CENG 313 Operating Systems Lab, Fall 2003

Labs included in this term:

➤ Lab #1 – Managing Multiple Tasks
➤ Lab #2 – Writing Multithreaded Software
➤ Lab #3 – Manipulating Kernel Objects
➤ Lab #4 – Thread Synchronization
➤ Lab #5 – Interprocess Communication
➤ Lab #6 – Virtual Memory
➤ Lab #7 – Memory-Mapped Files
➤ Lab #8 – Floppy Disk I/O

Objectives of the labs

➤ Learn the specific Windows NT mechanisms for the system-software operation and understand the design and implementation issues behind them.
LAB #1 Managing Multiple Tasks

Objectives

Windows NT code is executed in two modes: supervisor mode and user mode. Supervisor mode comprises NT kernel and NT Executive layers. User mode includes the NT subsystems. NT kernel holds the core operating system functions. NT Executive presents different managers for the different OS functions and supports the above subsystems. The NT subsystems are the extension components to the main system. The Win32 subsystem is the basic subsystem that presents common windows properties and the other subsystems are placed on it.

The design of Windows NT is object-oriented, all the processes and threads are represented as objects at the kernel and Executive levels.

After completing this lab, you will be able to:

➤ Understand the basic Windows NT organization.
➤ Use the available NT tools (such as the task manager, the process viewer, the performance monitor) for monitoring the basic system activities.
➤ Observe the status and activity of the processes, threads, and various other objects.

Materials Required

This lab will require the following:

➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Activities are defined in the problem statement of the exercise in Part II Chapter 1 of the book “Operating System Projects Using Windows NT” by Gary Nutt¹.

¹Nutt, Gary J., “Operating System Projects Using Windows NT”, Addison-Wesley, 1999
LAB #2 Writing Multithreaded Software

Objectives

The notion of a process in the traditional operating system designs comprises:

- The set of resources used when the process executes.
- The representation of the actual execution.

The schedulable unit of computation in this model is process.

In the modern operating systems, taking into consideration the fact that there can be multiple entries that can execute the same program while using the same resources; in order to minimize the context switching time the schedulable unit of computation is arranged as thread rather than process. In this new implementation, process represents the address space and thread represents the actual execution (with a minimum of its own allocated resources such as stack, thread status) Thus, a process may include multiple threads.

Materials Required

This lab will require the following:

- Windows NT/2000 operating system
- MS Visual C++ 6.0 IDE
- Administrator access to the machine.

Activities

Note: The required C code for this lab does not require the use of Windows graphics functions therefore the standard prototype is used for the main program:

```c
int main(int argc, char *argv[]);
```

In the Visual C++ environment, “Win32 Console Application” project type is selected for saving a standard C code. Go to “File/New” dialog, select “Projects” tab and highlight “Win32 Console Application”, give a name to the new project and press the “OK” button (Figure 1).
For setting the main program parameters before execution, go to “Project/Settings” dialog, select the “Debug” tab and fill in the “Program arguments” text box (Figure 2).

Figure 1 Creating a new Win32 Console Application.

Figure 2 Setting the main program arguments.
In the multithreaded programs “_beginthreadex” function is used. For using this Win32 API function, the compile/link environment must be adjusted.

For adjusting the environment:

Go to “Project/Settings” dialog, select the “C/C++” tab, to tell the compiler that the code is multithreaded change “/MLd” to “/MTd” (Figure 3).

![Figure 3 Settings for a multithreaded program.](image)

In the “Link” tab, include “/libcmtd.lib” or “/libcmtd.lib” to the list of libraries. Add the flag “/nodefaultlib:library” to the “Project Options” box (Figure 4).
Figure 4 Linker settings for the multithreaded programs.

Activities are defined in the problem statement of the exercise in Part II Chapter 2 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #3 Manipulating Kernel Objects

Objectives

Windows NT operating system mechanisms are implemented as objects at the kernel and the Executive levels. Processes and threads are represented with the process and thread objects. Processes and threads are used to create other kernel objects as well. Examples of the other kernel objects are event, mutex, semaphore, symbolic link, waitable timer etc. Kernel objects are referenced from the user mode using the object handles. A process is given a handle after creating or referencing a kernel object. Handle information is kept as part of the process descriptor. Handle is just an index to the handle table in the process description. Kernel objects include an object header and a body. Object header contains fields as object name, security descriptor, handle count, reference count and object type. Object body is specific to the object type. A process references a kernel object provided that the process knows the name of the object and has security access to the object.

A waitable timer object, a new kernel object type, provides a new tool for controlling execution based on asynchronous events.

After completing this lab, you will be able to:

➤ Understand kernel objects and handles
➤ Understand the model for managing objects
➤ Use Waitable timers.

Materials Required

This lab will require the following:
➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Note: Since waitable timers are not implemented in versions of NT earlier than 4.0 and they are not in Windows 9x or CE, a compiler flag should be included to recognize waitable timer related API functions. The compiler flag is set in one of two ways:

1. Add a new line to the source code
   #define _WIN32_WINNT 0x0400

2. In the Visual C++ 6.0 environment, go to “Project/Settings” dialog, select “C/C++” tab and add “/D _WIN32_WINNT 0x0400” to the “Project Options” box (Figure 5).
Figure 5 Settings for recognizing the Timer functions.

Activities are defined in the problem statement of the exercise in Part II Chapter 3 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #4 Thread Synchronization

Objectives

Windows NT permits the multi-threaded operation within or across process address spaces. A synchronization mechanism is required to coordinate the multiple, independent threads’ work in order to ensure safe access to the shared variables. Blocks of code that read or write shared variables are said to be executing in a critical section of code. Various OS mechanisms try to enforce mutual exclusion, whereby if one thread is in its critical section, then other threads are excluded from entering their critical sections as long as the first remains in the critical section. In Windows NT, thread synchronization for coordination and mutual exclusion is implemented using the following Executive kernel objects:

- **Event**
  Notifies waiting threads of the occurrence of a particular event

- **Mutex**
  Coordinates mutually exclusive access to a shared resource

- **Semaphore**
  Controls a shared resource that can support limited number of requests

- **Timers**
  Controls execution based on asynchronous events

After completing this lab, you will be able to:

➤ Understand Windows NT synchronization principles
➤ Understand critical sections
➤ Understand synchronization mechanisms

Materials Required

This lab will require the following:

➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Note: A compiler flag should be included to recognize waitable timer related API functions as explained in the previous lab.

Activities are defined in the problem statement of the exercise in Part II Chapter 4 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #5 Interprocess Communication

Objectives

Threads of the same process use the same address space so it is easy for them to share information among themselves. Threads operating in the different address spaces, on the other hand, are dependent on the interprocess communication mechanisms to share information. NT supports a shared-memory model within a computer and a few message-based mechanisms within a computer and across a network to provide interprocess communication.

The Local Procedure Call (LPC) mechanism is a message-based mechanism, which is built on top of kernel port objects and used for high-speed message-passing by client and server processes within the Executive and the subsystems.

Pipes are used to provide communication between different processes as well. NT supports two varieties of pipes: anonymous and named. An anonymous pipe is a half-duplex, character-based IPC mechanism. A named pipe is a full-duplex, character based IPC mechanism that can be used over a network.

After completing this lab, you will be able to:

➤ Understand general Windows NT mechanisms that support IPC
➤ Understand the high-level File I/O Model
➤ Use pipes

Materials Required

This lab will require the following:
➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Activities are defined in the problem statement of the exercise in Part II Chapter 5 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #6 Virtual Memory

Objectives

Windows NT uses a paging system to reference and allocate primary memory space. In a paging system, each process has its own virtual address space that it uses to reference the contents of memory locations. Windows NT has a fixed-size virtual address space (4 GB) containing two parts of equal size: one for referencing user space memory objects and one for referencing kernel space memory objects. Although the kernel space portion of the address space exists in a process’s virtual address space, it can only be referenced by a thread executing in kernel mode.

The Virtual Memory (VM) Manager is an Executive-level component that controls the virtual memory. It uses a page-based management scheme with a page size of 4 KB. Pages of data that are assigned to a process but are not in physical memory are stored in the paging file on disk. The VM Manager uses 32 bit addresses to reference the individual pages of 4K so each process has a 4 GB virtual address space.

The Win32 API provides several functions for inspecting the virtual memory configuration and dynamically controlling the way the address space is used.

After completing this lab, you will be able to:

➤ Understand the organization of Windows NT’s virtual memory system
➤ Control the virtual memory space
➤ Write a monitoring and reporting tool

Materials Required

This lab will require the following:
➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Activities are defined in the problem statement of the exercise in Part II Chapter 7 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #7 Memory-Mapped Files

Objectives

Memory-mapped files are the primary internal OS mechanism used to implement sharing across address spaces. In Windows NT, memory-mapped files are implemented at the Virtual Memory Manager. To use a memory-mapped file, it’s necessary to

- Obtain a handle to the file by creating or opening it
- Reserve virtual addresses for the file,
- Establish a mapping between the file and the virtual address space of the process.

After completing this lab, you will be able to:

➤ Construct an application using memory-mapped files for IPC
➤ Understand memory-mapped file design

Materials Required

This lab will require the following:
➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Activities are defined in the problem statement of the exercise in Part II Chapter 8 of the book “Operating System Projects Using Windows NT” by Gary Nutt.
LAB #8 Floppy Disk I/O

Objectives

Floppy disks and hard disks are rotating magnetic media. On these types of media, information is stored on the surfaces; read and written by read-write heads. On disk surfaces, information addressing is done according to two criteria: a radius R and an angle of rotation θ. Radius R represents a track and angle of rotation θ represents a sector. On a 1.44 MB floppy disk there are 80 tracks each containing 18 sectors. In a single I/O operation, an individual sector is read or written.

In disk terminology, hard formatting means defining the number of tracks and sectors on a disk and soft formatting means writing particular kinds of information into particular sectors on the floppy disk. Before an I/O operation, a disk must be soft formatted. Windows NT uses the MS-DOS soft format for disks. The MS-DOS Basic I/O System additionally defines logical sector numbers for addressing. In this addressing:

Logical sector 0 corresponds to the sector at surface 0 (on a hard disk), track 0, sector 1 of the disk.

Logical sector 1 corresponds to the sector at surface 0 (on a hard disk), track 0, sector 2 of the disk

After completing this lab, you will be able to:

➤ Understand the organization of a MS-DOS floppy disk
➤ Interact directly with the floppy disk.
➤ Acquire the background for managing the MS-DOS file allocation table

Materials Required

This lab will require the following:
➤ Windows NT/2000 operating system
➤ MS Visual C++ 6.0 IDE
➤ Administrator access to the machine.

Activities

Activities are defined in the problem statement of the exercise in Part II Chapter 10 of the book “Operating System Projects Using Windows NT” by Gary Nutt.