

# Файловые системы

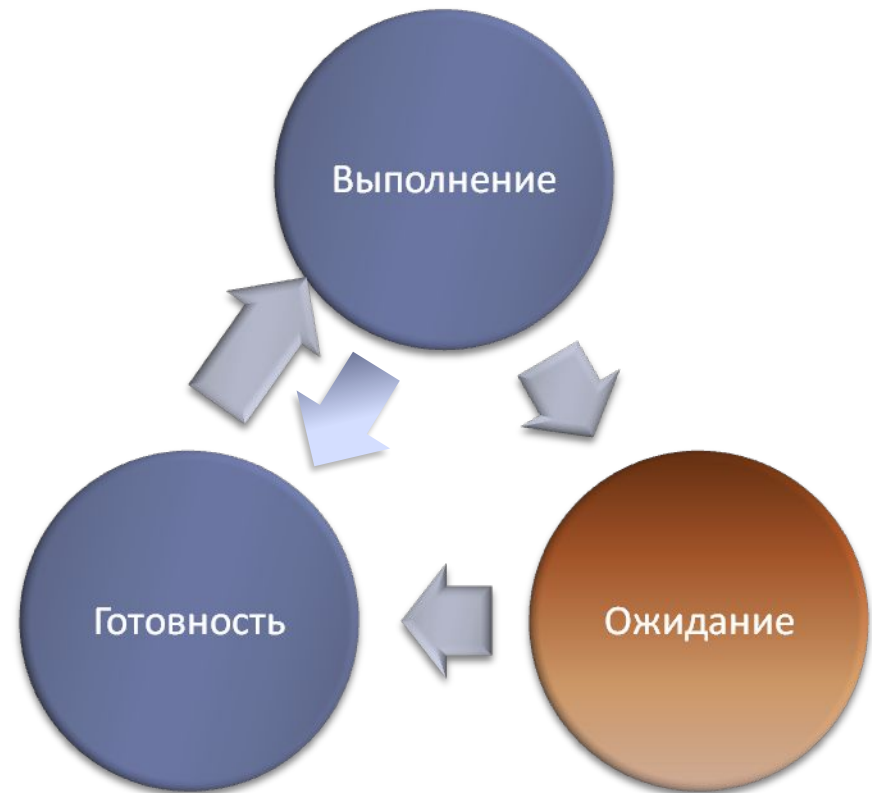
Работа с файлами в Windows API

# Работа с файлами в Windows API

Асинхронный и синхронный файловый ввод-вывод

# Синхронный и асинхронный ввод/вывод

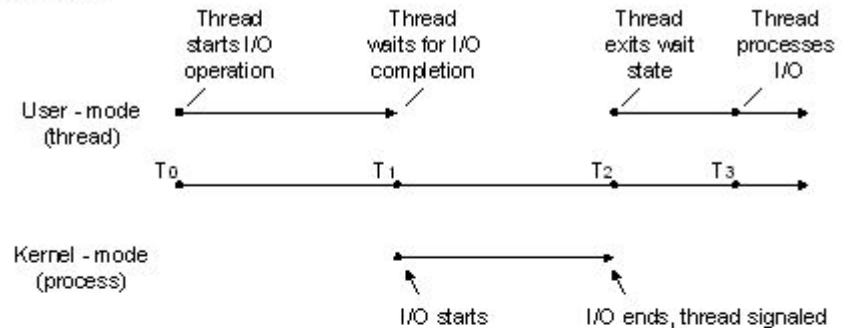
- При синхронной работе приложение, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).
- При асинхронной работе прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения, а продолжает исполняться.



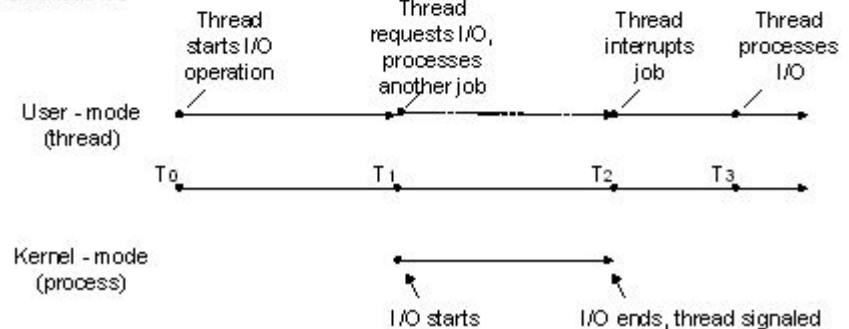
# Синхронный и асинхронный ввод/вывод

- При синхронной работе приложение, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).
- При асинхронной работе прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения, а продолжает исполняться.

Synchronous I/O



Asynchronous I/O



# Асинхронный ввод-вывод

---

- Для организации асинхронной работы с файлами необходимо при вызове функции *CreateFile ()* установить флаг `FILE_FLAG_OVERLAPPED` в параметре `dwFlagsAndAttributes`.
- После этого функции *ReadFile ()* и *WriteFile ()* будут работать асинхронно, т.е. только запускать операции ввода вывода и не ожидать их завершения.
- Структура данных `OVERLAPPED`, на которую указывает параметр *lpOverlapped*, должна оставаться допустимой для длительной операции чтения. Она не должна быть переменной, которая может выйти из области действия, пока происходит операция чтения файла.



# Перекрывающийся асинхронный ввод-вывод

---

- Когда над одним файлом (или другим объектом ввода-вывода) одновременно выполняют несколько асинхронных операций ввода-вывода, то говорят, что это перекрывающийся ввод-вывод (Overlapped I/O).
- Использование перекрывающегося ввода-вывода позволяет увеличить производительность приложений.



# Вопрос

---

- Какие проблемы с точки зрения реализации в операционной системе режима Overlapped I/O Вы видите?



# Функции файлового ввода-вывода

---

BOOL ReadFile(  
HANDLE hFile, // дескриптор файла  
LPVOID lpBuffer, // адрес буфера  
DWORD nNumberOfBytesToRead, // кол-во байт  
LPDWORD lpNumberOfBytesRead, // адрес кол-во байт  
LPOVERLAPPED lpOverlapped // адрес OVERLAPPED  
);

BOOL WriteFile(  
HANDLE hFile, // дескриптор файла  
LPCVOID lpBuffer, // адрес буфера  
DWORD nNumberOfBytesToWrite, // кол-во байт  
LPDWORD lpNumberOfBytesToWrite, // адрес кол-во байт  
LPOVERLAPPED lpOverlapped // адрес OVERLAPPED  
);





# Параметры функций файлового ввода-вывода

---

- `hFile` – дескриптор файла;
- `lpBuffer` – адрес буфера, в который будет производиться чтение/запись;
- `nNumberOfBytes...` – количество байт, которые необходимо прочитать/записать;
- `lpNumberOfBytes...` – адрес переменной, в которой будет размещено количество реально прочитанных/записанных байт;
- `lpOverlapped` – указатель на структуру `OVERLAPPED`, управляющую асинхронным вводом выводом.



# Пример синхронного копирования файла

---

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0,
    NULL);
outhandle = CreateFile ("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
    s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
    if (s && count > 0) WriteFile(outhandle, buffer, count, Socnt, NULL);
} while (s>0 && count>0);

/* Close the files. */
CloseHandle (inhandle);
CloseHandle (outhandle);
```



# Позиционирование указателя синхронного ввода-вывода

---

```
DWORD SetFilePointer(
```

```
    HANDLE hFile, // дескриптор файла
```

```
    LONG lDistanceToMove, // смещение указателя
```

```
    PLONG lpDistanceToMoveHigh, // указатель на старшую  
    часть // 64-разрядного смещения
```

```
    DWORD dwMoveMethod // точка отсчета
```

```
);
```

- ▣ **FILE\_BEGIN** – отсчет от начала файла;
- ▣ **FILE\_CURRENT** – отсчет от текущей позиции файла;
- ▣ **FILE\_END** – отсчет от конца файла.



# Установка конца файла

---

```
BOOL SetEndOfFile(  
    HANDLE hFile, // дескриптор файла  
);
```

- Функция перемещает позицию метки конца файла (EOF) для заданного файла к текущую позицию его указателя.



# Структура перекрывающегося асинхронного ввода-вывода

---

```
typedef struct _OVERLAPPED {  
    DWORD Internal; //Используется операционной системой.  
    //Хранит статус завершения операции.  
    DWORD InternalHigh; //Используется ОС. Хранит  
    //количество переданных байт.  
    DWORD Offset; //Позиция в файле, начиная с которой  
    //необходимо производить операцию  
    //чтения (записи).  
    DWORD OffsetHigh; //Количество байт для передачи.  
    HANDLE hEvent; //Описатель события, которое произойдет  
    //при завершении операции чтения //записи).  
} OVERLAPPED;
```



# Вариант 1 организации асинхронного ввода-вывода

---

- ❑ Перед запуском операции асинхронного ввода-вывода необходимо создать объект «событие» и затем передать его дескриптор в функцию *ReadFile ()* или *WriteFile ()* в качестве элемента *hEvent* структуры OVERLAPPED.
- ❑ Позиция файла, начиная с которой производится операция ввода-вывода, должна быть установлена в членах *Offset* и *OffsetHigh* структуры OVERLAPPED.
- ❑ Программа, выполнив необходимые действия одновременно с операцией передачи данных, вызывает одну из функций ожидания, например, *WaitForSingleObject ()*, передавая ей в качестве параметра дескриптор события.
- ❑ Выполнение программы при этом приостанавливается до завершения операции ввода-вывода.

# Функция *WaitForSingleObject*

---

DWORD WaitForSingleObject  
(HANDLE hObject, DWORD dwMilliseconds);

- *hObject* – идентифицирует объект ядра, относительно которого будет выполняться синхронизация, в случае асинхронного ввода-вывода файла это обычно дескриптор объекта «событие»;
- *dwMilliseconds* – указывает, сколько времени (в миллисекундах) поток готов ждать синхронизации:
  - значение «0» – функция просто проверит состояние объекта синхронизации;
  - значение INFINITE (-1) – ожидание будет «вечным», пока объект синхронизации не сработает.



# Функция *WaitForSingleObject*

---

- Функция *WaitForSingleObject ()* возвращает одно из следующих значений:
  - `WAIT_OBJECT_0` – синхронизация была выполнена;
  - `WAIT_TIMEOUT` – функция была завершена по тайм-ауту, синхронизация не выполнена;
  - `WAIT_ABANDONED` – для объекта типа «событие» не используется (используется для «семафоров» и «мьютексов»);
  - `WAIT_FAILED` – функция завершилась с ошибкой.





# Проверка завершения асинхронного ввода-вывода

---

- Проверить статус незавершенной операции асинхронного ввода-вывода можно используя макрос  
`BOOL HasOverlappedIoCompleted  
(LPOVERLAPPED lpOverlapped);`
- `#define HasOverlappedIoCompleted (lpOverlapped)  
((lpOverlapped)->Internal != STATUS_PENDING)`



# Вариант 2 организации асинхронного ввода-вывода

---

- Событие не создается. В качестве ожидаемого объекта выступает сам файл. Его дескриптор передается в функцию *WaitForSingleObject ()*.
- Этот метод прост и корректен, но не позволяет производить параллельно несколько операций ввода-вывода с одним и тем же файлом, т.е. не поддерживает перекрывающийся ввод-вывод.



# Вариант 3 организации асинхронного ввода-вывода

---

- «Тревожный» (alertable) асинхронный ввод-вывод предполагает использование функций *ReadFileEx ()* и *WriteFileEx ()*. В качестве дополнительного параметра в эти функции передается адрес функции завершения (APC –asynchronous procedure call), которая будет вызываться всякий раз при завершении операции ввода-вывода.
  - Существенно, что эти функции завершения выполняются в том же самом потоке что и функции файлового ввода/вывода. Это значит, что поток, запустивший операции чтения/записи должен приостановить себя, например, с помощью функций *Sleep ()* и *SleepEx ()*, и предоставить возможность выполнения функции завершения.
- 



# Функции *ReadFileEx* и *WriteFileEx*

---

- *BOOL ReadFileEx*(  
*HANDLE hFile, LPVOID lpBuffer,*  
*DWORD nNumberOfBytesToRead,*  
*LPOVERLAPPED lpOverlapped,*  
*LPOVERLAPPED\_COMPLETION\_ROUTINE lpCr)*
  
- *BOOL WriteFileEx*(  
*HANDLE hFile, LPVOID lpBuffer,*  
*DWORD nNumberOfBytesToWrite,*  
*LPOVERLAPPED lpOverlapped,*  
*LPOVERLAPPED\_COMPLETION\_ROUTINE lpCr)*



# Особенности тревожного асинхронного ввода-вывода

---

- Структура данных OVERLAPPED, на которую, указывает параметр *lpOverlapped* должна оставаться допустимой для длительной операции чтения. Она не должна быть переменной, которая может выйти из области действия, пока происходит операция чтения файла.
- Функции *ReadFileEx ()* и *WriteFileEx ()* игнорируют поле *hEvent* структуры OVERLAPPED.



# Функция завершения

---

```
VOID CALLBACK FileIOCompletionRoutine  
    (DWORD dwErrorCode,  
     DWORD dwNumberOfBytesTransferred,  
     LPOVERLAPPED lpOverlapped);
```

- ▣ *dwErrorCode* – состояние завершения ввода-вывода, значения параметра ограничены 0 (успешное завершение) и ERROR\_HANDLE\_EOF (при попытке выполнить чтение с выходом за пределы файла);
- ▣ *dwNumberOfBytesTransferred* – переданное число байтов;
- ▣ *lpOverlapped* – структура, которая использовалась завершившимся вызовом *ReadFileEx ()* или *WriteFileEx ()*.



# Функция *Sleep*

---

VOID Sleep (DWORD dwMilliseconds);

- Функция приостанавливает поток на *dwMilliseconds* миллисекунд.
- Особенности выполнения функции *Sleep ()*:
  - поток добровольно отказывается от остатка кванта времени;
  - система приостанавливает поток на период, **примерно** равный заданному;
  - Вы можете вообще запретить планировать поток, передав в качестве *dwMilliseconds* значение INFINITE (-1);
  - Вы можете вызвать *Sleep* и передать в качестве *dwMilliseconds* ноль. В этом случае поток будет вытеснен с процессора и помещен в очередь ожидания. Однако поток снова будет запущен, если нет других готовых потоков с тем же приоритетом.

# Функция *SleepEx*

---

DWORD SleepEx

(DWORD dwMilliseconds, BOOL bAlertable);

- ❑ Функция приостанавливает выполнения потока до наступления события ввода/вывода или на время.
- ❑ Отличия выполнения от функции *Sleep ()*:
- ❑ если параметр ***bAlertable*** = ***FALSE***, то функция ведет себя аналогично *Sleep ()*;
- ❑ если параметр ***bAlertable*** = ***TRUE***, и этот поток переходит в ожидание оповещения и может продолжить выполнение после срабатывания вызова APC или истечение времени блокировки;
- ❑ функция возвращает значение ***WAIT\_IO\_COMPLETION***, если завершение произошло в результате срабатывания вызова APC.





# Асинхронные вызовы процедур

---

- Главный поток указывает APC-функцию данной целевого потока путем помещения объекта APC в очередь APC данного потока (функция *QueueUserAPC()*). В очередь могут быть помещены несколько APC.
- Целевой поток переходит в состояние дежурного ожидания (alertable wait state), обеспечивающее возможность безопасного выполнения потоком APC.
- Целевой поток, находящийся в состоянии ожидания, выполняет все APC, находящиеся в очереди.
- **Примечание:** Порядок первых двух шагов не важен, поэтому о возникновении «гонок» можно не беспокоиться.



# Функция *QueueUserAPC*

---

- Текущий поток помещает APC в очередь целевого потока с помощью функции:

```
DWORD QueueUserAPC(
```

```
    PAPCFUNC pfnAPC, // указатель на APC-функцию
```

```
    HANDLE hThread, // дескриптор целевого потока
```

```
    ULONG_PTR dwData // передаваемое APC-функции  
    значение
```

```
);
```

- В случае успешного завершения функция возвращает – ненулевое значение, иначе – ноль.

