Unit 3: Processes and Threads

3.4. Win32 Process Creation

Win32 Process Management

- Process = Resource container
  - One or more threads
  - Virtual address space; distinct from other processes
  - One or more code segments
  - One or more data segments containing global variables
  - Environment strings
  - Process heap
  - Open handles, other heaps
- Thread
  - Stack for procedure calls, interrupts
  - Thread Local Storage (TLS) – array of pointers to allocate unique data
  - Argument on the stack (from creating thread)
  - Context structure (machine register values; maintained by kernel)

Process Creation

- No parent/child relation in Win32
- *CreateProcess()* – new process with primary thread

```
BOOL CreateProcess(
    LPCSTR lpApplicationName,
    LPSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation)
```

Parameters

- fdwCreate:
  - CREATE_SUSPENDED, DETACHED_PROCESS, CREATE_NEW_CONSOLE, CREATE_NEW_PROCESS_GROUP
- lpStartupInfo:
  - Main window appearance
  - Parent‘s info: GetStartupInfo
  - hStdIn, hStdOut, hStdErr fields for I/O redirection
- lpProcessInformation: Ptr to handle & ID of new proc/thread

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

UNIX & Win32 comparison

- Win32 has no equivalent to fork()
- CreateProcess() similar to fork()/exec()
- UNIX $PATH vs. lpCommandLine argument
  - Win32 searches in dir of curr. Proc. Image; in curr. Dir.; in Windows system dir. (GetSystemDirectory); in Windows dir. (GetWindowsDirectory); in dir. Given in PATH
- Win32 has no parent/child relations for processes
- No UNIX process groups in Win32
  - Limited form: group = processes to receive a console event

Exiting and Terminating a Process

- Shared resources must be freed before exiting
  - Mutexes, semaphores, events
  - Use structured exception handling

But:

- _finally, _except handlers are not executed on ExitProcess;
- no SEH on TerminateProcess

```
VOID ExitProcess(
    UINT uExitCode);

BOOL TerminateProcess(
    HANDLE hProcess,
    UINT uExitCode);

BOOL GetExitCodeProcess(
    HANDLE hProcess,
    LPDWORD lpExitCode);
```

Process Execution Times

- GetProcessTimes; available only on NT
- Proc. handle can refer to active or terminated process
- Filetime is 64 bit integer (union to perform sub)

```
BOOL GetProcessTimes(
    HANDLE hProcess,
    LPFILETIME lpCreationTime,
    LPFILETIME lpExitTime,
    LPFILETIME lpKernelTime,
    LPFILETIME lpUserTime
    );
```

Example: timep

```
#include "EvryThng.h"

int _tmain (int argc, LPTSTR argv []) {
   STARTUPINFO StartUp;
   PROCESS_INFORMATION ProcInfo;
   union {          /* Structure required for file time arithmetic. */
        LONGLONG li;
        FILETIME ft;
   } CreateTime, ExitTime, ElapsedTime;

   FILETIME KernelTime, UserTime;
   SYSTEMTIME ElTiSys, KeTiSys, UsTiSys, StartTimeSys,
              ExitTimeSys;
   LPTSTR targv = SkipArg (GetCommandLine ());

   GetStartupInfo (&StartUp);
   GetSystemTime (&StartTimeSys);
```

timep – contd.

```
/* Execute the command line and wait for the process to complete. */

if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
                    NORMAL_PRIORITY_CLASS, NULL, NULL,
                    &StartUp, &ProcInfo))
   ReportError (_T ("\nError starting process."),2,TRUE);
WaitForSingleObject (ProcInfo.hProcess, INFINITE);

GetSystemTime (&ExitTimeSys);
GetProcessTimes (ProcInfo.hProcess, &CreateTime.ft,
                 &ExitTime.ft, &KernelTime, &UserTime);
ElapsedTime.li = ExitTime.li - CreateTime.li;
/* file time arithmetic to compute and print elapsed time */
CloseHandle(ProcInfo.hThread);
CloseHandle(ProcInfo.hProcess); return 0;
}
```