# MS Windows API for Processes/Threads

In MS Windows, the system call interface is not documented. Instead the MS Windows API is documented, which helps with being able to run programs portably across muliple versions of the MS Windows operating systems.

Creating a process/thread gives you a *handle* that is used to refer to the actual object that represents a process/thread.

- CreateProcess(...). Fork-and-exec a new process.
- CloseHandle(...).
- ExitProcess(...), TerminateProcess(...), GetExitCodeProcess(...), GetCurrentProcessId(), GetCurrentProcess().
- CreateThread(...). Create a new thread and start running the start function specified in the new thread.
- ExitThread(...), GetExitCodeThread(...), TerminateThread(...), GetCurrentThreadId(), GetCurrentThread().
- WaitForSingleObject(...), WaitForMultipleObjects(...). These can be used to wait for either a process or a thread.

Get detailed information from `http://msdn.microsoft.com/library/`

# CreateProcess Call in MS Windows API

```
BOOL WINAPI CreateProcess(
  LPCTSTR lpApplicationName,
  LPTSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL bInheritHandles,
  DWORD dwCreationFlags,
  LPVOID lpEnvironment,
  LPCTSTR lpCurrentDirectory,
  LPSTARTUPINFO lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

# Processes Related Calls in MS Windows API

```
WaitForSingleObject(hProcess, INFINITE);

CloseHandle(pi.hProcess);

DWORD WINAPI GetCurrentProcessId(void);
HANDLE WINAPI GetCurrentProcess(void);

VOID WINAPI ExitProcess(
  UINT uExitCode
);
BOOL WINAPI TerminateProcess(
  HANDLE hProcess,
  UINT uExitCode
);
BOOL WINAPI GetExitCodeProcess(
  HANDLE hProcess,
  LPDWORD lpExitCode
);
```

# MS Windows API for Processes

```
typedef struct _PROCESS_INFORMATION {
  HANDLE hProcess;
  HANDLE hThread;
  DWORD dwProcessId;
  DWORD dwThreadId;
} PROCESS_INFORMATION,
 *LPPROCESS_INFORMATION;

typedef struct _SECURITY_ATTRIBUTES {
  DWORD nLength;
  LPVOID lpSecurityDescriptor;
  BOOL bInheritHandle;
} SECURITY_ATTRIBUTES,
 *LPSECURITY_ATTRIBUTES;
```

# MS Windows API for Threads

```
HANDLE WINAPI CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  SIZE_T dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
//prototype for a thread start method
DWORD WINAPI ThreadProc(
  LPVOID lpParameter
);

DWORD WINAPI GetCurrentThreadId(void);
HANDLE WINAPI GetCurrentThread(void);

VOID WINAPI ExitThread(
  DWORD dwExitCode
);
BOOL WINAPI TerminateThread(
  HANDLE hThread,
  DWORD dwExitCode
);
```

# Checking Errors in System Calls

- `DWORD GetLastErrorCode(void)`. Retrieves the calling thread's last-error code value. The last-error code is maintained on a per-thread basis. Multiple threads do not overwrite each other's last-error code. This function should be called right after a system call returns an error (usually we know that from a negative return value from the system call).
- To obtain an error string for system error codes, use the `FormatMessage` function.

```
DWORD FormatMessage(
  DWORD dwFlags,
  LPCVOID lpSource,
  DWORD dwMessageId,
  DWORD dwLanguageId,
  LPTSTR lpBuffer,
  DWORD nSize,
  va_list* Arguments
);
```

# Sample Error Code

```
void ErrSys(char *szMsg)
{
    LPVOID lpMsgBuf;

    // Try to format the error message from the last failed call
    //  (returns # of TCHARS in message -- 0 if failed)
    if (FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER | // source and processing options
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,                              // message source
        GetLastError(),                    // message identifier
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),  // language (Default)
        (LPTSTR) &lpMsgBuf,                          // message buffer
        0,    // maximum size of message buffer
              // (ignored with FORMAT_MESSAGE_ALLOCATE_BUFFER set)
        NULL  // array of message inserts
    ))
    {
      // Display the formatted string with the user supplied string at front.
      fprintf(stderr, "%s: %s\n", szMsg, (LPSTR)lpMsgBuf);
      LocalFree(lpMsgBuf); // Free the buffer.
    } else {
        fprintf(stderr, "%s: Could not get the error message!\n",  szMsg);
    }
    fflush(NULL); /* flush all output streams */
    ExitProcess(1); /* exit abnormally */
}
```

# Using MS Visual Studio

- Visual Studio is available via MSDN program from the CS department.
- Start up Visual Studio. Choose *New Project* → *Visual C++* → *Win32* → *Win32 Console Project*.
- In the Wizard window, choose *Application Settings* → *Empty Project* → *Finish*.
- Right click on the project in the right pane and then choose *Add* → *Add Existing Item...*. Note that this doesn't copy the file into the Visual Studio project folder.
- Also note that, Visual Studio uses Unicode by default. For now, we will simply turn this off. Press ALT+F7 to open the project properties, and navigate to Configuration Properties → General. Switch Character Set to "Multi-Byte Character Setting".
- **Tip**. If you want to know definition of MS Windows API typedefs, right-click on the type (e.g. LPVOID) and select "go to definition" from the drop down menu.

# MS Windows API Examples

- `lab/ms-windows/ch2/fork-and-exec.c`
- `lab/ms-windows/ch2/fork-and-wait.c`
- `lab/ms-windows/ch2/fork-hello-world.c`
- `lab/ms-windows/ch2/fork-test.c`
- `lab/ms-windows/ch2/file-copy.c`
- `lab/ms-windows/ch2/thread-hello-world.c`
- `lab/ms-windows/ch2/thread-scheduling.c`
- `lab/ms-windows/ch2/thread-test.c`
- and others in the ms-windows/ch2 examples folder....

# Microsoft PowerShell

Powershell is a shell with a command-line and scripting language available on Microsoft platforms.

- Aliases are built-in for common commands used in bash with Unix/Linux/Mac OSX systems. For example, TAB is used for command completion and aliases exist for `ls`, `cp`, `man`, `date` etc.
- Pipes are also supported but they pass objects instead of unstructured text streams.
- Includes a dynamically typed scripting language with .NET integration. Here is a simple example of a loop:

  ```
  while ($true) {.\fork-hello-world; echo ""}
  ```

- The following shows how to time a command or script in powershell:

  ```
  Measure-Command {sleep 2}
  ```

- Powershell script files are text files with a `.ps1` extension. By default, you cannot run scripts unless they are signed. To enable it, use the following command:

  ```
  Set-ExecutionPolicy RemoteSigned
  ```

  Use the command Get-Help About_Signing to learn more about signing.