Basic Setup:

Objectives:
The objectives of this lab are as follows:
- Discover how to insert debugging statements in the kernel source
- Explore the process and thread structures
- Utilizing the Platform Builder remote viewing tool to monitor threads and processes

Estimated Lab Time: 75 minutes

Introduction

This is an excerpt from [http://www.microsoft.com/technet/prodtechnol/wce/plan/chapt1.mspx](http://www.microsoft.com/technet/prodtechnol/wce/plan/chapt1.mspx).

“As a multitasking operating system, Windows CE can support up to 32 simultaneous processes, each process being a single instance of an application. In addition, multithreading support allows each process to create multiple threads of execution. A thread is a part of a process that runs concurrently with other parts. Threads operate independently, but each one belongs to a particular process and shares the same memory space. The total number of threads is limited only by available physical memory.”

In this exercise, we will briefly look at the process structure and thread structure. For every process created in the system, a process structure is created to represent that particular process in the kernel. The process structure contains all the necessary information to facilitate management of that process. The more information about the process structure is provided below.

Process Structure
The following is the process structure defined in kernel.h
(C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC):

```
struct Process {
    BYTE procnum; /* 00: ID of this process [ie: it's slot number] */
    BYTE dbgActive; /* 01: ID of process currently DebugActiveProcess'ing
this process */
    BYTE bChainDebug; /* 02: Did the creator want to debug child processes? */
    BYTE bTrustLevel; /* 03: level of trust of this exe */
    #define OFFSET_TRUSTLVL 3 // offset of the bTrustLevel member in Process structure
    LPPROXY pProxList; /* 04: list of proxies to threads blocked on this process */
    HANDLE hProc; /* 08: handle for this process, needed only for
SC_GetProcFromPtr */
    DWORD dwVMBase; /* 0C: base of process's memory section, or 0 if not in
use */
```

As stated in the excerpt, Windows CE only allows the maximum of 32 processes. This is very small compared to other operating systems that allows hundreds of process. For illustration, please log in to your CSE account and find out the number of processes currently existing on the system. Write your answer in the space below:

What was the command you used? ________________________

Number of processes on CSE: _________________________

Also find the number of processes existing on your PC right now.

What was the program that you used? ______________________

Number of processes on your PC: ________________________

Since each process can have multiple threads, you can have more than just 32 paths of execution. For each thread created, a structure is used to contain information related to that particular thread. More information about the thread structure is given below.

**Thread Structure**
Scheduler operates on threads based on their priorities. The following is the thread structure defined in kernel.h (C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC):

```c
struct Thread {
    WORD wInfo; /* 00: various info about thread, see above */
    BYTE bSuspendCnt; /* 02: thread suspend count */
    BYTE bWaitState; /* 03: state of waiting loop */
    LLPROXY pProxList; /* 04: list of proxies to threads blocked on this thread */
    PTHREAD pNextInProc; /* 08: next thread in this process */

    #if HARDWARE_PT_PER_PROC
    ulong pPTBL[HARDWARE_PT_PER_PROC]; /* hardware page tables */
    #endif

    PTHREAD pTh; /* 10: first thread in this process */
    ACCESSKEY aky; /* 14: default address space key for process's threads */
    HANDLE hDbgrThrd; /* 1C: handle of thread debugging this process, if any */
    LPWSTR lpszProcName; /* 20: name of process */
    DWORD tlsLowUsed; /* 24: TLS in use bitmask (first 32 slots) */
    DWORD tlsHighUsed; /* 28: TLS in use bitmask (second 32 slots) */
    PEXCEPTION_ROUTINE pfnEH; /* 2C: process exception handler */
    LPVOID BasePtr; /* 18: Base pointer of exe load */
    PMODULE pmodResource; /* 38: module that contains the resources */
    LPName pStdNames[3]; /* 3C: Pointer to names for stdio */
    LPCWSTR pcmdline; /* 48: Pointer to command line */
    DWORD dwDyingThreads; /* 4C: number of pending dying threads */
    openexe_t oe; /* 50: Pointer to executable file handle */
    o32_lite *o32_ptr; /* ???: o32 array pointer for exe */
    BYTE bPrio; /* ???: highest priority of all threads of the process */
    WORD wPad; /* padding */
    TPGPOOL Q pgqueue; /* ???: list of the page owned by the process */
```
PROCESS pProc; /* 0C: pointer to current process */
PROCESS pOwnerProc; /* 10: pointer to owner process */
ACCESSKEY aky; /* 14: keys used by thread to access memory & handles */
PCALLSTACK pcstkTop; /* 18: current api call info */
DWORD dwOrigBase; /* 1C: Original stack base */
DWORD dwOrigStkSize; /* 20: Size of the original thread stack */
LPDWORD tlsPtr; /* 24: tls pointer */
DWORD dwWakeUpTime; /* 28: sleep count, also pending sleepcnt on waitmul */
LPDWORD tlsSecure; /* 2C: TLS for secure stack */
LPDWORD tlsNonSecure; /* 30: TLS for non-secure stack */
LP PROXY lpProxy; /* 34: first proxy this thread is blocked on */
DWORD dwLastError; /* 38: last error */
HANDLE hTh; /* 3C: Handle to this thread, needed by NextThread */
BYTE bBPrio; /* 40: base priority */
BYTE bCPrio; /* 41: curr priority */
WORD wCount; /* 42: nonce for blocking lists */
P THREAD pPrevInProc; /* 44: previous thread in this process */
LP THRDBG pThrdDbg; /* 48: pointer to thread debug structure, if any */
LPBYTE pSwapStack; /* 4C: */
FILETIME ftCreate; /* 50: time thread is created */
CLEANEVENT *lpce; /* 58: cleanevent for unqueueing blocking lists */
DWORD dwStartAddr; /* 5C: thread PC at creation, used to get thread name */
CPUCONTEXT ctx; /* 60: thread's cpu context information */
P THREAD pNextSleepRun; /* ???: next sleeping thread, if sleeping, else next on rung if runnable */
P THREAD pPrevSleepRun; /* ???: back pointer if sleeping or runnable */
P THREAD pUpRun; /* ???: up run pointer (circular) */
P THREAD pDownRun; /* ???: down run pointer (circular) */
P THREAD pUpSleep; /* ???: up sleep pointer (null terminated) */
P THREAD pDownSleep; /* ???: down sleep pointer (null terminated) */
LPCRIT pOwnedList; /* ???: list of crits and mutexes for priority inversion */
LPCRIT pOwnedHash[PRIORITY_LEVELS_HASHSIZE];
DWORD dwQuantum; /* ???: thread quantum */
DWORD dwQuantLeft; /* ???: quantum left */
LP PROXY lpCritProxy; /* ???: proxy from last critical section block, in case stolen back */
LP PROXY lpPendProxy; /* ???: pending proxies for queuing */
DWORD dwPendReturn; /* ???: return value from pended wait */
DWORD dwPendTime; /* ???: timeout value of wait operation */
P THREAD pCrabPth;
WORD wCrabCount;
WORD wCrabDir;
DWORD dwPendWakeup; /* ???: pending timeout */
WORD wCount2; /* ???: nonce for SleepList */
BYTE bPendSusc; /* ???: pending suspend count */
BYTE bDbgCnt; /* ???: recurse level in debug message */
HANDLE hLastCrit; /* ???: Last crit taken, cleared by nextthread */
//DWORD dwCrabTime;
CALLSTACK IntrStk;
DWORD dwKernTime; /* ???: elapsed kernel time */
DWORD dwUserTime; /* ???: elapsed user time */
}; /* Thread */

Activity 1: Using remote viewing tools

1. Create a new platform with the following specification:
   • Platform name: lastname_kernel (e.g. shen_kernel)
   • Use c:\csce351_lab for the path of your project.
   • In step 3 of the “New Platform Wizard” choose “Internet Appliance”.
   • In step 4 choose only Internet Explorer.
   • In step 5 choose the default setting.
3. Build the platform.
4. Download the image to the emulator (make sure you set system memory in download setting to 64 MB and screen size to 320x240 with 8 bit).
5. Once the emulator is initialized, go to platform builder -> target and choose CE Debug Zone.
6. Once the Debug Zone is initialized, click refresh to get the latest data from module list (see figure below).

7. Click OK to exit.
8. Go to target->CE Modules and Symbols Viewer and look for nk.exe. This is where you can see the status of all your modules.

How many modules have been loaded? __________________


How many processes do you see? ______________


From the Process drop down menu choose device.exe.

How many threads do you see? _____________________________


12. From Platform Builder Tools menu, choose Remote Kernel Tracker. You should see a small window similar to the figure below.
13. Click **Cancel** to close the window.
14. From the **connection menu** (see figure below), choose **Configure Windows CE Platform Manager**.

![Select a Windows CE Device](image1.png)

15. Choose **properties** button and you should see a small configuration window (see figure below). Make sure your configuration is the same as the figure.

![Windows CE Remote Kernel Tracker](image2.png)

16. Click **test** to verify the connectivity. If successful, close the Testing Device window by clicking “OK” then Device Properties window (“OK”) then Platform Manager Configuration window (“OK”).
17. From tools menu, click **Remote Process Viewer**. It should connect successfully.
18. From target menu, select **CE Processes**. What do you think are some of the differences between monitoring process, threads, modules using tools from Target menu and remote tools from Tools menu?

---

**Activity 2: Source Code Revision and fast rebuild**

Please use the same workspace from Activity 1. Make sure that you disconnect the simulator and shut down all remote monitoring tasks you experiment with in Activity 1.

**Step 1.** Download the script file from the following link:
- cse.unl.edu/~lshen/build_kernel.bat

**Step 2.** Save the script file to
- C:\Program Files\Windows CE Platform Builder\4.20\cepb\bin

**Step 3.** Using My Computer go to C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\KERNEL and copy schedule.c to C:\csce351_lab directory. **IMPORTANT: Do not skip.**

**Step 4.** In the platform builder, click file | open and choose file c:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\KERNEL\schedule.c

**Step 5.** Insert one line in function SC_NKTerminateThread(DWORD dwExitCode) to print out debug message:
DEBUGMSG(L"* entered SC_NKTerminateThread ***");

**Step 6.** Once launched, you need to create a new platform workspace. To do so, you select:
- Build | Open Build Release Directory.
- go to C:\Program Files\Windows CE Platform Builder\4.20\cepb\bin.
- Type build_kernel at the command line prompt. Notice this is very fast. We are only building the core image here.

**Step 7.** Wait until the build process is complete then download the image to emulator.

**Step 8.** In emulator, open “Recycle Bin” and close it to see your print out message in the debug screen (see below)

**Step 9.** Disconnect from the emulator.

**Step 10.** Repeat the process but this time to print out the process’s name of the thread that has just been terminated in SC_NKTerminateThread.
- pCurThread is the pointer pointed to the thread to be terminated
- pProc is the pointer in the thread structure. It is a pointer to the process of the thread
- lpszProcName is the pointer to the process name in the process structure

Step 11. Reconnect the emulator and launch and terminate Recycle Bin again to check your correctness.

Step 12. You can save the modified schedule.c for future reference. Once you are done, show your work with answers to all the questions to the TA. Be sure to replace the modified schedule.c with the original version. Do not delete the original version from C:sce351_lab\schedule.c.

If you cannot finish during the lab period, finish this lab on your own time and show the result to the TA at the beginning of next week.

End of Lab 2