Name:

SID: _____

CSCE 351: Operating System Kernels

Lab 5 – "System Call" through OEMIoControl

Basic Setup:

• Windows 2000/XP workstation with Windows CE .Net 4.2 installed.

Prerequisite:

• Know how to create a new platform (covered in Lab 1) and know how to revise the kernel source code and rebuild the platform (covered in Lab 2)

Objectives:

The objectives of this lab are as follows:

- Familiarize students with calling system functions through OEMIoControl.
- Familiarize students with thread structure and kernel source code

Estimated Lab Time: 60 minutes

Introduction

The objective of this exercise is to familiarize students with the way of calling customized system functions throughOEMIoControl. System call is the interface between OS and the user level programs. Although Microsoft provides most of the kernel source code, they don't provide enough source code to creating a new system call. In this lab, we will study how to use *OEMIoControl* to create a "system call". We revise the source code of *OEMIoControl* implementation, where the system level functions are called. This works not exactly the same way as a system call. But, we are able to transfer user-level program to the system level. The method used in this lab may help you to debug your code in your final project.

Activity 1 Check if the library file are corrupted

Because we revised the source code in previous labs, the library files in some of the machines may be corrupted. Before we run this lab, we want to make sure the library files are fully functional. For the machines that have corrupted library files, we need to restore the library files. For each machine, we need to keep a backup for the library files.

- 1. Create a new platform with the following specification:
 - Platform name: *test_lastname_lab*
 - 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.
- 2. Change from *Emulator:X86 Win32 (WCE Emulator)* Release to *Emulator:X86 Win32 (WCE Emulator)* Debug.
- 3. Build the platform
- 4. If the build succeeds, make a backup directory of the directory C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\DEBUG in C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\ and name the backup directory as DEBUG.backup
- 5. If the build fails, do the following step to restore the library files
 - a. Remove the directory C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\DEBUG
 - b. Download the file rcf.unl.edu/~lshen/DEBUG.zip to C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\
 - c. Extract the DEBUG.zip by clicking right button and select "extract here..."
 - d. make a backup directory of the directory
 C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\DEBUG in
 C:\WINCE500\PUBLIC\COMMON\OAK\LIB\X86\ and name the backup directory as
 DEBUG.backup

Activity 2

1. Make a backup file for C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\kernel.h ; and then add one field (FILETIME myCreate;) in the thread structure in

	DWORD	dwQuantLeft;/* ??: quantum left */
	LPPROXY	<pre>lpCritProxy;/* ??: proxy from last critical section block, in case stolen back *;</pre>
	LPPROXY	lpPendProxy;/* ??: pending proxies for queueing */
	DWORD	dwPendketurn;/* 77: return value from pended wait */
	DWORD	dwPendTime; /* ??: timeout value of wait operation */
	PTHREAD	pCrabPth;
	WORD	wCrabCount;
	WORD	wCrabDir;
	DWORD	dwPendWakeup;/* ??: pending timeout */
	WORD	<pre>wCount2; /* ??: nonce for SleepList */</pre>
	BYTE	bPendSusp; /* ??: pending suspend count */
	BYTE	bDbgCnt; /* ??: recurse level in debug message */
白	HANDLE	hLastCrit; /* ??: Last crit taken, cleared by nextthread */
-	//DWORD	dwCrabTime;
	CALLSTACK	IntrStk;
	DWORD	dwKernTime; /* ??: elapsed kernel time */ γ
	DWORD	<u>dwUserTime</u> ; /* ??: elapsed user time */ [⊥]
	FILETIME	myCreate;
}	; 7* mmeau	*/
L.		
E #	define THREAD	CONTEXT OFFSET 0x60
E	RRFALSE (THREAD	CONTEXT OFFSET == offsetof(THREAD, ctx)):

2. Make a backup file for

C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\KERNEL\schedule.c , and then insert a new function in it:

```
BOOL
My_GetThreadTimes(
  HANDLE hThread,
  LPFILETIME lpCreationTime
  )
ł
        BOOL retval;
  PTHREAD pTh;
         if (!(pTh = HandleToThreadPerm(hThread)))
         ł
                  retval=FALSE;
         }
         else
         {
                    try {
                          *lpCreationTime = pTh->myCreate;
            retval = TRUE;
_except (EXCEPTION_EXECUTE_HANDLER) {
                 retval = FALSE;
         }
         }
         return retval;
}
```

3. In the same file of step 2, insert the following code to update my thread creation time In DoCreateThread:

```
// use per-process VM address for stack
                  if (!((ulong)lpStack >> VA_SECTION)) {
                      lpStack = (LPVOID) ((ulong) lpStack + pCurProc->dwVMBase);
                  ł
                  // record thread creation time
                  GCFT(&pth->ftCreate);
                   // By Lu
                  GCFT(&pth->myCreate)
                  // perform machine dependent thread initialization
                  if (flags & 0x80000000) {
                       SET_DYING(pth);
                      SET_DEAD(pth);
In SC CreateProc:
 // initialize the area for CoProc registers if required
 if (cbNKCoProcRegSize && pSwapStack && pOEMInitCoProcRegisterSavedArea) {
     pOEMInitCoProcRegisterSavedArea (pSwapStack);
 pNewth->pSwapStack = pSwapStack;
 pNewth->pNextInProc = pNewth->pPrevInProc = 0;
 AddAccess(&pNewth->aky, pCurThread->aky);
 GCFT(&pNewth->ftCreate);
 //By Lu
 GCFT(&pNewth->myCreate);
 MDCreateThread(pNewth, 1pStack, CNP_STACK_SIZE, (LPVOID)CreateNewProc, 0, TH_KMODE, (ulong)&psi);
 pNewth->dwOrigBase = (DWORD) lpStack;
 pNewth->dwOrigStkSize = CNP_STACK_SIZE;
 pNewth->tlsSecure = pNewth->tlsNonSecure = pNewth->tlsPtr;
 ZeroTLS(pNewth);
 IncRef(hNewproc, pNewproc);
 IncRef(hNewth, pNewproc);
 DEBUGMSG(ZONE_ENTRY, (L"SC_CreateProc switching to loader on thread %8.81x\r\n", pNewth));
In ProcInit:
      #ifdef DEBUG
         pCurProc->ZonePtr = &dpCurSettings;
      #else
          pCurProc->ZonePtr = 0;
      #endif
          pCurProc->pProxList = 0;
          pCurProc->o32_ptr = 0;
          pCurProc->e32.e32_stackmax = KRN_STACK_SIZE;
          InitThreadStruct(pCurThread, hCurThread, pCurProc, hCurProc, THREAD_RT_PRIORITY_ABOVE_NORMAL);
         SETCURKEY(GETCURKEY()); // for CPUs that cache the access key outside the thread structure
pCurThread->pNextInProc = pCurThread->pPrevInProc = 0;
          *(__int64 *)&pCurThread->ftCreate = 0;
            Ry Lu
          *(
             _int64 *)&pCurThread->myCreate = 0;
```

4. Make a backup file for C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\schedule.h, and then add a declaration in it:

extern void MDCreateThread(PTHREAD pTh, LPVOID lpStack, DWORD cbStack, LPVOID lpBase, LPVOID lpStart, BOOL kMode, ulong param); LPCWSTR MDCreateMainThread1(PTHREAD pTh, LPVOID lpStack, DWORD cbStack, LPEYTE buf, ulong buflen, LPBYTE buf2, ulong buflen2); void MDCreateMainThread2(PTHREAD pTh, DWORD cbStack, LPVOID lpBase, LPVOID lpStart, BOOL kmode, ulong p1, ulong p2, ulong buflen, ulong buflen2, ulong p4); VOID MakeRun(PTHREAD pth); DWORD ThreadResume(PTHREAD pth); void KillSnecialThread(void): BOOL My_GetThreadTimes(HANDLE hThread, LPFILETIME lpCreationTime); #endit I

5. Make a backup file for C:\WINCE420\PLATFORM\EMULATOR\KERNEL\HAL\ oemioctl.c, and then insert a declaration in it:

#ifdef IMGSHAREETH extern BOOL OEMEthCurrentPacketFilter(PDWORD pdwRequestedFilter); extern BOOL OEMEthMulticastList(PUCHAR pucMulticastAddressList, DWORD dwNoOfAddresses); #endif BOOL (* pfnKITLGetInfo) (DWORD dwCode, LPVOID lpData, LPDWORD pcbData); // added by Lu
extern BOOL My_CetThreadTimes(HANDLE hThread, LPFILETIME lpCreationTime) -11 inline LONG InterlockedBitSet (IN OUT PLONG Target, IN LONG Bit { asm { mov eax, Bit mov ecx, Target LOCK bts [ecx], eax } }

6. In C:\WINCE420\PLATFORM\EMULATOR\KERNEL\HAL\ oemioctl.c, insert the following code in function OEMIoControl:

```
-11
 BOOL.
 OEMIoControl(
     DWORD dwIoControlCode,
     LPVOID 1pInBuf,
     DWORD nInBufSize,
     LPVOID lpOutBuf,
     DWORD nOutBufSize,
     LPDWORD lpBytesReturned\
     )
 {
     BOOL retval = FALSE;
     DYORD len;
     DEBUGMSG(0, (TEXT("+OEMIoControl %X\r\n"), dwIoControlCode));
     switch (dwIoControlCode) {
         // Added By Lu
         case -3366:
             DEBUGMSG(1, (L"*** OEMIoControl entered *** \n"));
             retval=My_GetThreadTimes((HANDLE)lpInBuf, (LPFILETIME)lpOutBuf);
             return retval;
         case IOCTL_PROCESSOR_INFORMATION:
         if (!lpOutBuf) {
             SetLastError (ERROR INVALID PARAMETER);
             return FALSE;
         }
```

6. Create a new platform with the following specification: Platform name: *lastname_lab5* 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.

- 7. Add a new project
 - a. Select File | New Project or File...
 - b. Select a WinCE application project
 - c. Name is as *lastname_project_lab5*
 - d. Select an empty project
- 8. In CSE, copy the file /home/classes/cse351/lab5c to your home directory and move it to *c:\csce351_lab\ lab5.c*
- 9. Add source file into the project
 - a. Select Project | Insert | Files...
 - b. Select the file *c:\csce351_lab\ lastname_lab5\lab5.c* from your local disk
- 10. Build the platform

- 11. Download the image
- 12. Setup the breakpoints in Function My_GetThreadTimes in C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\KERNEL\schedule.c
- 13. Select Target | Run Program and select *lastname_project_lab5.exe* to run
- 14. Observe the output Note that the thread creation time printed out may not be the same as local time

Why calling KernelIoControl in lastname_lab5 will make My_GetThreadTimes get called in schedule.c

15. Restore the files that were changed in the lab with their backup files

- C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\KERNEL\schedule.c
- C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\kernel.h
- C:\WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\schedule.h
- C:\WINCE420\PLATFORM\EMULATOR\KERNEL\HAL\ oemioctl.c

16. Make sure that you can build a **new** platform after restoring these files

Appendix

1. Process Structure

The following is the process structure defined in kernel.h (%winceroot%\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 */
                                 /* 02: Did the creator want to debug child processes? */
                 bChainDebug;
    BYTE
                                 /* 03: level of trust of this exe */
    BYTE
                 bTrustLevel;
#define OFFSET TRUSTLVL 3 // offset of the bTrustLevel member in Process structure
                pProxList;
                                 /* 04: list of proxies to threads blocked on this process
    LPPROXY
*/
    HANDLE
                 hProc;
                                  /* 08: handle for this process, needed only for
SC_GetProcFromPtr */
                                  /* OC: base of process's memory section, or 0 if not in
    DWORD
                dwVMBase;
use */
               pTh;
    PTHREAD
                                  /* 10: first thread in this process */
    ACCESSKEY aky;
                                 /* 14: default address space key for process's threads */
               BasePtr; /* 18: Base pointer of exe load */
hDbgrThrd; /* 1C: handle of thread debugging this process, if any */
            BasePtr;
    LPVOID
    HANDLE
    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 */
    LPDBGPARAM ZonePtr; /* 30: Debug zone pointer */
PTHREAD pMainTh; /* 34 primary thread in this process*/
    PMODULE
               pmodResource; /* 38: module that contains the resources */
               pStdNames[3]; /* 3C: Pointer to names for stdio */
    LPName
    LPCWSTR pcmdline;
                                  /* 48: Pointer to command line */
                 dwDyingThreads; /* 4C: number of pending dying threads */
    DWORD
                        /* 50: Pointer to executable file handle */
    openexe t
                 oe;
                                  /* ??: structure containing exe header */
    e32 lite
                 e32;
    o32 lite
                                 /* ??: o32 array pointer for exe */
                *o32 ptr;
                                 /* ??: extend pdata */
    LPVOID
                 pExtPdata;
                                  /* ??: highest priority of all threads of the process */
    BYTE
                bPrio;
                                  /* ??: this process cannot be debugged */
    BYTE
                fNoDebug;
                                  /* padding */
    WORD
                wPad;
               pgqueue;
                                  /* ??: list of the page owned by the process */
    PGPOOL Q
#if HARDWARE PT PER PROC
    ulong
                 pPTBL[HARDWARE PT PER PROC]; /* hardware page tables */
#endif
```

}; /* Process */

This table gives some further explanation on some important fields in the process structure.

procnum the slot number of this process as its ID. there are only 32 slots in CE	
pProxList	a list of objects that the threads of this process are waiting for
pTh	a process main own multiple of threads. pTh is the first one
lpszProcName	name of the process

2. Thread Structure

In Windows CE .NET, each process may contain many threads (up to virtual memory limitation). Scheduling operates on threads based on their priorities. The following is the thread structure defined in kernel.h (%winceroot%\PRIVATE\WINCEOS\COREOS\NK\INC):

struct Thread	{
WORD	wInfo; /* 00: various info about thread, see above */
BYTE	bSuspendCnt;/* 02: thread suspend count */
BYTE	bWaitState; /* 03: state of waiting loop */
LPPROXY	pProxList; /* 04: list of proxies to threads blocked on this thread */
PTHREAD	<pre>pNextInProc;/* 08: next thread in this process */</pre>
PPROCESS	pProc; /* 0C: pointer to current process */
PPROCESS	pOwnerProc; /* 10: pointer to owner process */
ACCESSKEY	aky; /* 14: keys used by thread to access memory & handles */
PCALLSTACK	<pre>c pcstkTop; /* 18: current api call info */</pre>
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 waitmult */
LPDWORD	tlsSecure: /* 2c: TLS for secure stack */
LPDWORD	tlsNonSecure: /* 30: TLS for non-secure stack */
LPPROXY	pProxy: /* 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	wount: /* 42: nonce for blocking lists */
מאסתם מינים איני	proving 7 42. Honey out thread in this process */
	prieving loc, /* 44. provides thread he this process */
	pSwapStack, /* 40; pointer to thread debug structure, if any "/
	powapolack; /* 40 */
LTETTME CLEANEVENU	(+) and (-) (+ S0; clime thread is cleated -/
CLEANEVENT	ductor the formation of the second se
DWORD	uwstartadur; /* SC: thread PC at creation, used to get thread hame */
CPUCONTEXT	Ctx; /^ 60: thread's cpu context information ^/
PTHREAD	phextsleepkun; /* ??: next sleeping thread, if sleeping, else next on
	JIE */
PTHREAD	prevsleepkun; /* /:: back pointer it sleeping or runnable */
PTHREAD	popkun; /* ??: up fun pointer (circulaal) */
PTHREAD	pDownRun; /* ??: down run pointer (circular) */
PTHREAD	pupsleep; /* ??: up sleep pointer (null terminated) */
PTHREAD	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 */
LPPROXY	lpCritProxy;/* ??: proxy from last critical section block, in case stolen
back */	
LPPROXY	lpPendProxy;/* ??: pending proxies for queueing */
DWORD	dwPendReturn;/* ??: return value from pended wait */
DWORD	dwPendTime; /* ??: timeout value of wait operation */
PTHREAD	pCrabPth;
WORD	wCrabCount;
WORD	wCrabDir;
DWORD	dwPendWakeup;/* ??: pending timeout */
WORD	wCount2; /* ??: nonce for SleepList */
BYTE	bPendSusp; /* ??: pending suspend count */
BYTE	<pre>bDbgCnt; /* ??: recurse level in debug message */</pre>
HANDLE	hLastCrit; /* ??: Last crit taken, cleared by nextthread */
//DWORD	dwCrabTime;
CALLSTACK	IntrStk;
DWORD	dwKernTime; /* ??: elapsed kernel time */
DWORD	dwUserTime; /* ??: elapsed user time */
<pre>}; /* Thread</pre>	*/

2.1. wInfo

wInfo is a 16 bit integer, which contains the following information of a thread. The MACRO shown below the table indicates the start position for each field in wInfo (as well as their length).

RUNSTATE	running state of this thread (blocked, running, runnable, needsrun)
DYING	if terminating
DEAD	if dead
BURIED	if buried
SLEEPING	if sleeping
TIMEMODE	time mode
STACKFAULT	?
DEBUGBLK	?
NOPRIOCALC	?
DEBUGWAIT	?
USERBLOCK	if thread is able to enter sleeping state automatically
NEEDSLEEP	if the thread should be put into sleeplist
PROFILE	?

#define RUNSTATE SHIFT 0 // 2 bits #define DYING SHIFT 2 // 1 bit #define DEAD_SHIFT #define BURIED_SHIFT 3 // 1 bit 4 // 1 bit #define SLEEPING_SHIFT 5 // 1 bit #define TIMEMODE SHIFT 6 // 1 bit #define NEEDDBG SHIFT 7 // 1 bit #define STACKFAULT SHIFT 8 // 1 bit #define DEBUGBLK SHIFT 9 // 1 bit #define NOPRIOCALC_SHIFT 10 // 1 bit #define DEBUGWAIT SHIFT 11 // 1 bit #define USERBLOCK SHIFT 12 // 1 bit #ifdef DEBUG #define DEBUG LOOPCNT SHIFT 13 // 1 bit - only in debug #endif #define NEEDSLEEP SHIFT 14 // 1 bit #define PROFILE SHIFT 15 // 1 bit, must be 15! Used by assembly code!

2.2. Pointers for different queues (lists)

CE schedules threads in the system according to their priorities. Each thread is scheduled without considering which process it belongs to. Many queues, e.g., runlist, sleeplist, and list of threads of a mutex, are maintain for scheduling purpose. The following fields are important for maintaining those queues:

pProc	pointer to current process
pNextInProc	pointer to next thread in this process
pPrevInProc	pointer to previous thread in this process
pOwnerProc	pointer to owner process
pNextSleepRun	pointer to the next sleeping or runnable thread (depending on its state)

pPrevSleepRun	poinster to the previous sleeping or runnable thread (depending on its state)
pUpRun	pointer to the next run thread (explained later)
pDownRun	pointer to previous runnable thread
pUpSleep	pointer to the previous sleeping thread
pDownSleep	poinster to the next sleeping thread

Note that CE maintains 2D sleeping and running queues. pNextSleepRun and pPrevSleepRun point to the threads with different priority, while pUpRun, pDownRun, pUpSleep, and pDownSleep point to threads with the same priority. The following figure is an simplified illustration of the runlist. Note that the thread with grey color have the same priority.



2.3. Priority and Quantum

Each thread is assigned a quantum for execution. The default value of the quantum is 100ms. Scheduler operates according to the preemptive round-robin algorithm.

bBPrio	base priority
bCPrio	current priority
dwQuantum	quantum
dwQuanLeft	left quantum

2.4. RunList and SleepList

```
typedef struct {
    PTHREAD pRunnable; /* List of runnable threads */
    PTHREAD pth; /* Currently running thread */
    PTHREAD pHashThread[PRIORITY_LEVELS_HASHSIZE];
} RunList_t;
```

pRunnable is a pointer to the first thread of the current runnable thread list. pth is the current running thread. pHashThread is a hash table, which contains 32 levels (PRIORITY_LEVELS_HASHSIZE) of thread lists. Each level of thread list in the pHashThread may have 8 kind of priorities. Therefore, we have totally 8*32=256 priorities.

We have a runnable list (RunList_t RunList), a sleep list (PTHREAD sleepList;), and many queues for each object, such as mutex, semaphore, etc. Please refer to schedule.c.



Here is some explanation on those 6 states from mailing list (microsoft.public.windowsce.platbuilder):

From: "Bor-Ming Hsieh [MS]" <bmhsieh@anit-spam-prefix.microsoft.com> Subject: Re: schedule.c (OS Kernel) Date: 2004年7月9日 12:14

I IpProxy - the list of objects a thread are blocked on. For example, if you call WaitForMultipleObjects (n, .) and blocked, it'll be a queue of n proxies.

I RUNSTATE indicate the 'steady state of a thread'. RUNSTATE_RUNNING, RUNSTATE_RUNNABLE, and RUNSTATE_BLOCKED should explain themselves. RUNSTATE_NEEDRUN is a transition state, introduced for Real-Timeness, that a thread is about to make RUNNABLE. The reason being that it could take a long time to remove a thread from a 'wait' queue and put it into run queue in a single KCall. For real-time, we break the operation into 2 KCalls - remove a thread from the wait queue and put it into the run queue. This state is used to indicate that a thread is being removed from the wait queue, but not yet put to the run queue.

I WAITSTATE is only meaningful when a thread is doing a Wait operation (WaitForMultipleObjects, EnterCriticalSection).

n WAITSTATE_PROCESSING - in the middle of a wait operation.

n WAITSTATE_SIGNALED - while processing the 'wait' operation, at least one of the objects the thread waits on is signaled, thus the wait should return right away (wait completed)

n WAITSTATE_BLOCKED - the wait operation has completely and the thread calling the wait function is about to be blocked. The run-state of the thread will be changed to RUNSTATE_BLOCKED on the next reschedule.

-- Bor-Ming

This posting is provided "AS IS" with no warranties, and confers no rights. You assume all risk for your use. ?2004 Microsoft Corporation. All rights reserved. "lu shen" <lshen@csce.unl.edu> wrote in message news:%23P05O2KZEHA.2388@TK2MSFTNGP09.phx.gbl... > I am reading the kernel source code. Can anyone answer the following > guestions regarding scheduling: > > 1. what is IpProxy in "struct thread" > 2. the following states are defined for a thread. What is the difference > between RUNSTATE BLOCKED, WAITSTATE BLOCKED, and WAITSTATE SIGNALLED ? > > #define RUNSTATE RUNNING 0 // must be 0 > #define RUNSTATE RUNNABLE 1 > #define RUNSTATE_BLOCKED 2 > #define RUNSTATE NEEDSRUN 3 // on way to being runnable > #define WAITSTATE_SIGNALLED 0 > #define WAITSTATE_PROCESSING 1 > #define WAITSTATE_BLOCKED 2 > >

2.6. Key functions, nextthread, kcnextthread, makerun, reschedule...

In %winceroot%\PRIVATE\WINCEOS\COREOS\NK\Kernel\schedule.c

makerun	insert a thread into the appropriate position in the RunList
nextthread	Dequeue the blocked states from
kcnextthread	Check if the current running thread runs out of quantum and schedules for a
	new thread to run according to the scheduling algorithm

In %winceroot%\PRIVATE\WINCEOS\COREOS\NK\Kernel\X86\fault.c, Naked Reschedule is called (after each slice interrupt?). It first check if there is runnable threads. if not go to power idle state. Otherwise, it will call nextthread, kenextthread to reschedule a runnable thread and then executes the thread (possibly needs context switching).

```
//-----
11
// Do a reschedule.
11
    (edi) = ptr to current thread or 0 to force a context reload
11
11
//--
                    _____
Naked
Reschedule()
{
       //DEBUGMSG(1,(L"*** reschedule ***\r\n"));
     asm {
        test
                 [KData].bPowerOff, OFFh // Was a PowerOff requested?
        jz
                short rsd10
                 [KData].bPowerOff, 0
        mov
                                          // Yes - do it
        call
                DoPowerOff
rsd10:
        sti
                word ptr ([KData].bResched), 1
        cmp
        jne
                short rsd11
        mov
                 word ptr ([KData].bResched), 0
                NextThread
        call
rsd11:
        cmp
                dword ptr ([KData].dwKCRes), 1
                short rsd12
        jne
                 dword ptr ([KData].dwKCRes), 0
        mov
                KCNextThread
        call
        cmp
                 dword ptr ([KData].dwKCRes), 1
        je
                short rsd10
rsd12:
        mov
                eax, [RunList.pth]
        test
                 eax, eax
                short rsd50
                                      // nothing to run
        jz
        cmp
                 eax, edi
        jne
                 short rsd20
                                      // redispatch the same thread
        imp
                RunThread
// Switch to a new thread's process context.
// Switching to a new thread. Update current process and address space
// information. Edit the ring0 stack pointer in the TSS to point to the
// new thread's register save area.
11
11
        (eax) = ptr to thread structure
rsd20: mov
                                          // Save thread pointer
                 edi, eax
                                                  // (esi) = thread handle
                esi, (THREAD)[eax].hTh
        mov
        push
                edi
                SetCPUASID
                                          // Sets hCurProc for us!
        call
                ecx
                                          // Clean up stack
        pop
                hCurThd, esi
                                         // set the current thread handle
        mov
                hCurThd, esi // set the current thread handle
PtrCurThd, edi // and the current thread pointer
ecx, [edi].tlsPtr // (ecx) = thread local storage ptr
[KData].lpvTls, ecx // set TLS pointer
        mov
        mov
        mov
```

cmp edi, g_CurFPUOwner jne SetTSBit clts MuckWithFSBase jmp SetTSBit: eax, CR0 mov eax, TS MASK test MuckWithFSBase jnz eax, TS MASK or CR0, eax mov MuckWithFSBase: mov edx, offset q aGlobalDescriptorTable+KGDT PCR ecx, FS_LIMIT+1 // (ecx) = ptr to NK_PCR base
word ptr [edx+2], cx // set low word of FS base sub word ptr [edx+2], cx mov shr ecx, 16 mov byte ptr [edx+4], cl // set third byte of FS base byte ptr [edx+7], ch // set high byte of FS base mov push fs рор fs ecx, [edi].ctx.TcxSs+4 // (ecx) = ptr to end of context save area lea [MainTSS].Esp0, ecx mov RunThread // Run thread pointed to by edi jmp // No threads ready to run. Call OEMIdle to shutdown the cpu. rsd50: cli word ptr ([KData].bResched), 1 cmp short DoReschedule je OEMIdle call mov byte ptr ([KData].bResched), 1 Reschedule jmp DoReschedule: sti

jmp Reschedule

}

}