File Handling

CreateFile Function

The CreateFile function is used to create a new file or open an existing file. It returns a handle to the open file if successful, otherwise, it returns INVALID_HANDLE_VALUE.

```
335 CreateFile PROTO,
336 lpFilename: PTR BYTE,
337 dwDesiredAccess: DWORD,
338 dwShareMode: DWORD,
339 lpSecurityAttributes: DWORD,
340 dwCreationDisposition: DWORD,
341 dwFlagsAndAttributes: DWORD,
342 hTemplateFile: DWORD
```

Parameters:

- **lpFilename:** Points to the null-terminated string containing the filename.
- **dwDesiredAccess:** Specifies the type of access (read, write, read/write, device query, etc.).
- **dwShareMode:** Controls how multiple programs can access the file while it's open.
- **lpSecurityAttributes:** Points to a security structure controlling security rights.
- **dwCreationDisposition:** Specifies what to do when the file exists or doesn't exist.
- **dwFlagsAndAttributes:** Contains bit flags specifying file attributes like archive, encrypted, hidden, etc.
- **hTemplateFile:** An optional handle to a template file for attributes and extended attributes.

dwDesiredAccess Parameter Options:

The dwDesiredAccess parameter specifies the type of access to the file. You can choose from the following options or specific flag values:

• **0:** Device query access, to check device attributes or file existence.
• **GENERIC_READ:** Read access for reading from the file.
• **GENERIC_WRITE**: Write access for writing to the file.

dwCreationDisposition Parameter Options:

The dwCreationDisposition parameter specifies actions on existing and non-existing files.

Choose one of the following options:

• **CREATE_NEW:** Creates a new file, fails if it already exists.
• **CREATE_ALWAYS:** Creates a new file, overwrites if it exists.
• **OPEN_EXISTING:** Opens an existing file, fails if it doesn't exist.
• **OPEN_ALWAYS:** Opens the file if it exists, creates if it doesn't.
• **TRUNCATE_EXISTING:** Opens the file and truncates it to size zero, fails if it doesn't exist.

==========

Let's delve deeper into the CreateFile function and provide some code examples.

lpFilename: This is the path to the file you want to create or open. It can be a fully qualified filename (including the drive and path) or just the filename. For example, "C:\myfolder\myfile.txt" or "myfile.txt".

dwDesiredAccess: Specifies the type of access to the file. You can use a combination of these flags (bitwise OR) to specify the desired access:

• GENERIC_READ: Read access.
• GENERIC_WRITE: Write access.

0: Device query access (useful for checking device attributes or file existence). dwShareMode: This parameter controls how other processes can access the file while it is open. It can take one or a combination of these flags (bitwise OR):

- **FILE_SHARE_READ:** Other processes can read the file.
- **FILE_SHARE_WRITE:** Other processes can write to the file.
- **0:** No sharing allowed.

lpSecurityAttributes: This parameter allows you to specify a security structure, but you can usually set it to NULL if you don't need to set specific security attributes.

dwCreationDisposition: Specifies what to do when the file exists or doesn't exist. You can choose from these options:

- **CREATE_NEW:** Creates a new file. If the file already exists, the function fails.
- **CREATE_ALWAYS:** Creates a new file. If it exists, it overwrites it.
- **OPEN_EXISTING:** Opens an existing file. If it doesn't exist, the function fails.
- **OPEN_ALWAYS:** Opens the file if it exists or creates it if it doesn't.
- **TRUNCATE_EXISTING:** Opens the file and truncates it to size zero. Fails if the file doesn't exist.

dwFlagsAndAttributes: This parameter allows you to set various file attributes. Common attributes include FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_HIDDEN, etc.

hTemplateFile: You can typically set this to NULL. It's an optional handle to a template file that can supply file attributes and extended attributes for the file being created.

Here's an example of how you might use the CreateFile function in assembly language to create or open a text file and get a handle to it:

```
.data
  filePath BYTE "myfile.txt", 0
  handle HANDLE ?

.code
  ; Create or open the file for writing
  INVOKE CreateFile, ADDR filePath, GENERIC_WRITE, 0, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0

  ; Check if the file handle is valid
  cmp eax, INVALID_HANDLE_VALUE
  je fileCreationFailed

  ; Store the file handle
  mov handle, eax

  ; Now you can write to the file using the handle

  ; Close the file when done
  INVOKE CloseHandle, handle

fileCreationFailed:
  ; Handle the case where file creation/opening failed
  ; This could include error checking and cleanup
```

This code opens or creates the file "myfile.txt" for writing and checks if the operation was successful.

Combined Code:

Here's a single code example that demonstrates the creation and opening of files using CreateFile, and reading from a file using ReadFile. It uses the file "mydata.txt" for illustration:

```
.data
  filePath BYTE "mydata.txt", 0
  handle HANDLE ?
  bytesRead DWORD ?
  buffer BYTE 128 DUP(?)
.code
  ; Create or open the file for writing
  INVOKE CreateFile, ADDR filePath, GENERIC_WRITE, 0, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
  ; Check if the file handle is valid
  cmp eax, INVALID_HANDLE_VALUE
  je fileCreationFailed
  ; Store the file handle
  mov handle, eax
  ; Write some data to the file (assuming data is in the buffer)
  ; Close the file handle
  INVOKE CloseHandle, handle
  ; Reopen the file for reading
  INVOKE CreateFile, ADDR filePath, GENERIC_READ, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0
  ; Check if the file handle is valid
  cmp eax, INVALID_HANDLE_VALUE
  je fileOpenFailed
  ; Store the file handle
  mov handle, eax
  ; Read data from the file
  INVOKE ReadFile, handle, ADDR buffer, 128, ADDR bytesRead, 0
  ; Handle the data read from the file
  ; Close the file handle
  INVOKE CloseHandle, handle
fileOpenFailed:
fileCreationFailed:
  ; Handle any failures during file creation or opening
```

CreateFile Function:
The CreateFile function is used to either create a new file or open an existing file. It returns a handle to the open file if successful or INVALID_HANDLE_VALUE if it fails. Several parameters determine how the file is accessed and what happens in various scenarios.

lpFileName: This parameter points to a null-terminated string representing the file's name and location.

dwDesiredAccess: Specifies the type of access, such as read, write, or both. It uses flags like GENERIC_READ and GENERIC_WRITE.

dwShareMode: It controls the sharing of the file among multiple programs. You can specify sharing options using constants like FILE_SHARE_READ and FILE_SHARE_WRITE.

lpSecurityAttributes: This parameter can point to a security structure that controls security rights, but for most use cases, it's set to NULL.

dwCreationDisposition: Determines what happens when the file already exists or not. It uses options like CREATE_NEW, CREATE_ALWAYS,

OPEN_EXISTING, etc.

dwFlagsAndAttributes: Contains attributes that define the file, like being hidden or read-only, and can be a combination of various attribute flags.

hTemplateFile: Optional and is used for specifying a template file that provides attributes and extended attributes for the new file.

File Access and Sharing:

dwDesiredAccess defines the type of access you want. GENERIC_READ allows reading, GENERIC_WRITE permits writing, and you can combine them for both read and write access.

dwShareMode controls how the file can be shared among different programs. It includes options like FILE_SHARE_READ and FILE_SHARE_WRITE, which determine whether other processes can read or write to the file simultaneously.

Creation and Opening Scenarios:

dwCreationDisposition specifies what should happen when opening a file: CREATE_NEW: Creates a new file, failing if it already exists. CREATE_ALWAYS: Creates a new file, overwriting an existing one. OPEN_EXISTING: Opens an existing file. OPEN_ALWAYS: Opens the file if it exists or creates a new one if it doesn't. TRUNCATE_EXISTING: Opens the file and truncates it to size zero. Attributes:

dwFlagsAndAttributes allows you to set file attributes like FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_NORMAL, and FILE_ATTRIBUTE_READONLY. CloseHandle Function:

After working with a file, it's crucial to close the handle using the CloseHandle function. This releases system resources and ensures data integrity.

ReadFile Function:
ReadFile is used to read data from a file. It requires the file handle, a buffer to store the data, the number of bytes to read, and a pointer to a variable that will hold the number of bytes actually read.

Synchronous reading is achieved by setting lpOverlapped to NULL. In

practice, you would use these functions in a sequence, like opening a file, reading or writing data, and then closing the file handle to ensure proper file handling.

The code example provided earlier demonstrates a simple file creation, writing, and reading scenario, where you can see these functions in action.