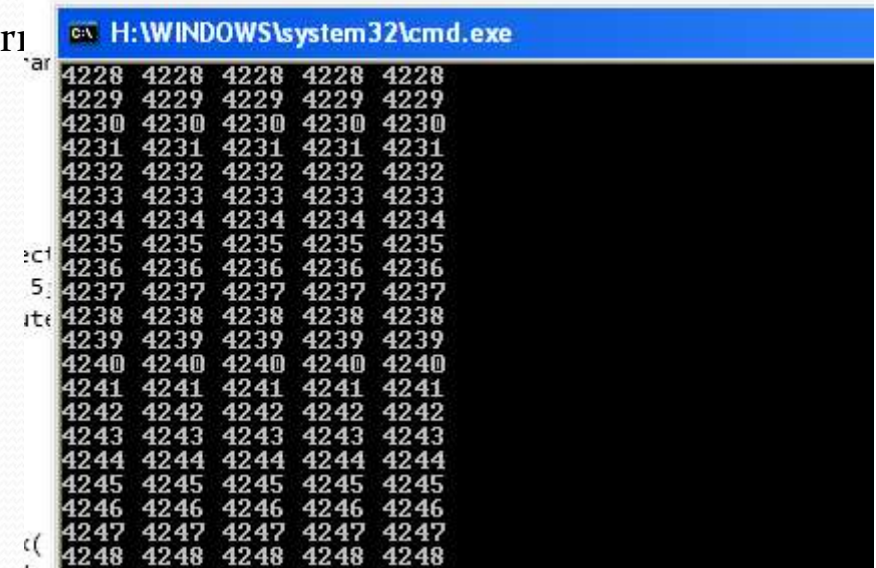
# Мьютекс. ПРИМЕР3

```
HANDLE hMutex;  
int a[ 5 ];  
  
void Thread( void* pParams )  
{  
    int i, num = 0;  
  
    while ( TRUE )  
    {  
        WaitForSingleObject( hMutex, INFINITE );  
        for ( i = 0; i < 5; i++ ) a[ i ] = num;  
        ReleaseMutex( hMutex );  
        num++;  
    }  
}
```

*bInitialOwner* = FALSE



```
int main( void )  
{  
    hMutex = CreateMutex( NULL, FALSE, NULL );  
    _beginthread( Thread, 0, NULL );  
  
    while( TRUE )  
    {  
        WaitForSingleObject( hMutex, INFINITE );  
        printf( "%d %d %d %d %d\n",  
            a[ 0 ], a[ 1 ], a[ 2 ],  
            a[ 3 ], a[ 4 ] );  
        ReleaseMutex( hMutex );  
    }  
    return 0;  
}
```





# Критические секции ПРИМЕР4

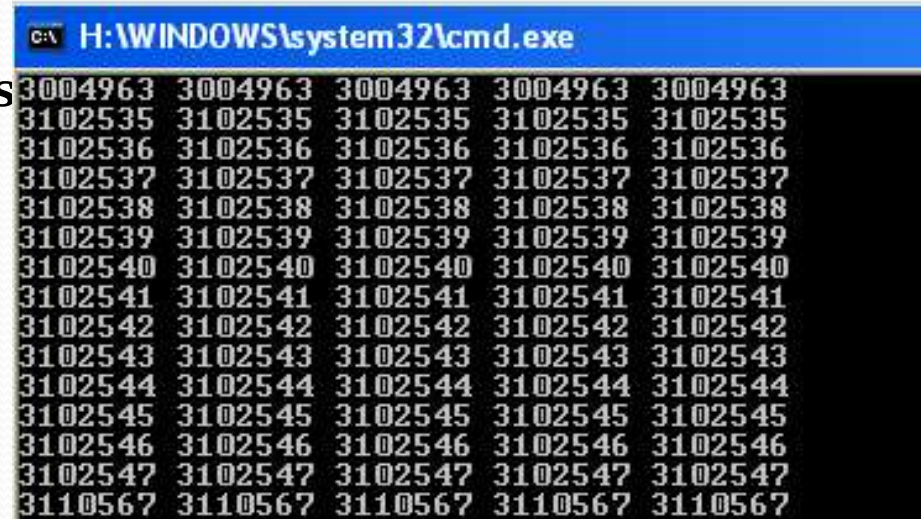
```
CRITICAL_SECTION cs;
int a[ 5 ];

void Thread( void* pParams )
{
    int i, num = 0;

    while ( TRUE )
    {
        EnterCriticalSection( &cs);
        for ( i = 0; i < 5; i++ )
            a[ i ] = num;
        LeaveCriticalSection( &cs);
        num++;
    }
}
```

```
int main( void )
{
    InitializeCriticalSection( &cs );
    _beginthread( Thread, 0, NULL );

    while( TRUE )
    {
        EnterCriticalSection( &cs );
        printf( "%d %d %d %d %d\n",
            a[ 0 ], a[ 1 ], a[ 2 ],
            a[ 3 ], a[ 4 ] );
    }
}
```



```
C:\ H:\WINDOWS\system32\cmd.exe
3004963 3004963 3004963 3004963 3004963
3102535 3102535 3102535 3102535 3102535
3102536 3102536 3102536 3102536 3102536
3102537 3102537 3102537 3102537 3102537
3102538 3102538 3102538 3102538 3102538
3102539 3102539 3102539 3102539 3102539
3102540 3102540 3102540 3102540 3102540
3102541 3102541 3102541 3102541 3102541
3102542 3102542 3102542 3102542 3102542
3102543 3102543 3102543 3102543 3102543
3102544 3102544 3102544 3102544 3102544
3102545 3102545 3102545 3102545 3102545
3102546 3102546 3102546 3102546 3102546
3102547 3102547 3102547 3102547 3102547
3110567 3110567 3110567 3110567 3110567
```



# События. ПРИМЕР5

```
HANDLE hEvent1, hEvent2;
```

```
int a[ 5 ];
```

```
void Thread( void* pParams )
```

```
{
```

```
    int i, num = 0;
```

```
    while ( TRUE )
```

```
    {
```

```
        WaitForSingleObject( hEvent2, INFINITE );
```

```
        for ( i = 0; i < 5; i++ ) a[ i ] = num;
```

```
        SetEvent( hEvent1);
```

```
        num++;
```

```
    }
```

```
}
```

```
int main( void )
```

```
{
```

```
    hEvent1 = CreateEvent( NULL,FALSE, TRUE, NULL );
```

```
    hEvent2 = CreateEvent( NULL, FALSE, FALSE, NULL );
```

```
    _beginthread( Thread, 0, NULL );
```

```
    while( TRUE )
```

```
    {
```

```
        WaitForSingleObject( hEvent1, INFINITE );
```

```
        printf( "%d %d %d %d %d\n",
```

```
            a[ 0 ], a[ 1 ], a[ 2 ],
```

```
            a[ 3 ], a[ 4 ] );
```

```
        SetEvent( hEvent2 );
```

```
    }
```

```
    return 0;
```

```
}
```

SetEvent

PulseEvent

ReversEvent

# Семафоры. ПРИМЕР6

```
class CMyClass
{ HANDLE m_hSemaphore;
  public:
  CMyClass()
      {m_hSemaphore = CreateSemaphore(NULL, 0, 1000, NULL);}
  ~CMyClass()
      { CloseHandle( m_hSemaphore);}
  void AddItem(void *NewItem)
      {
        // Добавляем элемент в очередь
        ReleaseSemaphore(m_hSemaphore,1, NULL);
      }
  void GetItem(void *Item)
      {
        WaitForSingleObject(m_hSemaphore,INFINITE);
        // Удаляем элемент из очереди
      }
}
```