Практика 10: работа с файлами.

Открытие и закрытие файла.

Для открытия файла используем системный вызов **CreateFile**

```
push NULL
push FILE_ATTRIBUTE_NORMAL
push OPEN_EXISTING
push NULL
push 0
push GENERIC_READ
push newfilename
call [CreateFile]
mov [hfile],eax
```

Логичнее было бы использовать вызов **OpenFile**, но он является не рекомендуемым и оставлен для совместимости. Как объявляет официальное описание вызова – вызов обладает слишком ограниченной функциональностью и не рекомендуется к использованию.
Параметры системного вызова CreateFile следующие:

- **lpFileName** – строковый параметр, содержащий имя открываемого файла. Длина имени файла ограничена параметром MAX_PATH, который обычно ограничен 256 символов. Кому этого недостаточно, могут попробовать использовать Unicode-версию CreateFile. В ней ограничение равно 32767 символов.
- **dwDesiredAccess** – желаемые права доступа. Возможные варианты GENERIC_READ, GENERIC_WRITE, GENERIC_EXECUTE (права на исполнение, внутри приложений не используются) и GENERIC_ALL (все три права сразу, тоже не используется).
- **dwShareMode** – режим совместного доступа. Влияет - совместно с предыдущим параметром - на возможность

повторного открытия файла. Возможные варианты 0 (нет совместного доступа, наиболее часто-используемый вариант), FILE_SHARE_DELETE, FILE_SHARE_READ (совместное чтение) и FILE_SHARE_WRITE (совместная запись). Подробнее есть в документации MicroSoft.

- **lpSecurityAttributes** – ссылка на структуру разрешений безопасности. Эти параметры влияют на потомков процесса, подробнее опять же в документации MicroSoft. В нашем случае нет подчиненных процессов и потоков, так что выберем значение NULL.
- **lpCreationDisposition** – определяет действие с файлом при открытии. Возможные варианты: CREATE_ALWAYS (всегда создавать пустой файл. Если файл уже есть, то он будет замещен пустым файлом и содержимое будет потеряно), CREATE_NEW (если файла нет, то он будет создан и открыт; иначе вызов завершится ошибкой), OPEN_ALWAYS (открывает файл, если он есть; если нет, то создается и открывается пустой файл), OPEN_EXISTING (открывает только если файл существует) и TRUNCATE_EXISTING (открывает и очищает только уже существующий файл).
- **dwFlagsAndAttributes** – атрибуты и флаги. Чаще всего используется значение FILE_ATTRIBUTE_NORMAL.
- **hTemplateFile** – описатель открытого на чтение файла-шаблона. С этого файла берутся атрибуты для открываемого файла. Значение NULL (нет шаблона) используется чаще всего.

Вызов возвращает описатель открытого файла. Фактически это номер в таблице файлов в структурах ОС и для его хранения достаточно переменной типа DD.

После работы файл должен быть закрыт, иначе останется неиспользуемый объект в таблице открытых файлов, который может блокировать доступ к файлу. Для закрытия используется системный вызов **CloseHandle**

Push hFile

Call [CloseHandle]

Который обладает только одним параметром – описателем файла.
Откроем к примеру файл примера hello.asm и закроем его.

```
.data
        hfile DD ?
        filenamestr DB '.\hello.asm',0
.code
<..>
; open file
        push NULL
        push FILE_ATTRIBUTE_NORMAL
        push OPEN_EXISTING
        push NULL
        push 0
        push GENERIC_READ
        push filenamestr
        call [CreateFile]
        mov [hfile],eax
<..>
;close file
       push [hfile]
       call [CloseHandle]
<..>
```


Чтение из файла.

Рассмотрим основной способ чтения из файла – посимвольный. Для этого воспользуемся двумя системными вызовами

- **GetFileSize**
- **ReadFile**

Первый вызов позволит получить длину файла в байтах. Он имеет два параметра

1. **hFile** - описатель файла
2. **lpFileSizeHigh** - старшее слово размера файла; переменная, куда поместить старшее слово размера файла. Требуется если размер файла не помещается в eax (т.е. более 2Гб).

Второй вызов производит чтение из текущей позиции файла указанного числа байт. После чтения текущая позиция сдвигается на число прочитанных байт. Параметрами являются

1. **hFile** - дескриптор файла
2. **lpBuffer** - буфер данных, строка для получаемых символов
3. **nNumberOfBytesToRead** - число байтов для чтения
4. **lpNumberOfBytesRead** - число фактически прочитанных байтов
5. **lpOverlapped** - асинхронный буфер; указываем NULL. т.е. не используем его

В итоге, цикл побайтового чтения файла и вывода его на экран выглядит так

```
.data
  filesize DD 0
  inputnumber DD 0
  inputstr DB " ",0
.code

  push 0
  push [hfile]
  call [GetFileSize]
  mov [filesize],eax
  xor ecx,ecx
  push ecx
readstring:
  push NULL
  push inputnumber
  push 1
  push inputstr
  push [hfile]
```

```
call [ReadFile]
push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0
push inputnumber
push 1
push inputstr
push eax
call [WriteConsole]
pop ecx
inc ecx
push ecx
cmp ecx,[filesize]
jne readstring
```

Задание 1: вывести в начале каждой строки надпись «[NEWLINE]».
Решение: сравнивать байт с символом с кодом 10 и при положительном результате выводить нужную строку после него.

Запись в файл.

Для записи потребуются практически те же шаги, что и для чтения

- создание файла
- запись файла
- закрытие файла

Для создания пустого файла используется тот же системный вызов **CreateFile**, только параметр dwShareMode указываем равным FILE_SHARE_WRITE, а параметр dwCreationDisposition указываем равным CREATE_NEW.
Запись выполняется системным вызовом WriteFile с парметрами

1. **hFile** – дескриптор файла
2. **lpBuffer** – буфер записи, то есть строка с данными

3. **nNumberOfBytesToWrite** – число записываемых байт
4. **lpNumberOfBytesWritten** – число фактически записанных байт
5. **lpOverlapped** – буфер ассинхронной записи

Пример команды

push NULL
push writennumber
push 1
push witestr
push [filehandler]
call [WriteFile]

В примере выполняется запись в текущую позицию файла 1 байта из строки writestr и без использования асинхронного буфера (NULL вместо указателя на буфер).
Закрытие файла выполняется тем же системным вызовом **CloseHandle**.

Задание 2: вывести в отдельный файл результаты программы из предыдущего задания 1.