Operating Systems Week 11: Case Studies

In this lecture we examine 2 case studies: the Unix and Windows XP operating Systems:

Unix
The originator of Unix was Ken Thompson of Bell Labs who wrote a version of Multics, a very early time sharing system. The name was changed (partly as a joke) to Unics and it subsequently became Unix. Thompson was later joined by Dennis Ritchie - the pair wrote a landmark paper in the mid 70's. Unix was originally written in assembler - later and current versions are in C. Linux was developed by Linus Torvalds as a Unix ‘clone’ which contained a number of features lacking in Unix. The first version was released in 1991. We present here a very short description of the system - with much omitted. POSIX is a standard for Unix developed by IEEE which defines a minimal system call interface that conformant Unix versions must support. Figure 1 shows the layers in a Unix system - the features in the figure are present in all Unix systems.

The bottom layer is hardware and running on the bare hardware is the operating system. This controls the hardware and provides a system call interface to all the programs. Programs make system calls by putting the arguments in registers (or on the stack) and issuing instructions to switch from user to kernel mode (trap instructions). These procedures are written in assembler - which is called from the C code. Thus to execute a ‘read’ the C program calls the assembler ‘read’ from the library. POSIX tells which
library procedures a conformant system must supply, what their parameters are, what they must do and what they must return. User programs in Unix include the shell, compilers, editors, text processing programs and file manipulation utilities. Unix also fully supports a graphical environment.

The Kernel Structure
There are many versions of Unix and Figure 2 shows a representative, 4.4BSD - the features shown are present in many versions of Unix.

The bottom layer of the kernel consists of the device (either character or block) drivers plus process dispatching. Network devices are treated separately. **Process dispatching** occurs when an interrupt happens. The low level code stops the current process, saves its state in the kernel process table and starts the appropriate driver. Further process dispatching (which is written in assembler) occurs to restart the interrupted process (or another process). The functions of the various components are as follows - starting from the left:

**Input** can be either ‘raw’ from the terminal or ‘cooked’ which is more line oriented.

**Network Software** is often modular with different devices and protocols supported. A routing function makes sure that the right packet goes to the right device or protocol handler. The protocol stack always includes IP and TCP (and others). Overlaying the network is the socket interface which allows programs to create sockets for particular networks and protocols, getting back a file descriptor for later use.

![Figure 2: Structure of the 4.4BSD kernel](image)
**Files and Memory:** the buffer and Page caches appear next and above these are the file systems and virtual memory system. All the paging logic is here such as the page replacement algorithm. Every Unix process has an address space consisting of three segments: text, data, stack. Memory mapping was originally based on swapping - processes were swapped into and out of main memory. However virtually all Unix systems now have paging.

**Process management:** In Unix each file has a process identifier (PID) - we have looked at process creation etc. in Lecture 2 and Tutorial 2. Process scheduling is based on a multilevel queue structure (see Figure 3)

**System Interface:** The top layer of the kernel is the interface into the system. The system call interface is the recipient of system calls and these are directed to one of the lower modules. The entrance for traps and interrupts includes signals, page faults, processor exceptions of all kinds and I/O interrupts.

---

Figure 3: The Unix scheduler is based on a multilevel queue structure.
Figure 4: The structure of Windows 2000 (slightly simplified). The shaded area is the executive. The boxes indicated by D are device drivers. The service processes are system daemons.

Windows XP
The “New Technology” (NT) portable operating system was developed in 1988. There is a whole family of ‘NT’ operating systems and Windows XP is one member. Windows XP was released both as an update to the Windows 2000 desktop OS and a replacement for Windows 95/98. (See also Week on OS file systems.) The following describes Windows 2000 with a brief resume at the end about Windows XP.

Windows 2000 is characterised by the use of objects for they provide a uniform data structure for the OS. The operating system is huge - over 29 million lines of C code. This can be compared with 1 million in a 1999 version of Linux and 2 million in Solaris. However the entire windows system and GUI is part of the kernel in Windows but simply a user process in Unix. Figure 4 shows the structure. We shall mainly be examining the lower two layers.

HAL is written in C - it is intended to hide many of the machine dependencies since there are small hardware differences between the different hardware platforms (Pentium and Intel IA-64). It is the job of HAL to present the rest of the operating system with abstract hardware devices.

The services chosen for inclusion in HAL are those that relate to the chip set on the parentboard and which vary from machine to machine. The Windows CD-ROM contains many versions of HAL and the appropriate one is selected at installation. A summary of what HAL does is provided
in Figure 5. The HAL does NOT manage specific hardware devices such as keyboards, mice etc. As an example of what it DOES do - consider memory-mapped vs I/O ports. Some machines have one and some the other. HAL offers flexibility - three procedures for reading the device registers (integers) and another three for writing them:

```c
uc = READ_PORT_UCHAR(port); WRITE_PORT_UCHAR(port,uc); /* 8 bits */
us = READ_PORT_USHORT(port); WRITE_PORT_USHORT(port,us); /* 16 bits */
ul = READ_PORT_ULONG(port); WRITE_PORT_ULONG(port,ul); /* 32 bits */
```

It is up to HAL to decide whether memory-mapped I/O is needed.

In many cases two or more devices have the same bus address and HAL provides a service for identifying devices by mapping bus-relative addresses onto system-wide logical addresses. These logical addresses are analogous to the handles the operating system gives user programs to files and other system resources.

**The Kernel** In Unix the ‘kernel’ is everything running in kernel mode. (In the Figure this is shaded.) However in Windows 2000 only a part of this is called “the kernel” by Microsoft. The purpose of this kernel is to make the rest of the operating system independent of the hardware and thus highly portable. It picks up where HAL leaves off. Hardware is accessed via HAL and the kernel builds higher level abstractions from the low level HAL services. For example the HAL has calls to associate interrupt service procedures with interrupts and set their priorities but does little else in that area. The kernel, in contrast provides a complete mechanism for doing context switches. Thus it saves all the CPU registers, changes the page tables, flushes the CPU cache etc. It then sets up the new thread’s memory map and loads its registers so that the new thread can start running. The code for thread scheduling is also in the kernel - for example if a quantum has run out or after I/O interrupt completes the kernel chooses the new thread and does the context switch necessary to run it.
Control Objects are those objects that control the system. Examples are as follows:

Deferred Procedure Call (DPC) object used to split off the non-time-critical part of an interrupt service procedure from the time critical part. For example after a key is struck, the keyboard interrupt service procedure reads the key code from the register and reenables the interrupt but does not need to process the key immediately. The bulk of the processing is left till later - and as long as it is done within 100msec the user will be none the wiser. The DPC queue is the mechanism for remembering that there is more work to be done.

Asynchronous Procedure Call (APC) object are like DPC objects - but they execute in the context of a specific process. When processing a key press it does not matter whose context a DPC runs in as the key code will be put in a kernel buffer. However if we are copying a buffer from kernel space to a buffer in some user process' address space then the copying procedure needs to run in the receivers context - i.e. the user process in question.

dispatcher objects include semaphores, mutexes, events, waitable timers etc. These are handled (in part) in the kernel because they are involved closely in thread scheduling - a kernel task.

The Executive This is the shaded area in Figure 4. Although there are boundaries in the diagram there are no hard boundaries in the system. The components designates a collection of procedures which work together to accomplish some goal. However the procures from each collection call each other extensively. Most of the “managers” are not independent threads - with the exception of the power and plug-and-play and a few others.

The I/O Manager manages I/O devices - providing the rest of the system with device independent I/O. It is also home to all the device drivers. These are indicated by ‘D’ on the diagram.

The process manager handles processes and threads, including their creation and termination.

Memory Manager Windows 2000 supports a 4-GB demand-paged address space per process. Segmentation is not supported. The memory manager manages the mapping of virtual pages onto physical page frames. Theoretically page sizes can be any power of 2 up to 64KB. On the Pentium they are fixed at 4 KB; on the Itanium they can be 8 KB or 16 KB. In addition the operating system itself can use 4 KB pages to reduce table space consumed.

The security Manager enforces Windows 2000’s elaborate security mechanism, which meets the U.S. Dept. of Defence’s Orange Book C2 requirements.

The Win32 Application Programming Interface (API) Microsoft has never made its system calls public - however it does define a set of calls called the Win32 API which are fully documented. The Win32 program can be seen as part of the top of the hierarchy in Figure 4.. The Win32 philosophy is quite different from the Unix philosophy. In Unix the system calls are all
publicly known and form a minimal interface; removing even one would reduce the operating system functionality. The POSIX subsystem is designed to run POSIX applications - these can be started by the Win32 subsystem or by another POSIX application. The BIOS (Basic Input Output System) contains low-level I/O software, including procedures to read the keyboard, write to screen and do disk I/O. The BIOS is started when the computer is booted.

Windows XP is an improved version of Windows 2000 - the following is a brief description of the differences. Windows XP improves the facilities for catching programmer errors in user-level code. There are also new facilities for monitoring the health of the PC including downloading fixes for problems before they are encountered by users. There is also a new GUI which differs from Windows 2000 and is designed for easier use. There is also a much higher compatibility with Windows 95/98 applications. Also POSIX support is much improved - a new subsystem Interix is now available. Most available Unix-compatible software compiles and runs under Interix without modification.