Basic Thread Operations (MFC)

- `CWinThread* AfxBeginThread(AFX_THREADPROC pfnThreadProc, LPVOID pParam, int nPriority = THREAD_PRIORITY_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL)`
  - Call this function to create a new thread
- `CWinThread* AfxGetThread()`
  - Pointer to the currently executing thread
- `CWinThread`
  - `m_bAutoDelete`: Specifies whether to destroy the object at thread termination
  - `m_hThread`: Handle to the current thread
  - `m_nThreadID`: ID of the current thread
  - `SuspendThread()`
  - `ResumeThread()`

Example: Create a Thread

```c
UINT ThreadProc(LPVOID pv)
{
    int *p = (int*)pv;
    printf("%d\n", *p);
    return 0;
}
```

```c
int nCount = 0;
AfxBeginThread(ThreadProc, &nCount);
```

Thread Synchronization (1/2)

- Critical section
  - `VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `VOID DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `VOID LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
- Mutex objects
  - `HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName)`
  - `HANDLE OpenMutex(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)`
  - `BOOL ReleaseMutex(HANDLE hMutex)`
Thread Synchronization (2/2)

- Event objects
  - HANDLE CreateEvent(LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName)
  - HANDLE OpenEvent(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)
  - BOOL CloseHandle(HANDLE hObject)
  - BOOL SetEvent(HANDLE hEvent)
  - BOOL PulseEvent(HANDLE hEvent)
  - BOOL ResetEvent(HANDLE hEvent)

- Wait functions
  - DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds)
  - DWORD WaitForMultipleObjects(DWORD nCount, CONST HANDLE *lpHandles, BOOL fWaitAll, DWORD dwMilliseconds)

Example: Wait a Thread

Create:
CWinThread *pThread = AfxBeginThread(ThreadProc, NULL, THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
pThread->m_bAutoDelete = FALSE;pThread->ResumeThread();

Wait:
WaitForSingleObject(pThread->m_hThread, INFINITE);
delete pThread;

Example: Critical Section

Initialize:
CRITICAL_SECTION MemCS;
InitializeCriticalSection(&MemCS);

Critical Section:
EnterCriticalSection(&MemCS);
// … put your code here
LeaveCriticalSection(&MemCS);

Release:
DeleteCriticalSection(&MemCS);

Example: Mutex

Initialize:
HANDLE hMutex = CreateMutex(NULL, FALSE, NULL);

Critical Section:
WaitForSingleObject(hMutex, INFINITE);
// … put your code here
ReleaseMutex(hMutex);

Release:
CloseHandle(hMutex);
Example: Event

Initialize:
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

Waiting:
WaitForSingleObject(hEvent, INFINITE);

Signaling:
SetEvent(hEvent);

Release:
CloseHandle(hEvent);

Use WS2_32.DLL

- An application or DLL is required to perform a successful WSASStartup() call before it can use Windows Sockets services.
- When it has completed the use of Winsock, the application or DLL must call WSACleanup() to deregister itself and free any resources allocated on behalf of the application or DLL.
- Ex:
  int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData)
  int WSACleanup(void)

Ex:
WORD wVersionRequested;
WSADATA wsaData;
wVersionRequested = MAKEWORD(2, 2);
int err = WSAStartup(wVersionRequested, &wsaData);
if (err != 0)
return;
...
WSACleanup();
- Use the MAKEWORD macro to create an unsigned 16-bit integer by concatenating two given unsigned character values
  - Ex: version 2.2

Summary of Winsock APIs (1/2)

- SOCKET socket(int af, int type, int protocol)
- int connect(SOCKET s, const struct sockaddr FAR* name, int namelen)
- int bind(SOCKET s, const struct sockaddr FAR* name, int namelen)
- int listen(SOCKET s, int backlog)
- SOCKET accept(SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen)
- int recv(SOCKET s, char FAR* buf, int len, int flags)
- int send(SOCKET s, const char FAR * buf, int len, int flags)
- int shutdown(SOCKET s, int how)
  - To assure that all data is sent and received on a connected socket before it is closed, an application should use shutdown() to close connection before calling closesocket().
  - int closesocket(SOCKET s)

Summary of Winsock APIs (2/2)

- u_long htonl(u_long hostlong)
- u_short htons(u_short hostshort)
- u_long ntohl(u_long netlong)
- u_short ntohs(u_short netshort)
- struct hostent FAR * gethostbyname(const char FAR * name)
- unsigned long inet_addr(const char FAR * cp)
- char FAR * inet_ntoa(struct in_addr in)
- int getsockname(SOCKET s, struct sockaddr FAR* name, int FAR* namelen)
- int getpeername(SOCKET s, struct sockaddr FAR* name, int FAR* namelen)
- int getsockopt(SOCKET s, int level, int optname, char FAR* optval, int FAR* optlen)
- int setsockopt(SOCKET s, int level, int optname, const char FAR * optval, int optlen)