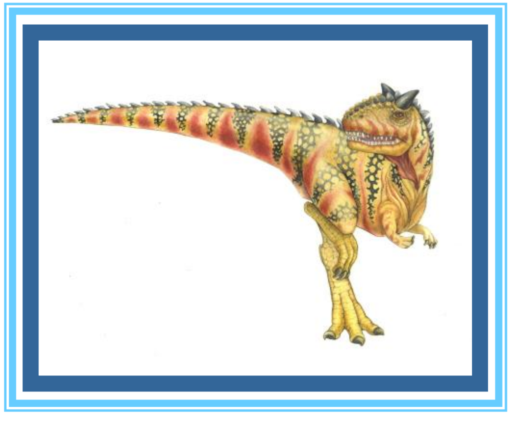
Chapter 2: Operating-System Structures



Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Operating System Debugging
- Operating System Generation
- System Boot

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

Operating System Services

Operating systems provide an environment for execution of programs and services to programs and users

One set of operating-system services provides functions that are helpful to the user:

- **User interface** - Almost all operating systems have a user interface (**UI**).
  - Varies between **Command-Line** **(CLI)**, **Graphics User Interface** **(GUI)**, **Batch**
- **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device

Operating System Services (Cont.)

One set of operating-system services provides functions that are helpful to the user (Cont.):

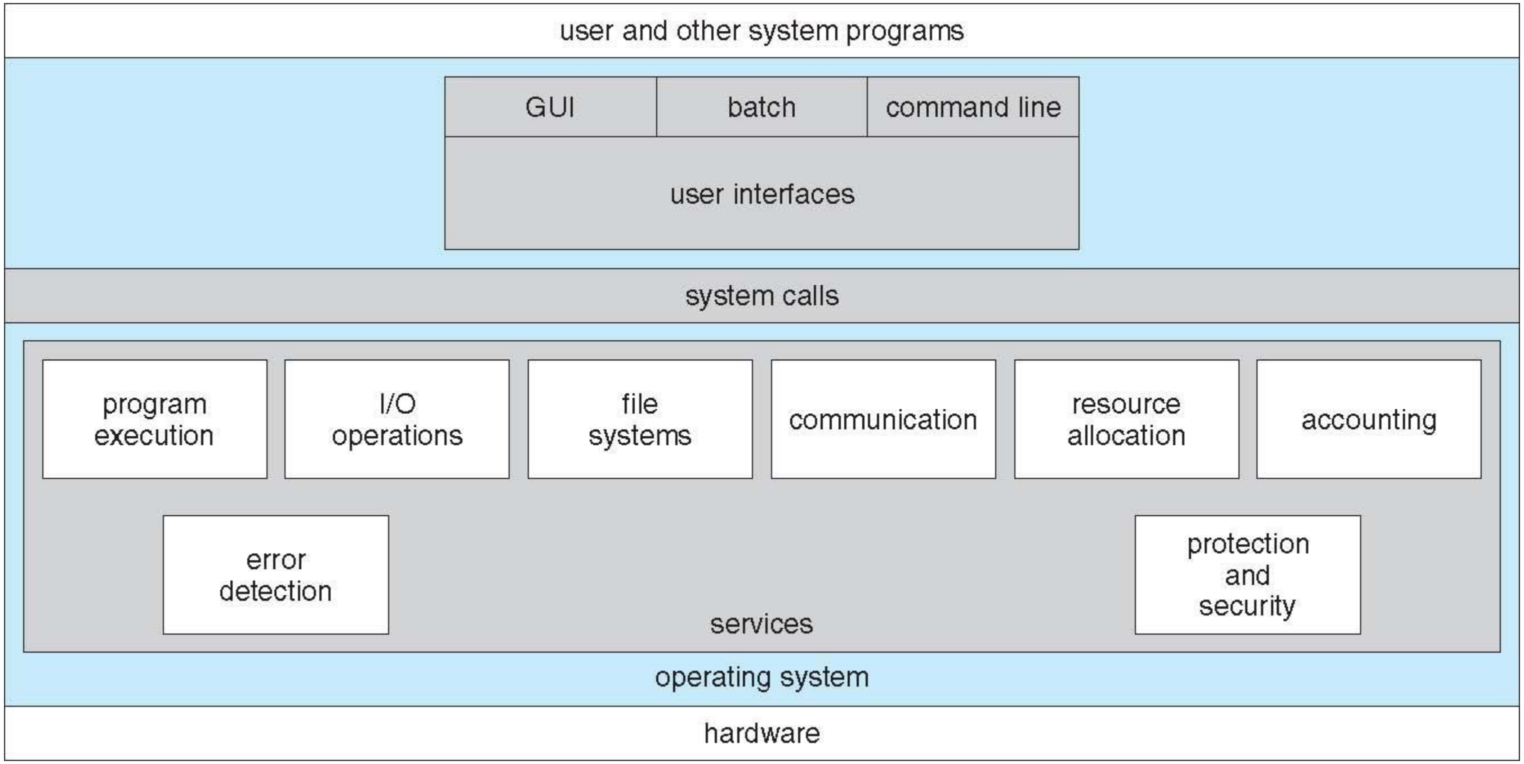
- **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing

- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



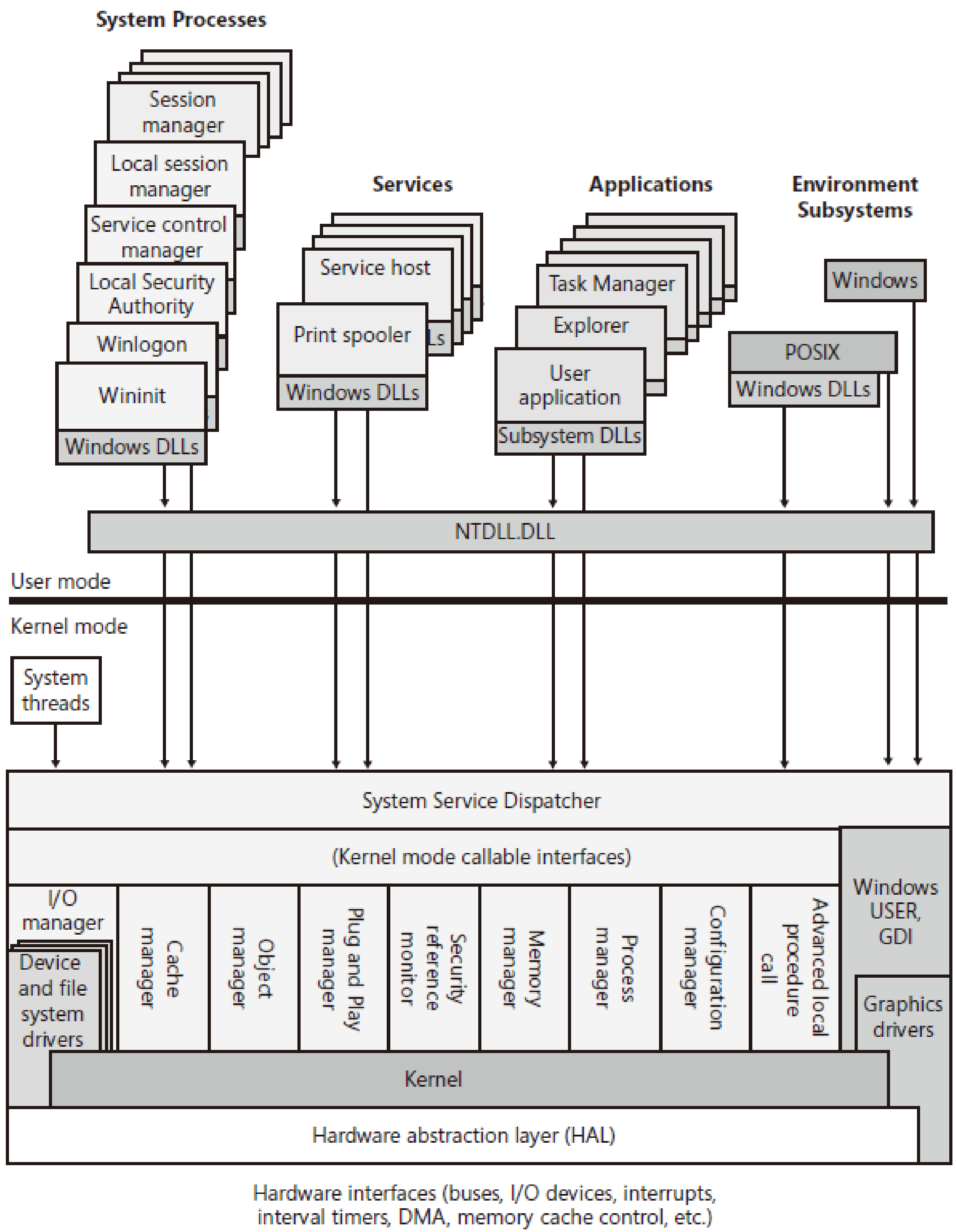


FIGURE 2-3 Windows architecture

User Operating System Interface - CLI

CLI or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter

```
Default
New  Info  Close  Execute  Bookmarks
Default   Default
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10        cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

User Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands.

The Mac OS X GUI

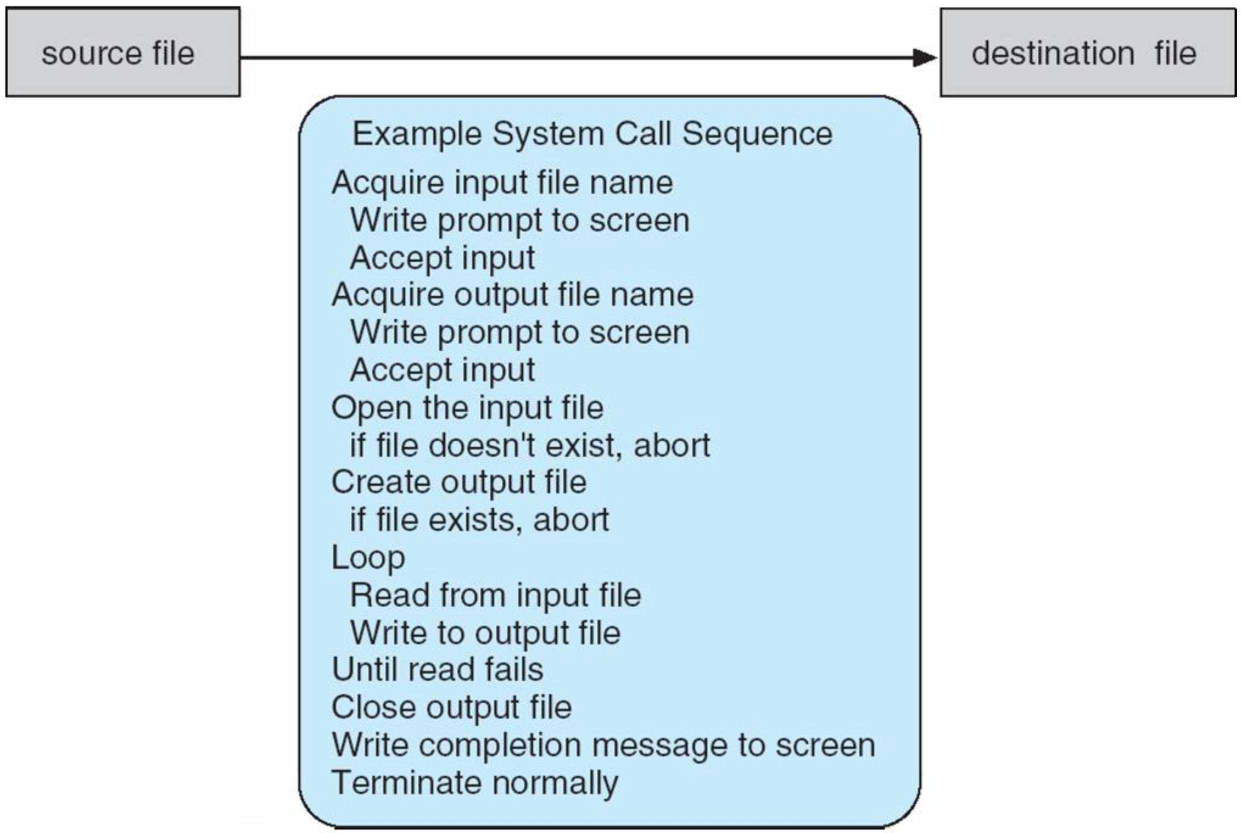
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout this text are generic

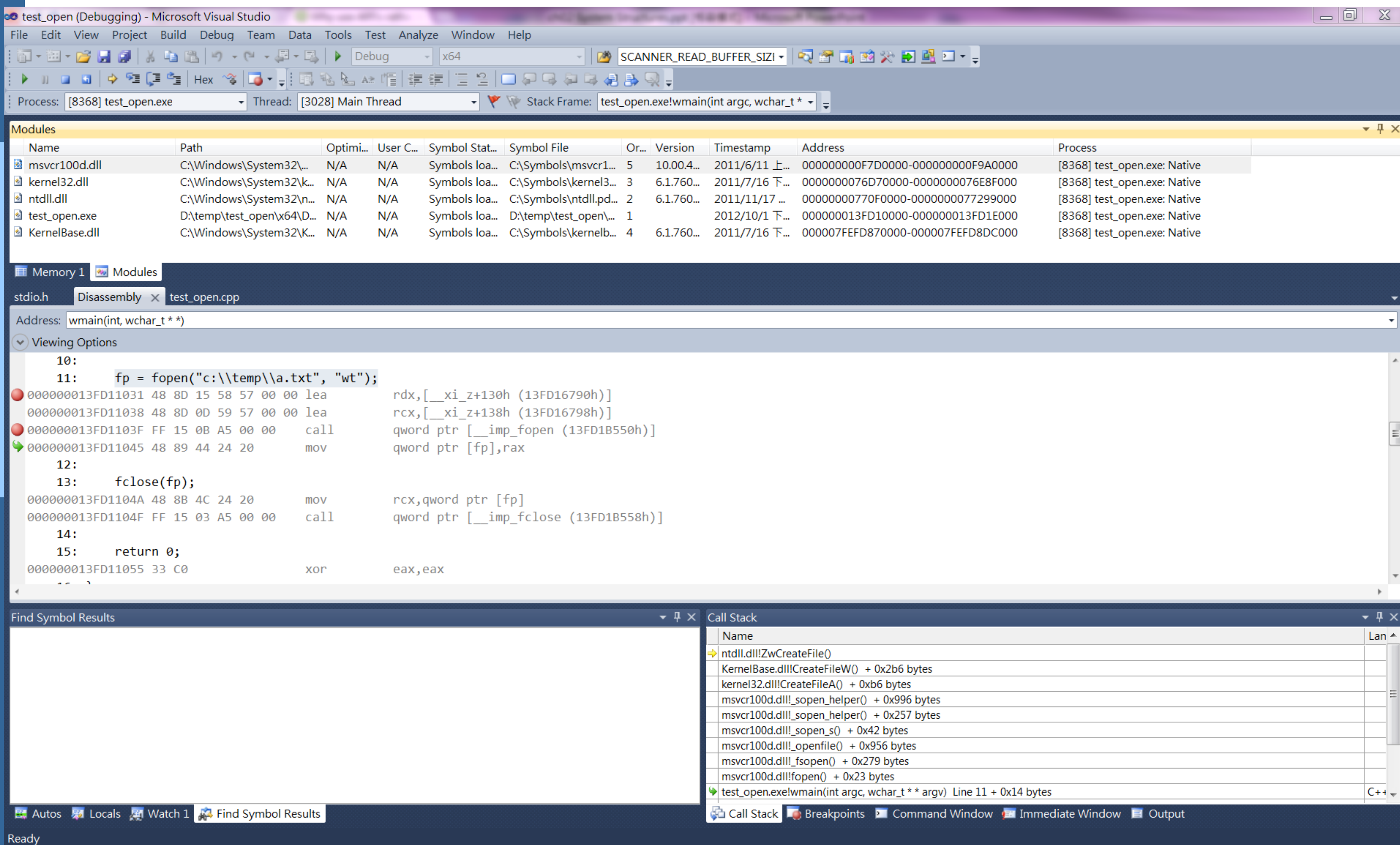
Example of System Calls

System call sequence to copy the contents of one file to another file

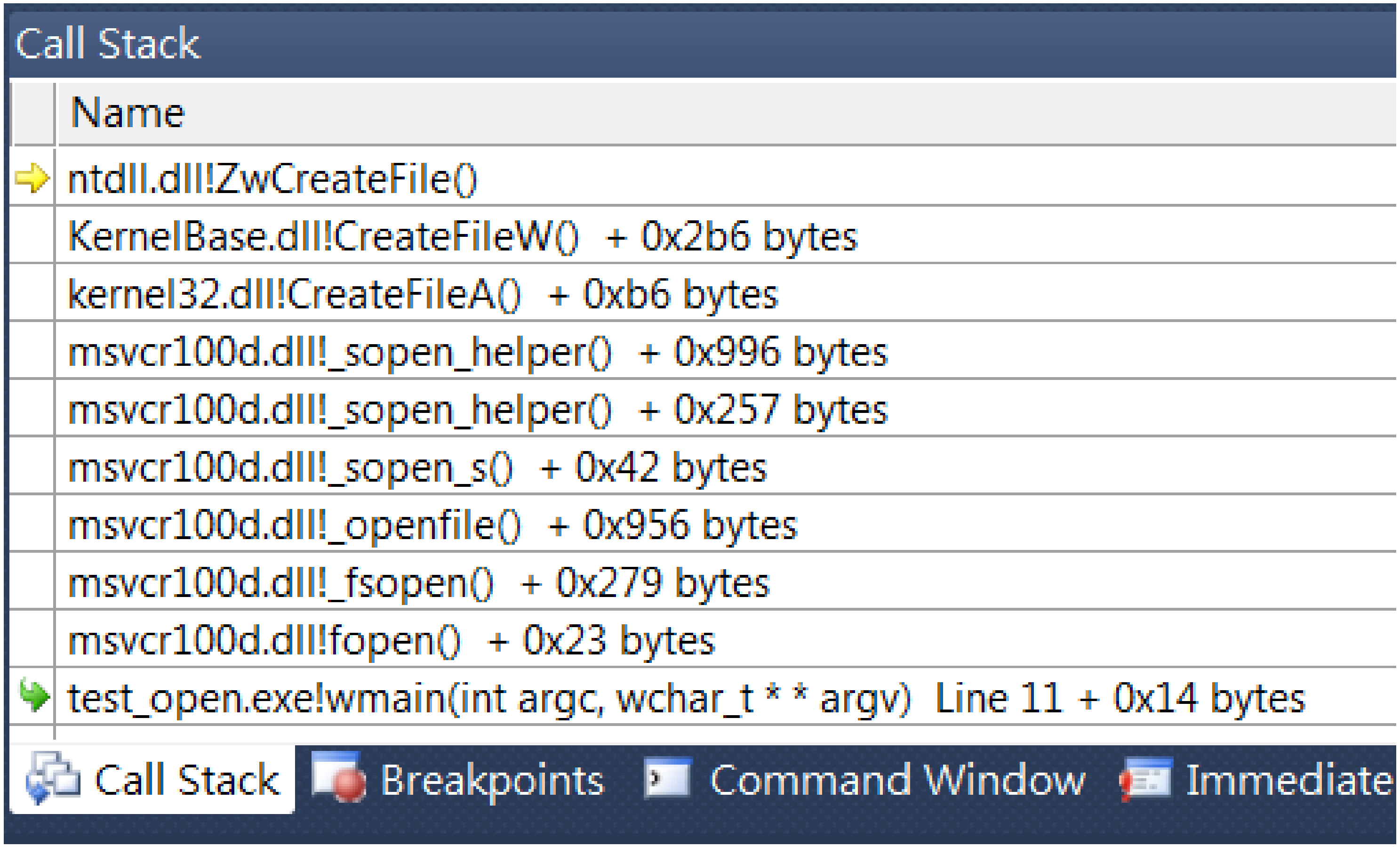


System Calls

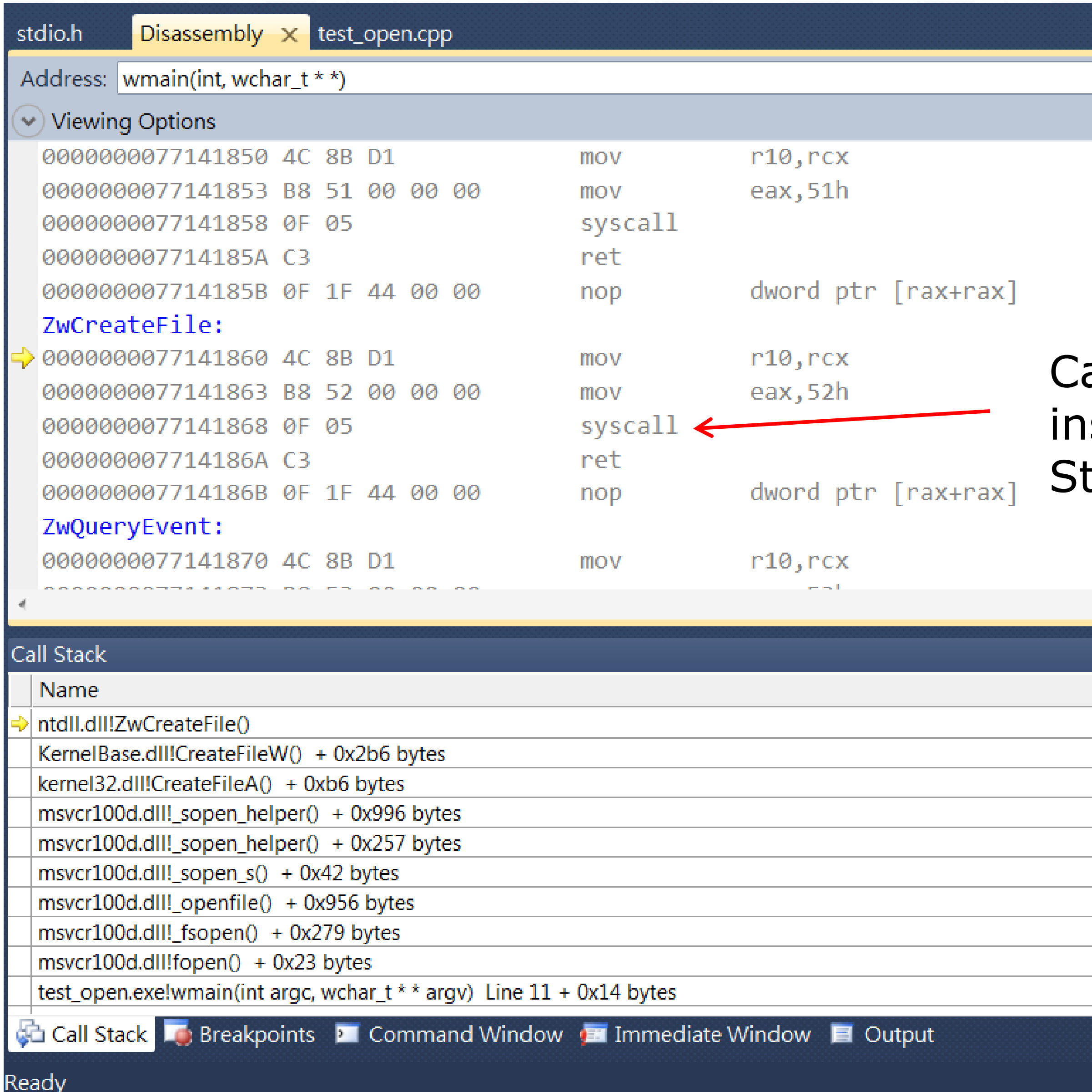
This is what happened in user mode when calling *fopen* on Win7 x86_64



System Calls



System Calls



Can’t step into this instruction in Visual Studio

System Calls

SYSCALL—Fast System Call

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 05	SYSCALL	NP	Valid	Invalid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

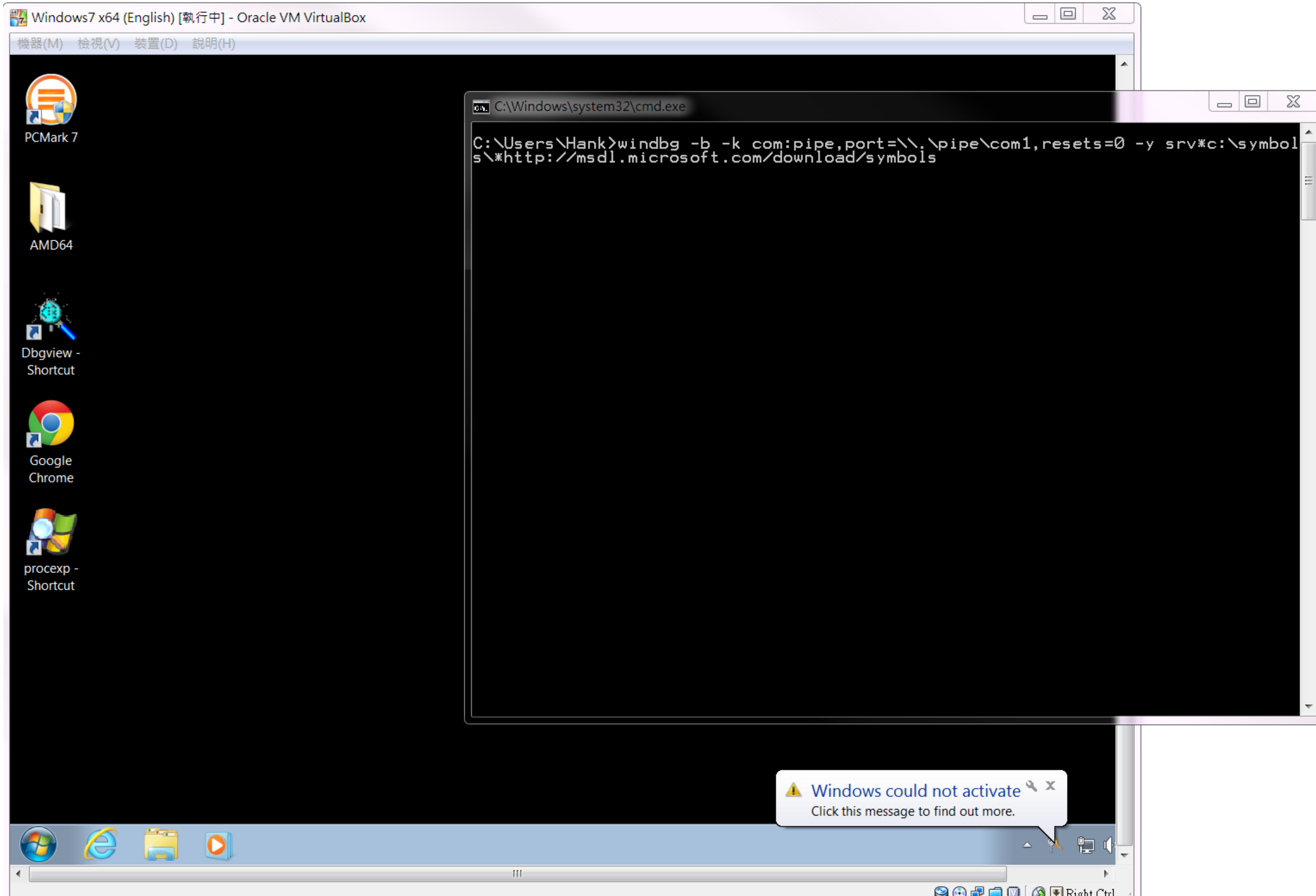
SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR (after saving the address of the instruction following SYSCALL into RCX). (The WRMSR instruction ensures that the IA32_LSTAR MSR always contain a canonical address.)

SYSCALL also saves RFLAGS into R11 and then masks RFLAGS using the IA32_FMASK MSR (MSR address C0000084H); specifically, the processor clears in RFLAGS every bit corresponding to a bit that is set in the IA32_FMASK MSR.

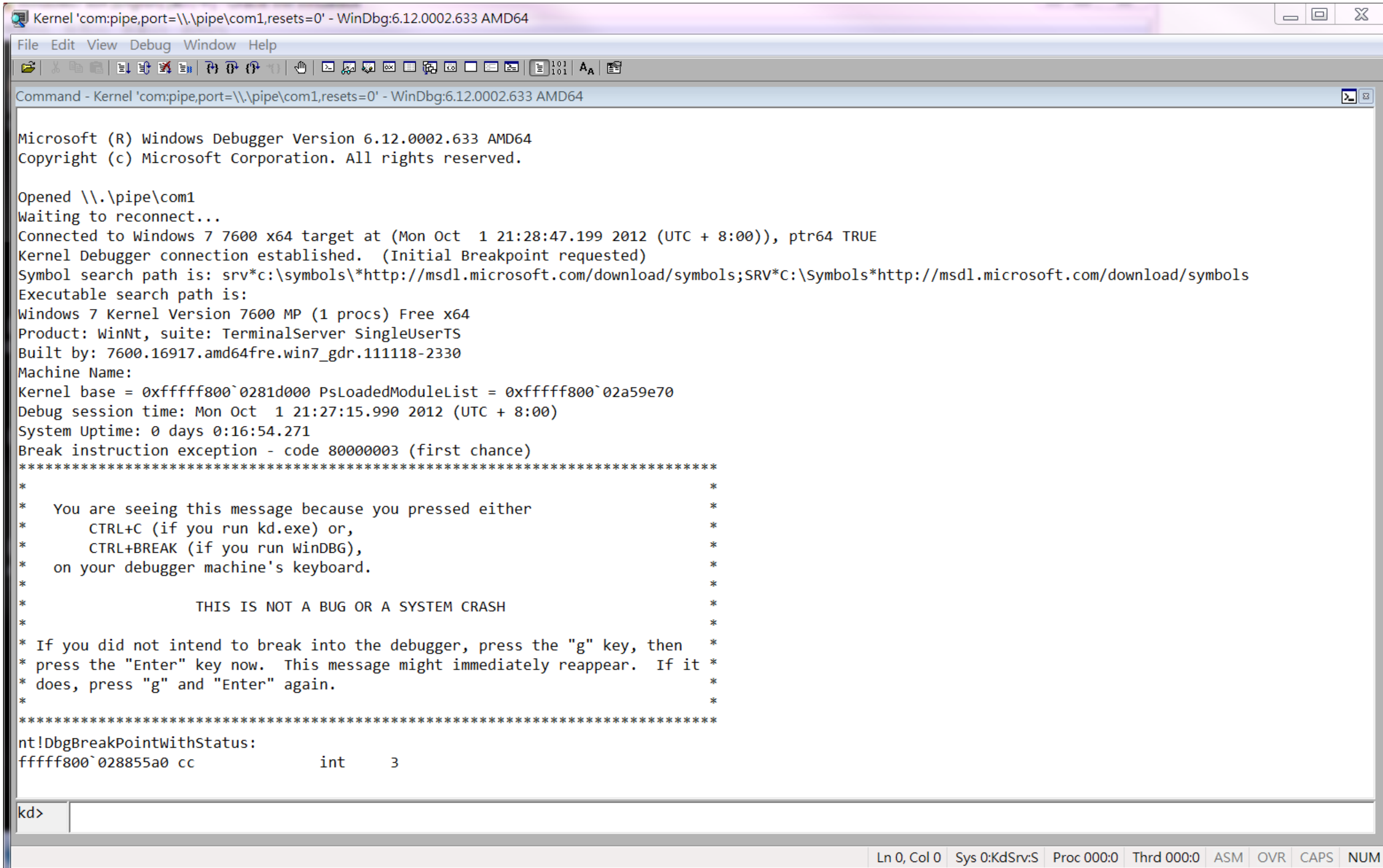
SYSCALL loads the CS and SS selectors with values derived from bits 47:32 of the IA32_STAR MSR. However, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSCALL instruction does not ensure this correspondence.

The SYSCALL instruction does not save the stack pointer (RSP). If the OS system-call handler will change the stack pointer, it is the responsibility of software to save the previous value of the stack pointer. This might be done prior to executing SYSCALL, with software restoring the stack pointer with the instruction following SYSCALL (which will be executed after SYSRET). Alternatively, the OS system-call handler may save the stack pointer and restore it before executing SYSRET.

System Calls



System Calls



Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

File Edit View Debug Window Help

Command - Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

```
Microsoft (R) Windows Debugger Version 6.12.0002.633 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\pipe\com1
Waiting to reconnect...
Connected to Windows 7 7600 x64 target at (Mon Oct  1 21:28:47.199 2012 (UTC + 8:00)), ptr64 TRUE
Kernel Debugger connection established.  (Initial Breakpoint requested)
Symbol search path is: srv*c:\symbols\*http://msdl.microsoft.com/download/symbols;SRV*C:\Symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Kernel Version 7600 MP (1 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 7600.16917.amd64fre.win7_gdr.111118-2330
Machine Name:
Kernel base = 0xfffff800`0281d000 PsLoadedModuleList = 0xfffff800`02a59e70
Debug session time: Mon Oct  1 21:27:15.990 2012 (UTC + 8:00)
System Uptime: 0 days 0:16:54.271
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run kd.exe) or,                                        *
*       CTRL+BREAK (if you run WinDBG),                                       *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!DbgBreakPointWithStatus:
fffff800`028855a0 cc              int     3

kd>
```

Ln 0, Col 0 | Sys 0:KdSrv:S | Proc 000:0 | Thrd 000:0 | ASM | OVR | CAPS | NUM

System Calls

Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

File Edit View Debug Window Help

Command - Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

```
kd> rdmsr c0000082
msr[c0000082] = fffff800`0288c500    <-- MSR[IA32_LSTAR]
kd> u fffff800`0288c500 L35
nt!KiSystemCall64:
fffff800`0288c500 0f01f8            swapgs
fffff800`0288c503 654889242510000000 mov   qword ptr gs:[10h],rsp
fffff800`0288c50c 65488b2425a8010000 mov   rsp,qword ptr gs:[1A8h]
fffff800`0288c515 6a2b              push    2Bh
fffff800`0288c517 65ff342510000000 push    qword ptr gs:[10h]
fffff800`0288c51f 4153              push    r11
fffff800`0288c521 6a33              push    33h
fffff800`0288c523 51                push    rcx
fffff800`0288c524 498bca            mov     rcx,r10
fffff800`0288c527 4883ec08          sub     rsp,8
fffff800`0288c52b 55                push    rbp
fffff800`0288c52c 4881ec58010000    sub     rsp,158h
fffff800`0288c533 488dac2480000000 lea     rbp,[rsp+80h]
fffff800`0288c53b 48899dc0000000    mov     qword ptr [rbp+0C0h],rbx
fffff800`0288c542 4889bdc8000000    mov     qword ptr [rbp+0C8h],rdi
fffff800`0288c549 4889b5d0000000    mov     qword ptr [rbp+0D0h],rsi
fffff800`0288c550 c645ab02          mov     byte ptr [rbp-55h],2
fffff800`0288c554 65488b1c2588010000 mov   rbx,qword ptr gs:[188h]
fffff800`0288c55d 0f0d8bd8010000    prefetchw [rbx+1D8h]
fffff800`0288c564 0fae5dac          stmxcsr dword ptr [rbp-54h]
fffff800`0288c568 650fae142580010000 ldmxcsr dword ptr gs:[180h]
fffff800`0288c571 807b0300          cmp     byte ptr [rbx+3],0
fffff800`0288c575 66c785800000000000 mov   word ptr [rbp+80h],0
fffff800`0288c57e 0f848c000000      je      nt!KiSystemCall64+0x110 (fffff800`0288c610)
fffff800`0288c584 488945b0          mov     qword ptr [rbp-50h],rax
fffff800`0288c588 48894db8          mov     qword ptr [rbp-48h],rcx
fffff800`0288c58c 488955c0          mov     qword ptr [rbp-40h],rdx
```

System Calls

Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

File Edit View Debug Window Help

Command - Kernel 'com:pipe,port=\\.\pipe\com1,resets=0' - WinDbg:6.12.0002.633 AMD64

```
kd> u nt!NtCreateFile L30
nt!NtCreateFile:
fffff800`02b97e70 4c8bdc           mov     r11,rsp
fffff800`02b97e73 4881ec88000000   sub     rsp,88h
fffff800`02b97e7a 33c0             xor     eax,eax
fffff800`02b97e7c 498943f0         mov     qword ptr [r11-10h],rax
fffff800`02b97e80 c744247020000000 mov     dword ptr [rsp+70h],20h
fffff800`02b97e88 89442468         mov     dword ptr [rsp+68h],eax
fffff800`02b97e8c 498943d8         mov     qword ptr [r11-28h],rax
fffff800`02b97e90 89442458         mov     dword ptr [rsp+58h],eax
fffff800`02b97e94 8b8424e0000000   mov     eax,dword ptr [rsp+0E0h]
fffff800`02b97e9b 89442450         mov     dword ptr [rsp+50h],eax
fffff800`02b97e9f 488b8424d8000000 mov     rax,qword ptr [rsp+0D8h]
fffff800`02b97ea7 498943c0         mov     qword ptr [r11-40h],rax
fffff800`02b97eab 8b8424d0000000   mov     eax,dword ptr [rsp+0D0h]
fffff800`02b97eb2 89442440         mov     dword ptr [rsp+40h],eax
fffff800`02b97eb6 8b8424c8000000   mov     eax,dword ptr [rsp+0C8h]
fffff800`02b97ebd 89442438         mov     dword ptr [rsp+38h],eax
fffff800`02b97ec1 8b8424c0000000   mov     eax,dword ptr [rsp+0C0h]
fffff800`02b97ec8 89442430         mov     dword ptr [rsp+30h],eax
fffff800`02b97ecc 8b8424b8000000   mov     eax,dword ptr [rsp+0B8h]
fffff800`02b97ed3 89442428         mov     dword ptr [rsp+28h],eax
fffff800`02b97ed7 488b8424b0000000 mov     rax,qword ptr [rsp+0B0h]
fffff800`02b97edf 49894398         mov     qword ptr [r11-68h],rax
fffff800`02b97ee3 e8c85fffff       call    nt!IopCreateFile (fffff800`02b8deb0)
fffff800`02b97ee8 4881c488000000   add     rsp,88h
fffff800`02b97eef c3               ret
fffff800`02b97ef0 90               nop
fffff800`02b97ef1 90               nop
fffff800`02b97ef2 90               nop
fffff800`02b97ef3 90               nop
fffff800`02b97ef4 90               nop
```

System Calls

User Mode

```
test_open.exe!wmain(int argc, wchar_t * * argv)  Line 11 + 0x14
bytes
msvcr100d.dll!fopen()  + 0x23 bytes
msvcr100d.dll!_fsopen()  + 0x279 bytes
msvcr100d.dll!_openfile()  + 0x956 bytes
msvcr100d.dll!_sopen_s()  + 0x42 bytes
msvcr100d.dll!_sopen_helper()  + 0x257 bytes
msvcr100d.dll!_sopen_helper()  + 0x996 bytes
kernel32.dll!CreateFileA()  + 0xb6 bytes
KernelBase.dll!CreateFileW()  + 0x2b6 bytes
ntdll.dll!ZwCreateFile()  + 0xa bytes
syscall
```

Kernel Mode

```
nt!KiSystemCall64
…..
nt!NtCreateFile
```

System Calls (Linux x86_64)

b.c

```c
#include <stdio.h>

void main()
{

        FILE *fp;

        fp = fopen("/tmp/a.txt", "wt");

        getchar();

        fclose(fp);

}
```

```
[hank@Maestro t]$ gcc -g b.c
[hank@Maestro t]$ ls -al a.out
-rwxrwxr-x. 1 hank hank 9109 Sep 30 23:39 a.out
[hank@Maestro t]$
[hank@Maestro t]$
```

System Calls

```
File   Edit   View   Search   Terminal   Help
[hank@Maestro t]$ gdb ./a.out
GNU gdb (GDB) Fedora (7.4.50.20120120-50.fc17)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/hank/t/a.out...done.
(gdb) list
1       #include <stdio.h>
2
3       void main()
4       {
5
6               FILE *fp;
7
8               fp = fopen("/tmp/a.txt", "wt");
9
10              getchar();
(gdb) break 8
Breakpoint 1 at 0x400584: file ./b.c, line 8.
(gdb) r
Starting program: /home/hank/t/a.out

Breakpoint 1, main () at ./b.c:8
8               fp = fopen("/tmp/a.txt", "wt");
(gdb) 
```

```
Breakpoint 1, main () at ./b.c:8
8               fp = fopen("/tmp/a.txt", "wt");
(gdb) break __libc_open
Breakpoint 2 at 0x35a1817770: __libc_open. (2 locations)
(gdb) c
Continuing.

Breakpoint 2, open64 () at ../sysdeps/unix/syscall-template.S:82
82      T_PSEUDO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
(gdb) where
#0  open64 () at ../sysdeps/unix/syscall-template.S:82
#1  0x00000035a1c77359 in _IO_file_open (is32not64=<optimized out>, read_write=4, prot=438, posix_mod
e=<optimized out>,
    filename=<optimized out>, fp=0x601010) at fileops.c:240
#2  _IO_new_file_fopen (fp=fp@entry=0x601010, filename=filename@entry=0x400663 "/tmp/a.txt", mode=<op
timized out>,
    mode@entry=0x400660 "wt", is32not64=is32not64@entry=1) at fileops.c:345
#3  0x00000035a1c6bb56 in __fopen_internal (filename=0x400663 "/tmp/a.txt", mode=0x400660 "wt", is32=
1) at ../libio/iofopen.c:93
#4  0x0000000000400593 in main () at ./b.c:8
(gdb)
```

```
[hank@Maestro t]$ ps -ef|grep a.out
hank     16984  1555  0 21:55 pts/0    00:00:00 gdb ./a.out
hank     16986 16984  0 21:55 pts/0    00:00:00 /home/hank/t/a.out
hank     17006  1715  0 22:00 pts/1    00:00:00 grep --color=auto a.out
[hank@Maestro t]$
[hank@Maestro t]$ cat /proc/16986/maps
00400000-00401000 r-xp 00000000 fd:02 1831565                            /home/hank/t/a.out
00600000-00601000 rw-p 00000000 fd:02 1831565                            /home/hank/t/a.out
00601000-00622000 rw-p 00000000 00:00 0                                  [heap]
35a1800000-35a1820000 r-xp 00000000 fd:01 175898                         /usr/lib64/ld-2.15.so
35a1a1f000-35a1a20000 r--p 0001f000 fd:01 175898                         /usr/lib64/ld-2.15.so
35a1a20000-35a1a21000 rw-p 00020000 fd:01 175898                         /usr/lib64/ld-2.15.so
35a1a21000-35a1a22000 rw-p 00000000 00:00 0
35a1c00000-35a1dac000 r-xp 00000000 fd:01 175914                         /usr/lib64/libc-2.15.so
35a1dac000-35a1fac000 ---p 001ac000 fd:01 175914                         /usr/lib64/libc-2.15.so
35a1fac000-35a1fb0000 r--p 001ac000 fd:01 175914                         /usr/lib64/libc-2.15.so
35a1fb0000-35a1fb2000 rw-p 001b0000 fd:01 175914                         /usr/lib64/libc-2.15.so
35a1fb2000-35a1fb7000 rw-p 00000000 00:00 0
7ffff7fe1000-7ffff7fe4000 rw-p 00000000 00:00 0
7ffff7ffd000-7ffff7ffe000 rw-p 00000000 00:00 0
7ffff7ffe000-7ffff7fff000 r-xp 00000000 00:00 0                          [vdso]
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
[hank@Maestro t]$
```

System Calls

```
hank@Maestro:~/t
File  Edit  View  Search  Terminal  Help
timized out>,
    mode@entry=0x400660 "wt", is32not64=is32not64@entry=1) at fileops.c:345
#3  0x00000035a1c6bb56 in __fopen_internal (filename=0x400663 "/tmp/a.txt", mode=0x400660 "wt", is32=
1) at ../libio/iofopen.c:93
#4  0x0000000000400593 in main () at ./b.c:8
(gdb) disassemble __libc_open
Dump of assembler code for function open64:
=> 0x00000035a1ce46e0 <+0>:     cmpl   $0x0,0x2d1acd(%rip)        # 0x35a1fb61b4 <__libc_multiple_thr
eads>
   0x00000035a1ce46e7 <+7>:     jne    0x35a1ce46f9 <open64+25>
   0x00000035a1ce46e9 <+0>:     mov    $0x2,%eax
   0x00000035a1ce46ee <+5>:     syscall
   0x00000035a1ce46f0 <+7>:     cmp    $0xfffffffffffff001,%rax
   0x00000035a1ce46f6 <+13>:    jae    0x35a1ce4729 <open64+73>
   0x00000035a1ce46f8 <+15>:    retq
   0x00000035a1ce46f9 <+25>:    sub    $0x8,%rsp
   0x00000035a1ce46fd <+29>:    callq  0x35a1cff4a0 <__libc_enable_asynccancel>
   0x00000035a1ce4702 <+34>:    mov    %rax,(%rsp)
   0x00000035a1ce4706 <+38>:    mov    $0x2,%eax
   0x00000035a1ce470b <+43>:    syscall
   0x00000035a1ce470d <+45>:    mov    (%rsp),%rdi
   0x00000035a1ce4711 <+49>:    mov    %rax,%rdx
   0x00000035a1ce4714 <+52>:    callq  0x35a1cff500 <__libc_disable_asynccancel>
   0x00000035a1ce4719 <+57>:    mov    %rdx,%rax
   0x00000035a1ce471c <+60>:    add    $0x8,%rsp
   0x00000035a1ce4720 <+64>:    cmp    $0xfffffffffffff001,%rax
   0x00000035a1ce4726 <+70>:    jae    0x35a1ce4729 <open64+73>
   0x00000035a1ce4728 <+72>:    retq
   0x00000035a1ce4729 <+73>:    mov    0x2cb700(%rip),%rcx        # 0x35a1fafe30
   0x00000035a1ce4730 <+80>:    xor    %edx,%edx
   0x00000035a1ce4732 <+82>:    sub    %rax,%rdx
   0x00000035a1ce4735 <+85>:    mov    %edx,%fs:(%rcx)
   0x00000035a1ce4738 <+88>:    or     $0xffffffffffffffff,%rax
   0x00000035a1ce473c <+92>:    jmp    0x35a1ce4728 <open64+72>
End of assembler dump.
(gdb)
```

Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

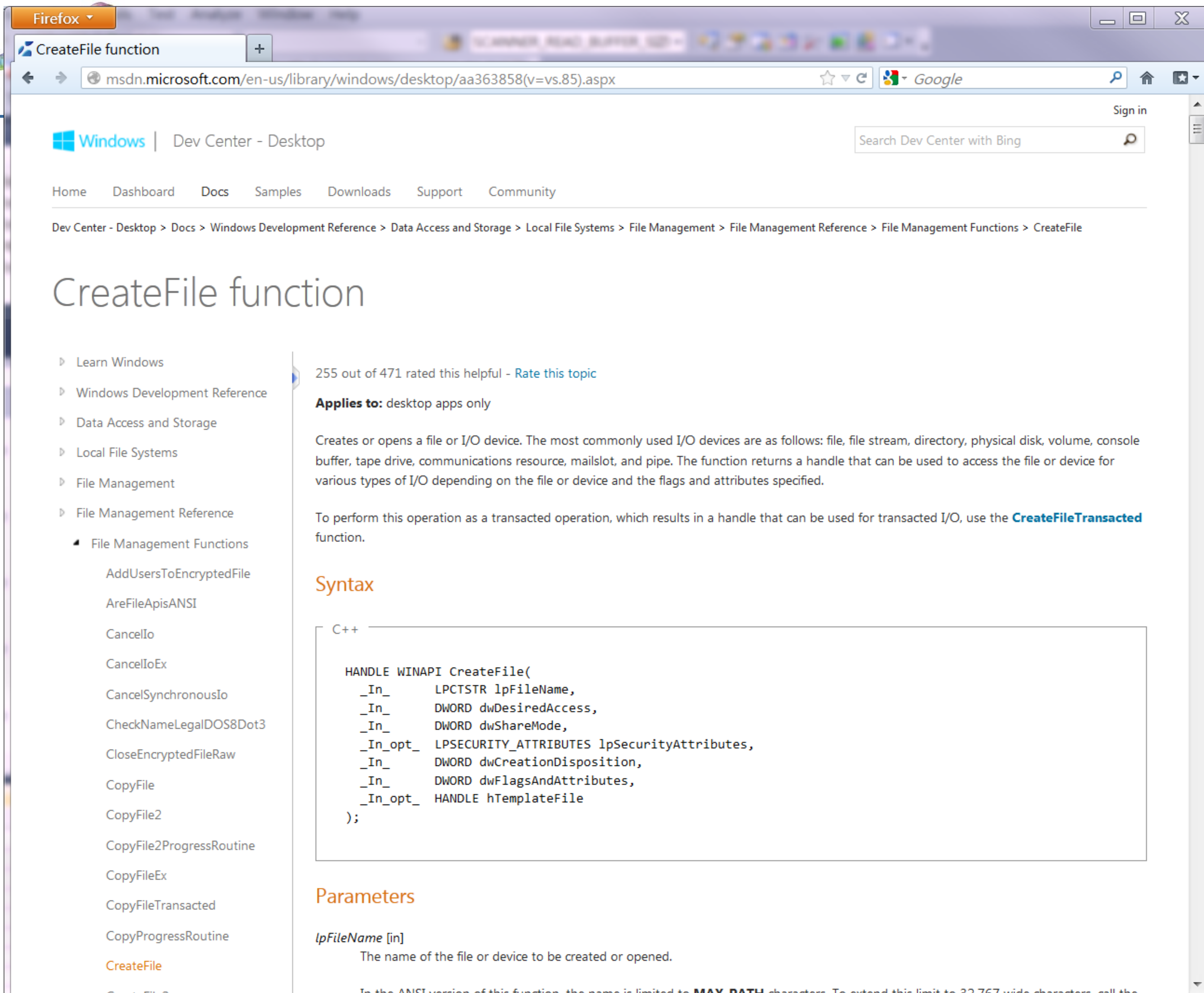
```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value	function name	parameters
`ssize_t`	`read`	`(int fd, void *buf, size_t count)`

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.



CreateFile function

255 out of 471 rated this helpful - Rate this topic

Applies to: desktop apps only

Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified.

To perform this operation as a transacted operation, which results in a handle that can be used for transacted I/O, use the **CreateFileTransacted** function.

Syntax

C++

```
HANDLE WINAPI CreateFile(
  _In_      LPCTSTR lpFileName,
  _In_      DWORD dwDesiredAccess,
  _In_      DWORD dwShareMode,
  _In_opt_  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  _In_      DWORD dwCreationDisposition,
  _In_      DWORD dwFlagsAndAttributes,
  _In_opt_  HANDLE hTemplateFile
);
```

Parameters

lpFileName [in]

The name of the file or device to be created or opened.

In the ANSI version of this function, the name is limited to **MAX_PATH** characters. To extend this limit to 32,767 wide characters, call the

System Call Implementation

Typically, a number associated with each system call

- **System-call interface** maintains a table indexed according to these numbers

The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values

The caller need know nothing about how the system call is implemented

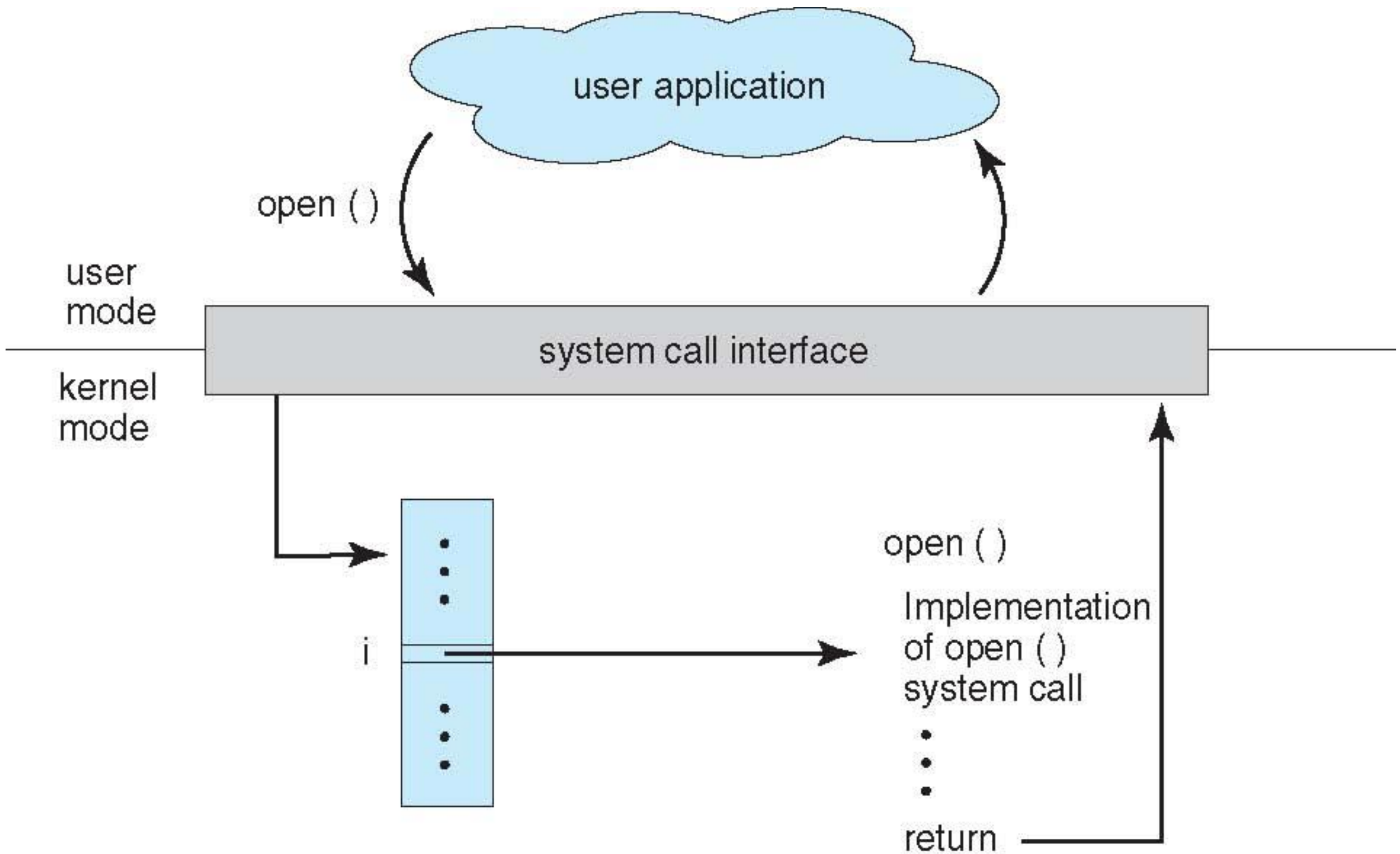
- Just needs to obey API and understand what OS will do as a result call
- Most details of OS interface hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

(/usr/include/asm/unistd_64.h)

linux1:/usr/include/asm

```
#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H

#ifndef __SYSCALL
#define __SYSCALL(a, b)
#endif

/*
 * This file contains the system call numbers.
 *
 * Note: holes are not allowed.
 */

/* at least 8 syscall per cacheline */
#define __NR_read                               0
__SYSCALL(__NR_read, sys_read)
#define __NR_write                              1
__SYSCALL(__NR_write, sys_write)
#define __NR_open                               2
__SYSCALL(__NR_open, sys_open)
#define __NR_close                              3
__SYSCALL(__NR_close, sys_close)
#define __NR_stat                               4
--More--(2%)
```

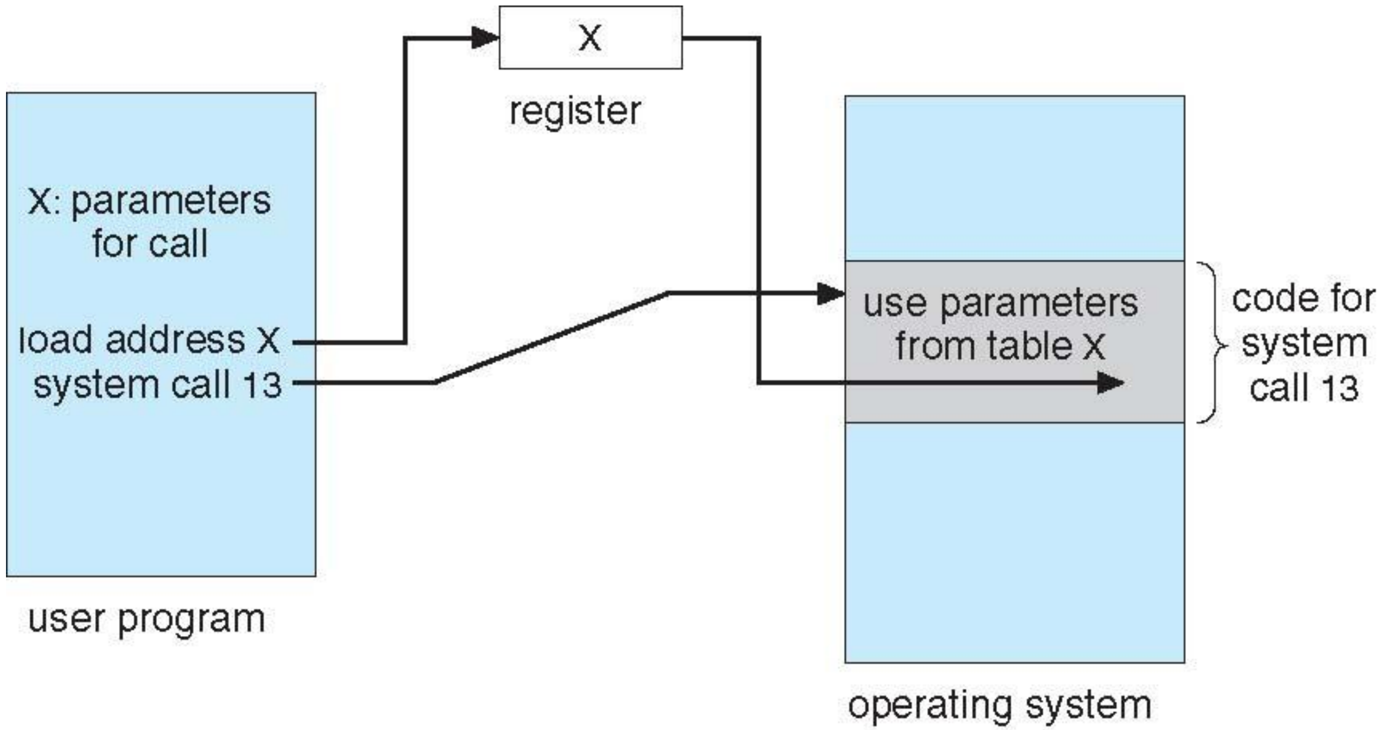
API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Parameter Passing

http://www.x86-64.org/documentation/abi-0.99.pdf

A.2.1 Calling Conventions

The Linux AMD64 kernel uses internally the same calling conventions as user-level applications (see section 3.2.3 for details). User-level applications that like to call system calls should use the functions from the C library. The interface between the C library and the Linux kernel is the same as for the user-level applications with the following differences:

1. User-level applications use as integer registers for passing the sequence `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8` and `%r9`. The kernel interface uses `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8` and `%r9`.
2. A system-call is done via the `syscall` instruction. The kernel destroys registers `%rcx` and `%r11`.
3. The number of the syscall has to be passed in register `%rax`.
4. System-calls are limited to six arguments, no argument is passed directly on the stack.
5. Returning from the `syscall`, register `%rax` contains the result of the system-call. A value in the range between -4095 and -1 indicates an error, it is `-errno`.
6. Only values of class INTEGER or class MEMORY are passed to the kernel.

lxr.linux.no/linux+v3.6/fs/open.c

```c
        return ERR_PTR(-ENOTDIR);
        return do_file_open_root(dentry, mnt, filename, &op, lookup);
}
EXPORT_SYMBOL(file_open_root);

long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
        struct open_flags op;
        int lookup = build_open_flags(flags, mode, &op);
        char *tmp = getname(filename);
        int fd = PTR_ERR(tmp);

        if (!IS_ERR(tmp)) {
                fd = get_unused_fd_flags(flags);
                if (fd >= 0) {
                        struct file *f = do_filp_open(dfd, tmp, &op, lookup);
                        if (IS_ERR(f)) {
                                put_unused_fd(fd);
                                fd = PTR_ERR(f);
                        } else {
                                fsnotify_open(f);
                                fd_install(fd, f);
                        }
                }
                putname(tmp);
        }
        return fd;
}

SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
        long ret;

        if (force_o_largefile())
                flags |= O_LARGEFILE;

        ret = do_sys_open(AT_FDCWD, filename, flags, mode);
        /* avoid REGPARM breakage on x86: */
        asmlinkage_protect(3, ret, filename, flags, mode);
        return ret;
}
```

Types of System Calls

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (Cont.)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

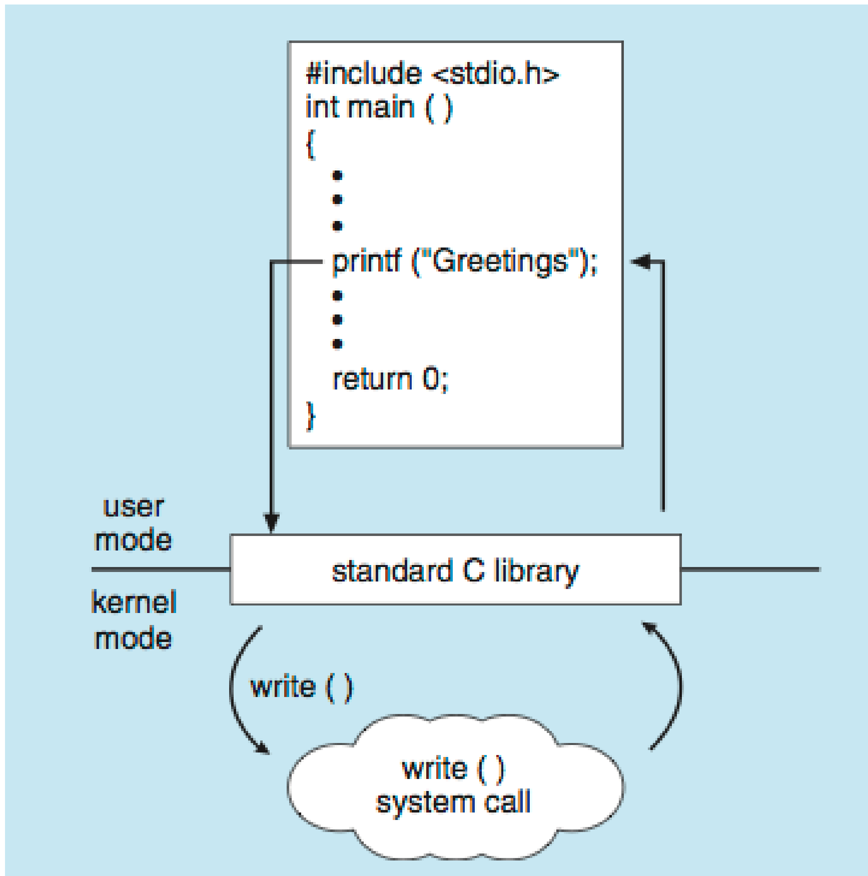
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

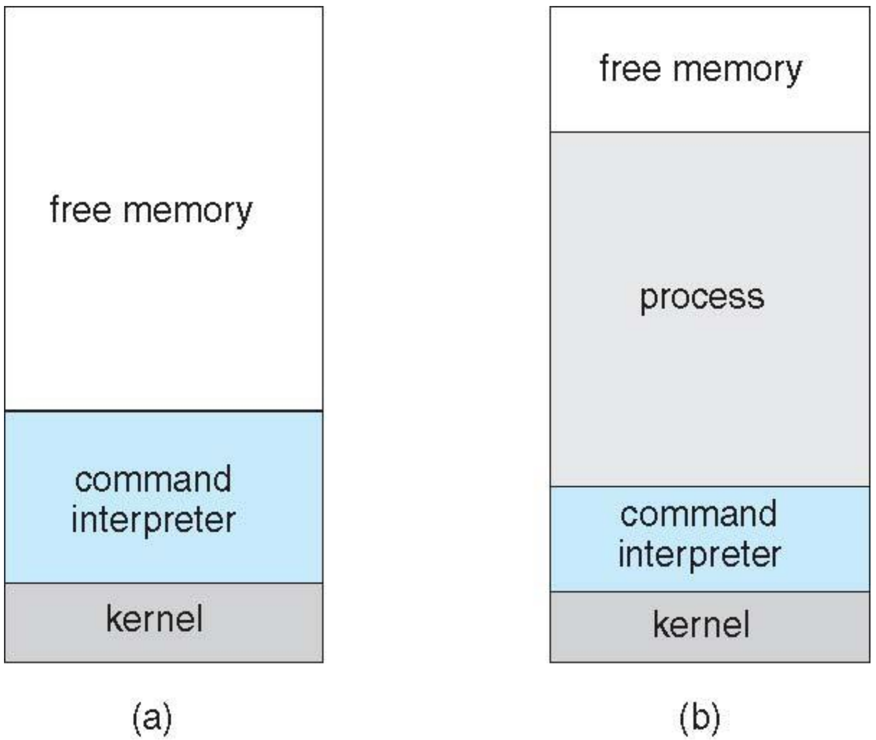
Standard C Library Example

C program invoking printf() library call, which calls write() system call



Example: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup running a program

Example: FreeBSD

Unix variant

Multitasking

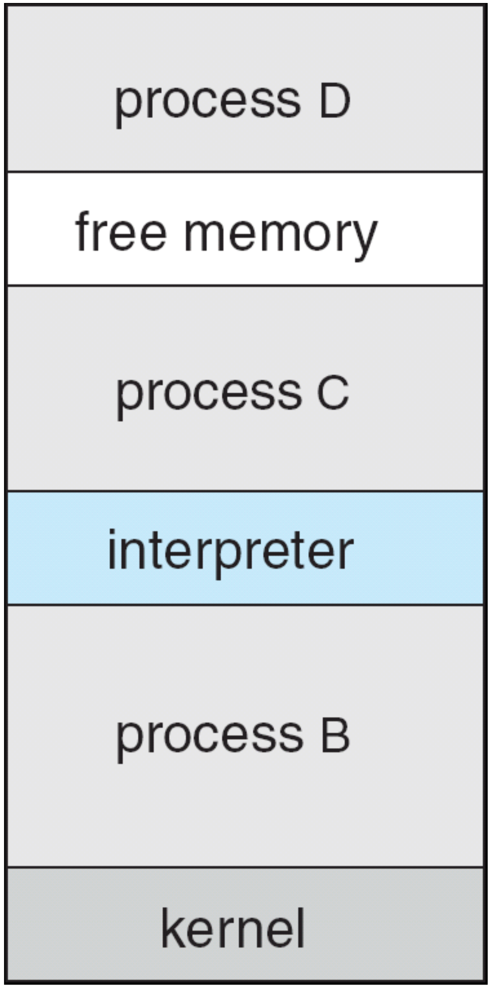
User login -> invoke user's choice of shell

Shell executes fork() system call to create process

- Executes exec() to load program into process
- Shell waits for process to terminate or continues with user commands

Process exits with:

- code = 0 – no error
- code > 0 – error code



System Programs

System programs provide a convenient environment for program development and execution. They can be divided into:

- File manipulation
- Status information sometimes stored in a File modification
- Programming language support
- Program loading and execution
- Communications
- Background services
- Application programs

Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

Provide a convenient environment for program development and execution

- Some of them are simply user interfaces to system calls; others are considerably more complex

File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

Status information

- Some ask the system for info - date, time, amount of available memory, disk space, number of users
- Others provide detailed performance, logging, and debugging information
- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a **registry** - used to store and retrieve configuration information

System Programs (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs (Cont.)

Background Services

- Launch at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as **services**, **subsystems**, **daemons**

Application programs

- Don’t pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke

Operating System Design and Implementation

Design and Implementation of OS not "solvable", but some approaches have proven successful

Internal structure of different Operating Systems can vary widely

Start the design by defining goals and specifications

Affected by choice of hardware, type of system

User goals and **System** goals

- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

Important principle to separate

Policy: ***What*** will be done?
Mechanism: ***How*** to do it?

Mechanisms determine how to do something, policies decide what will be done

The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example – timer)

Specifying and designing an OS is highly creative task of **software engineering**

Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware

Operating System Structure

General-purpose OS is very large program

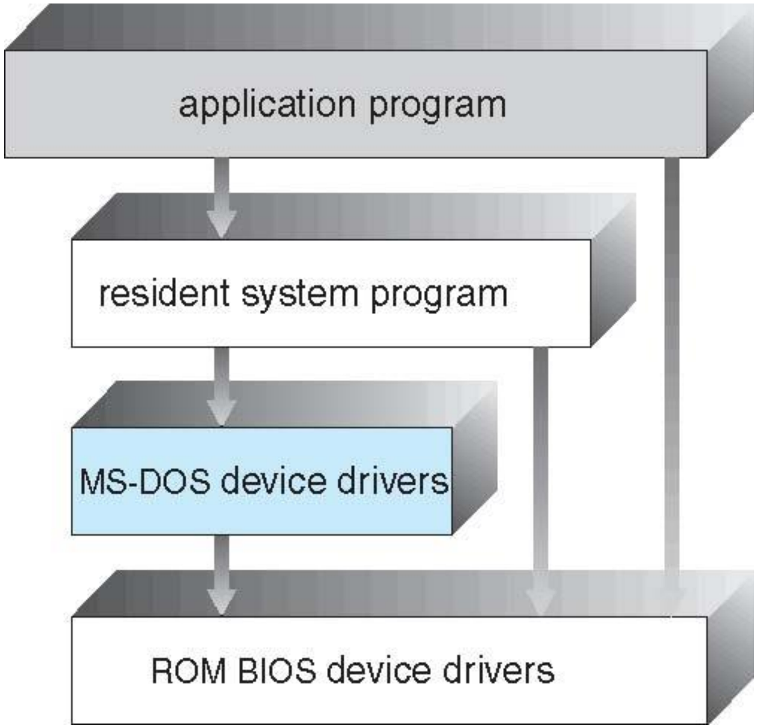
Various ways to structure ones

- Simple structure – MS-DOS
- More complex -- UNIX
- Layered – an abstrcation
- Microkernel -Mach

Simple Structure -- MS-DOS

MS-DOS – written to provide the most functionality in the least space

- Not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



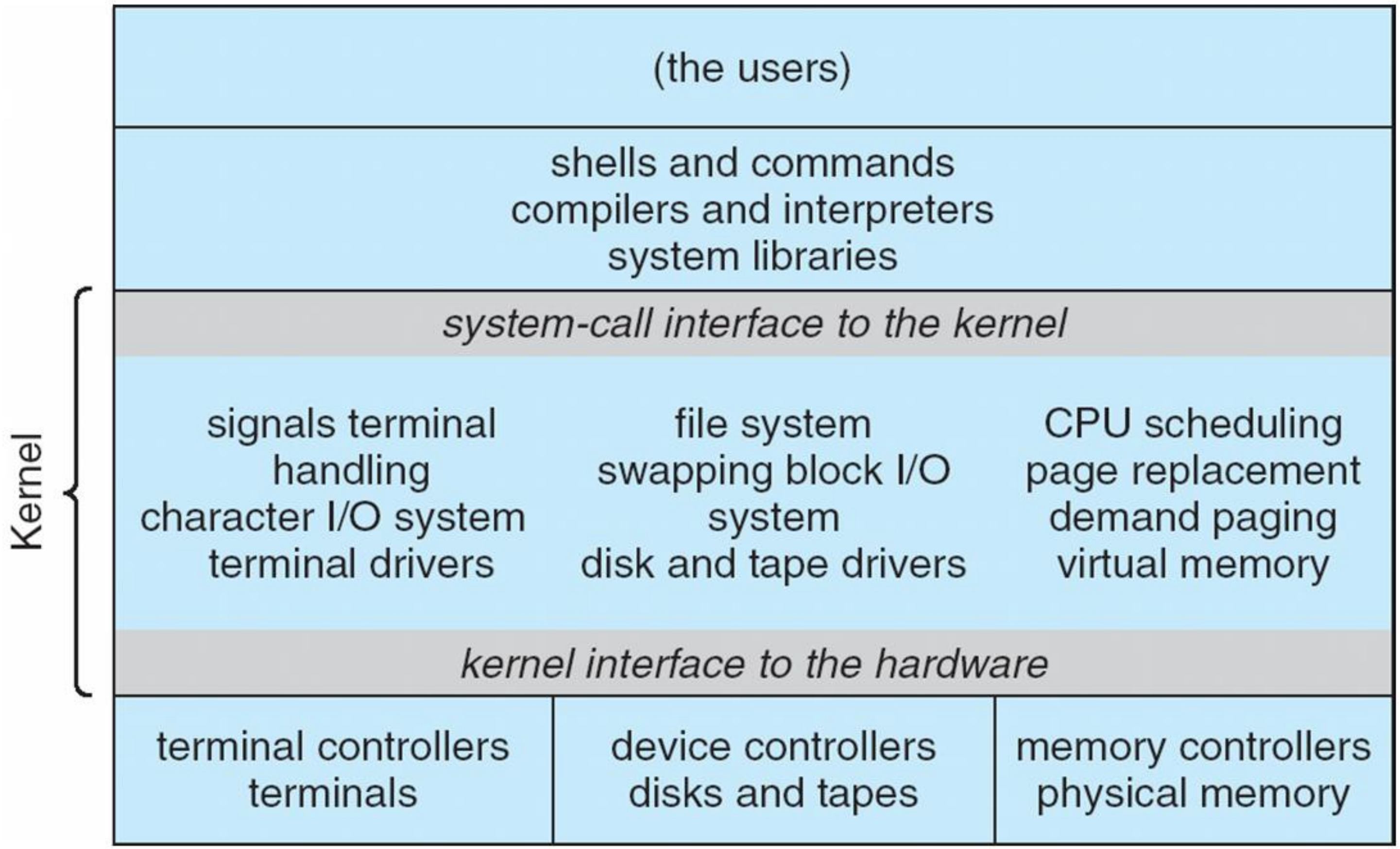
Non Simple Structure -- UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- Systems programs
- The kernel
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Traditional UNIX System Structure

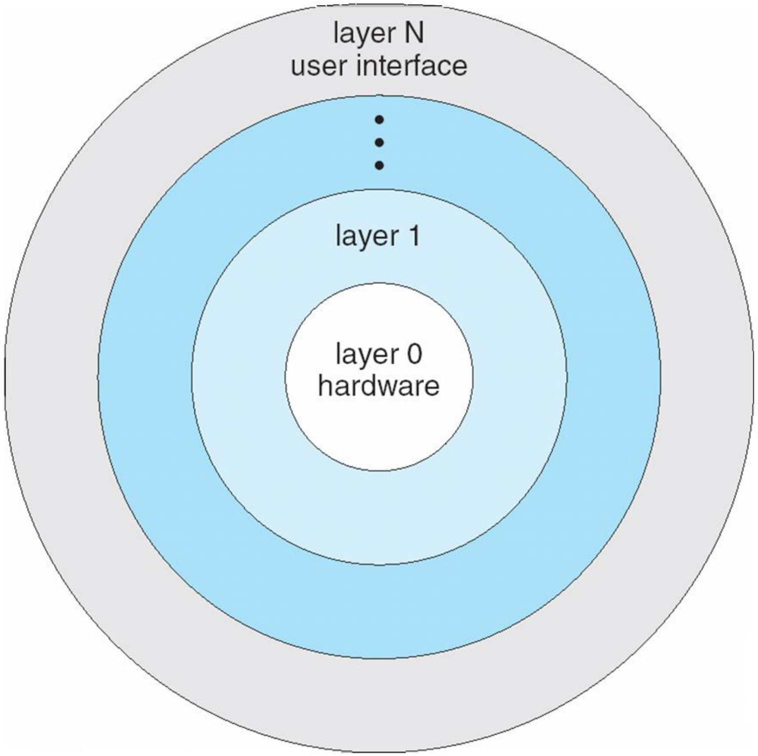
Beyond simple but not fully layered



Layered Approach

The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

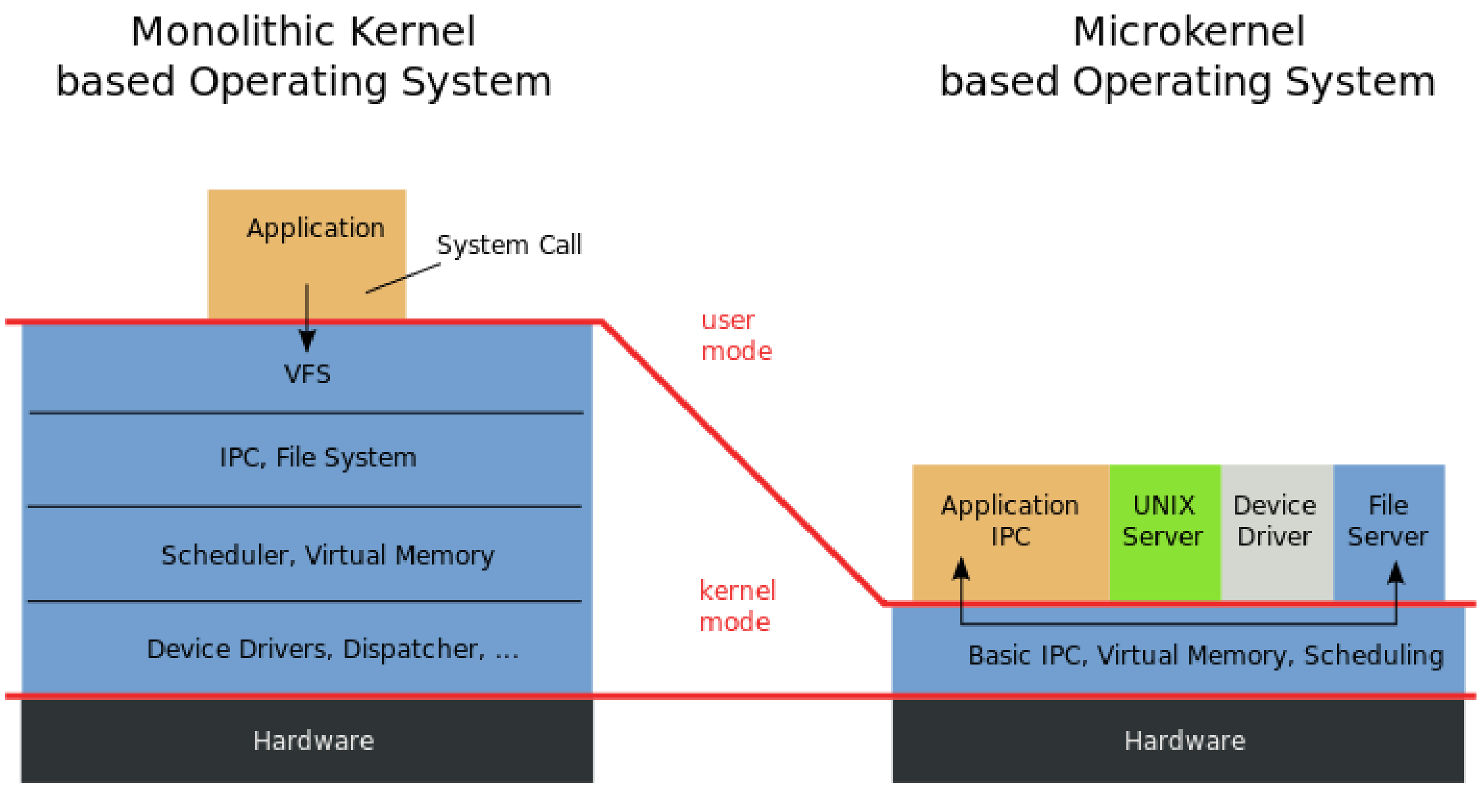
With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Microkernel System Structure



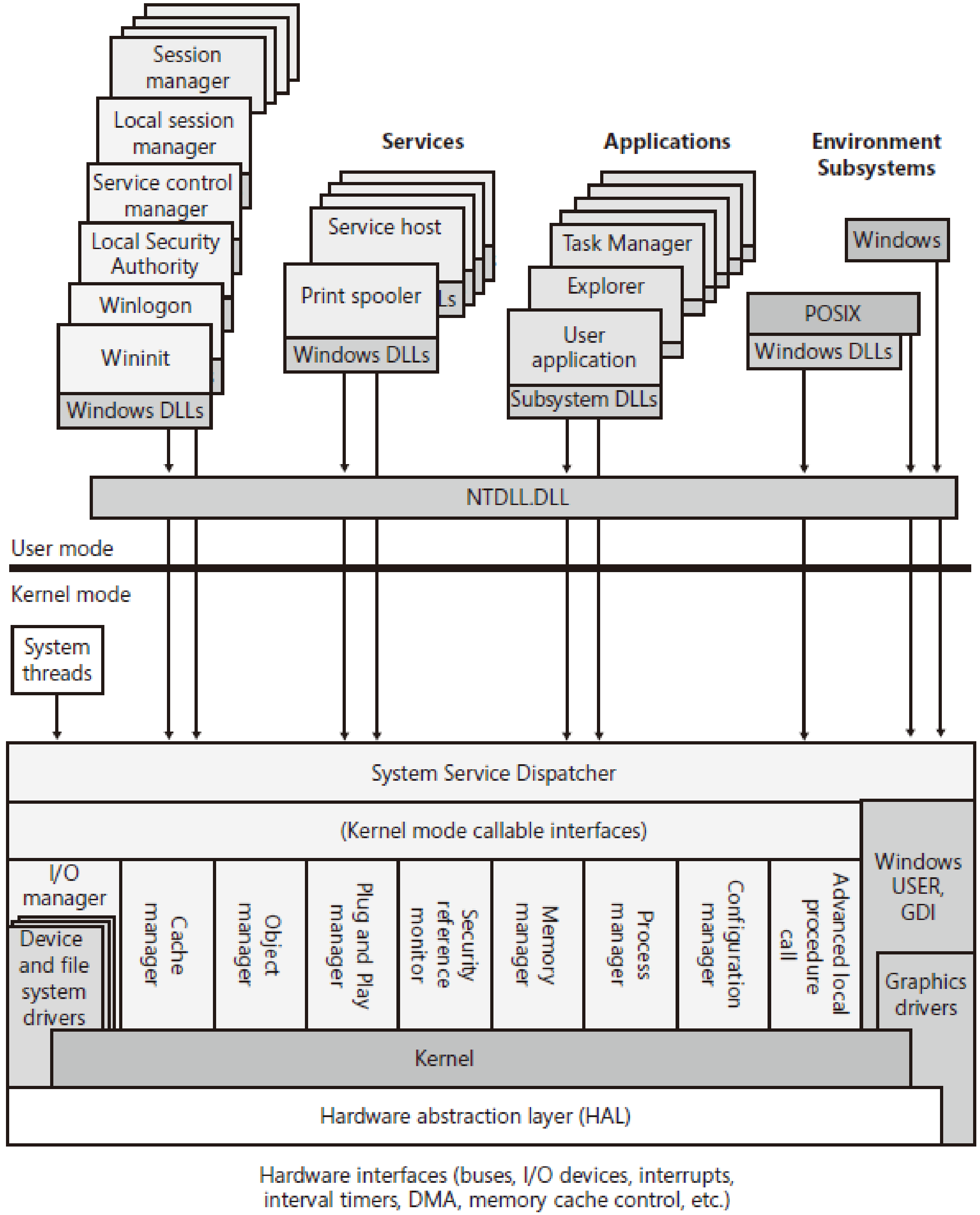
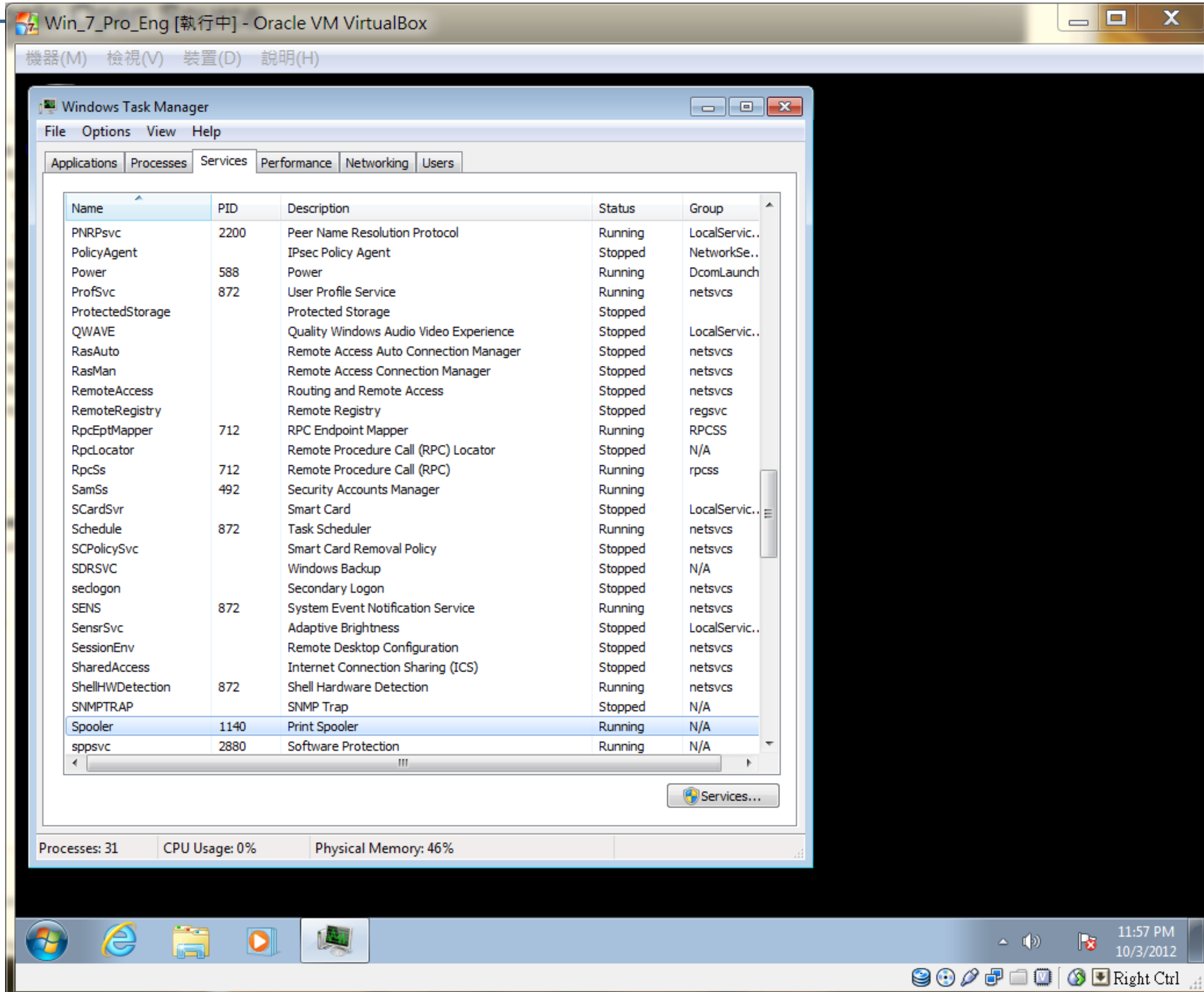
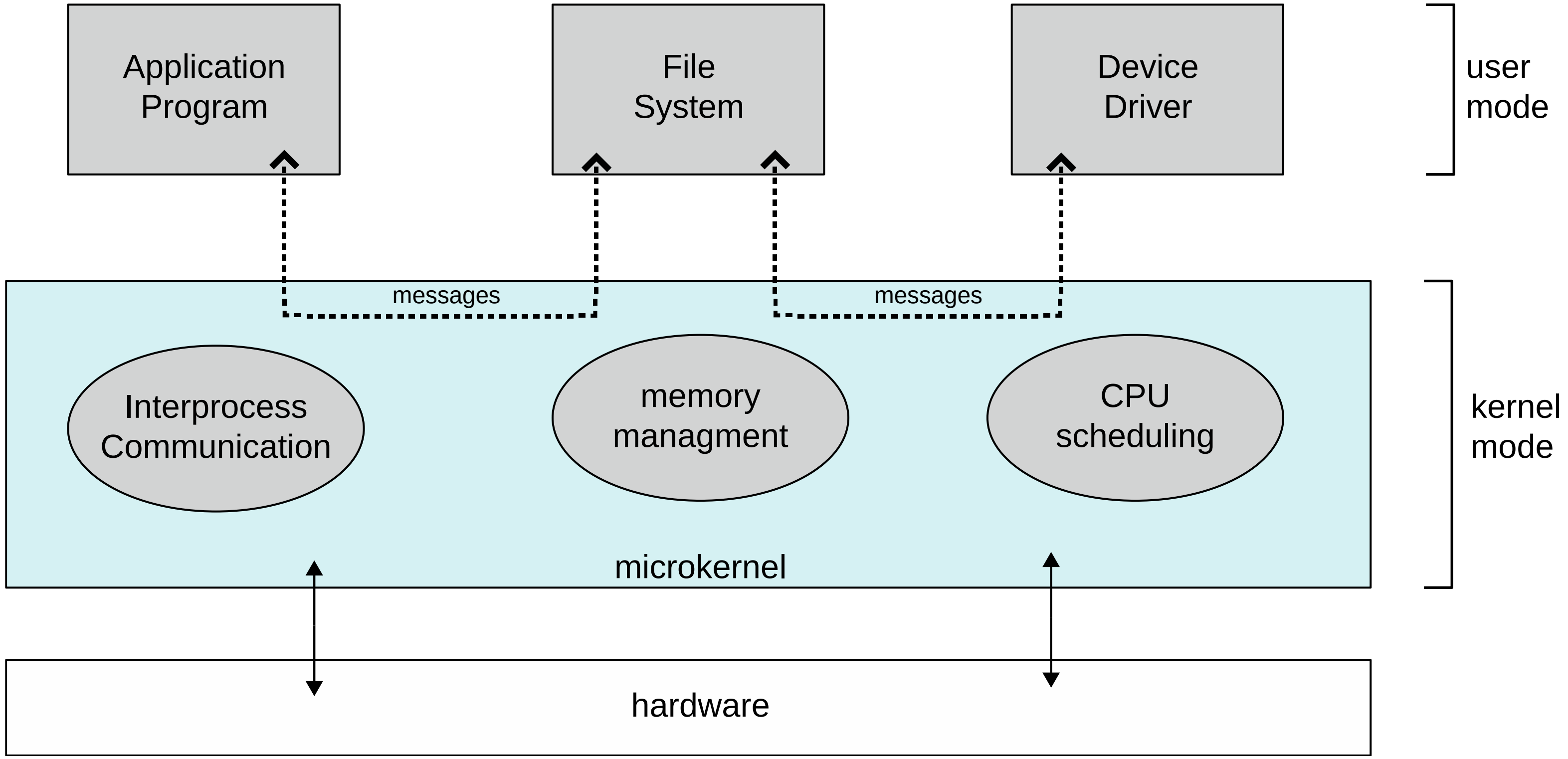


FIGURE 2-3 Windows architecture

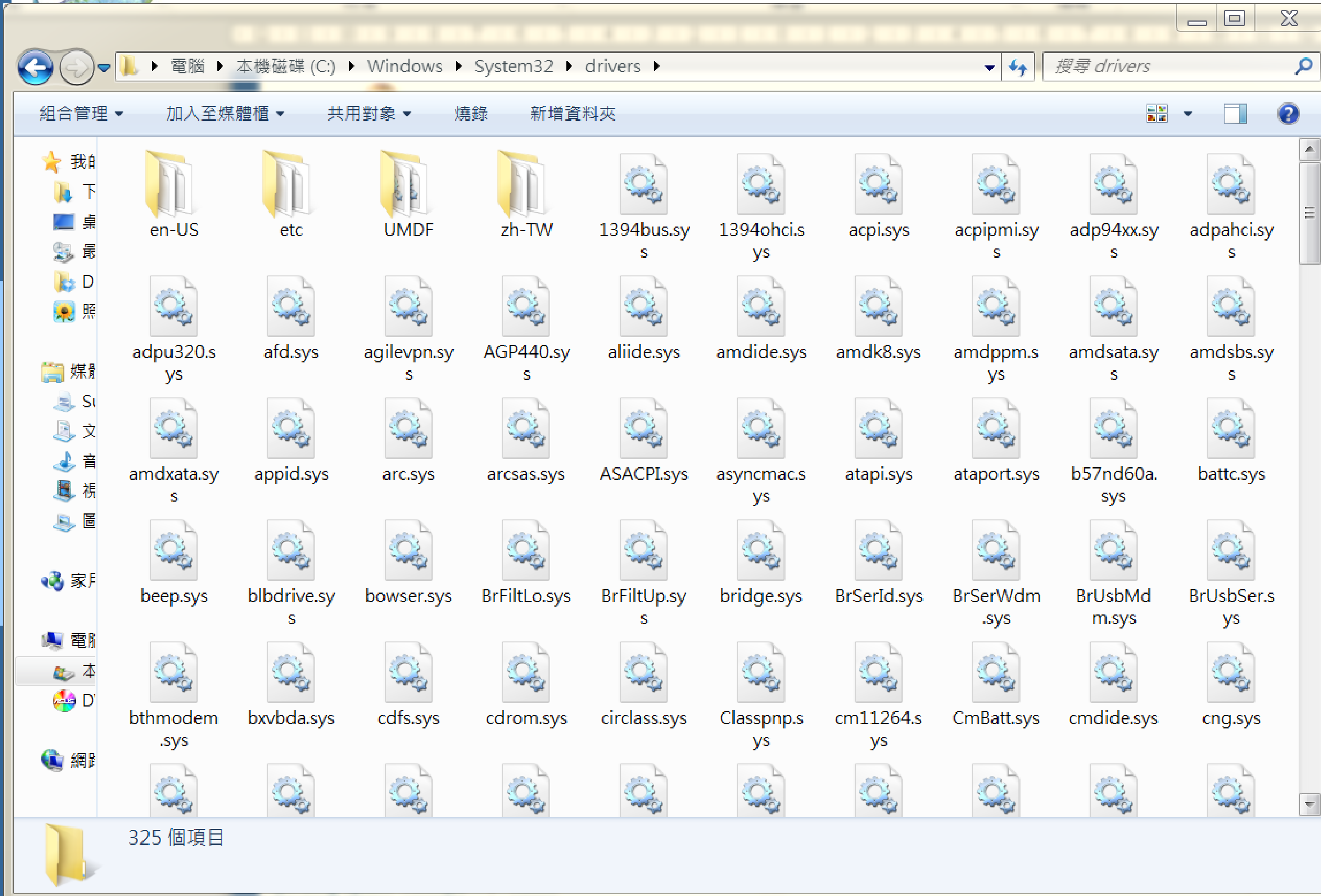

Win_7_Pro_Eng [執行中] - Oracle VM VirtualBox
機器(M) 檢視(V) 裝置(D) 說明(H)
Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
Name PID Description Status Group
PNRPsvc 2200 Peer Name Resolution Protocol Running LocalServic...
PolicyAgent IPsec Policy Agent Stopped NetworkSe...
Power 588 Power Running DcomLaunch
ProfSvc 872 User Profile Service Running netsvcs
ProtectedStorage Protected Storage Stopped
QWAVE Quality Windows Audio Video Experience Stopped LocalServic...
RasAuto Remote Access Auto Connection Manager Stopped netsvcs
RasMan Remote Access Connection Manager Stopped netsvcs
RemoteAccess Routing and Remote Access Stopped netsvcs
RemoteRegistry Remote Registry Stopped regsvc
RpcEptMapper 712 RPC Endpoint Mapper Running RPCSS
RpcLocator Remote Procedure Call (RPC) Locator Stopped N/A
RpcSs 712 Remote Procedure Call (RPC) Running rpcss
SamSs 492 Security Accounts Manager Running
SCardSvr Smart Card Stopped LocalServic...
Schedule 872 Task Scheduler Running netsvcs
SCPolicySvc Smart Card Removal Policy Stopped netsvcs
SDRSVC Windows Backup Stopped N/A
seclogon Secondary Logon Stopped netsvcs
SENS 872 System Event Notification Service Running netsvcs
SensrSvc Adaptive Brightness Stopped LocalServic...
SessionEnv Remote Desktop Configuration Stopped netsvcs
SharedAccess Internet Connection Sharing (ICS) Stopped netsvcs
ShellHWDetection 872 Shell Hardware Detection Running netsvcs
SNMPTRAP SNMP Trap Stopped N/A
Spooler 1140 Print Spooler Running N/A
sppsvc 2880 Software Protection Running N/A
Services...
Processes: 31 CPU Usage: 0% Physical Memory: 46%
11:57 PM
10/3/2012
Right Ctrl

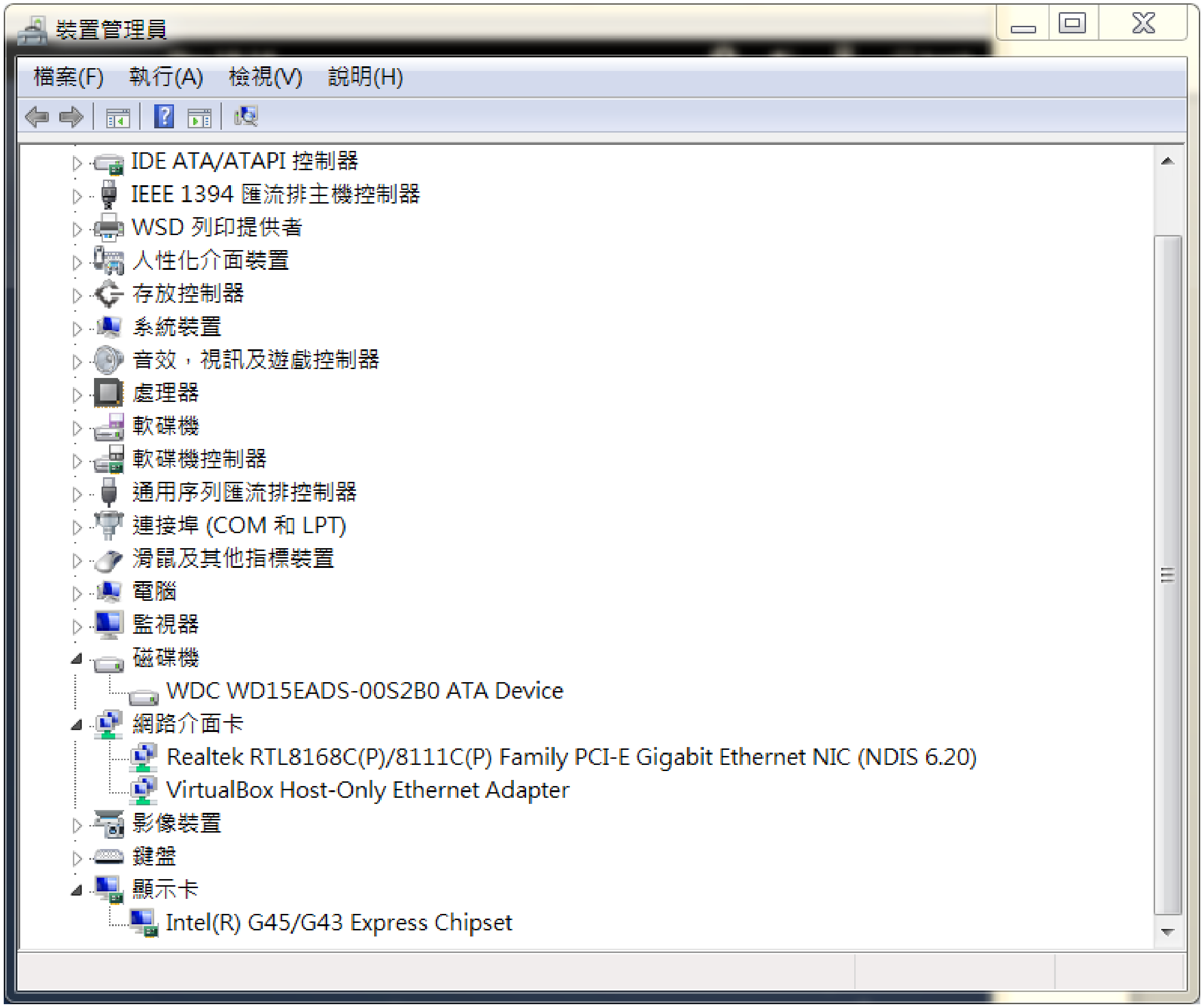
Microkernel System Structure



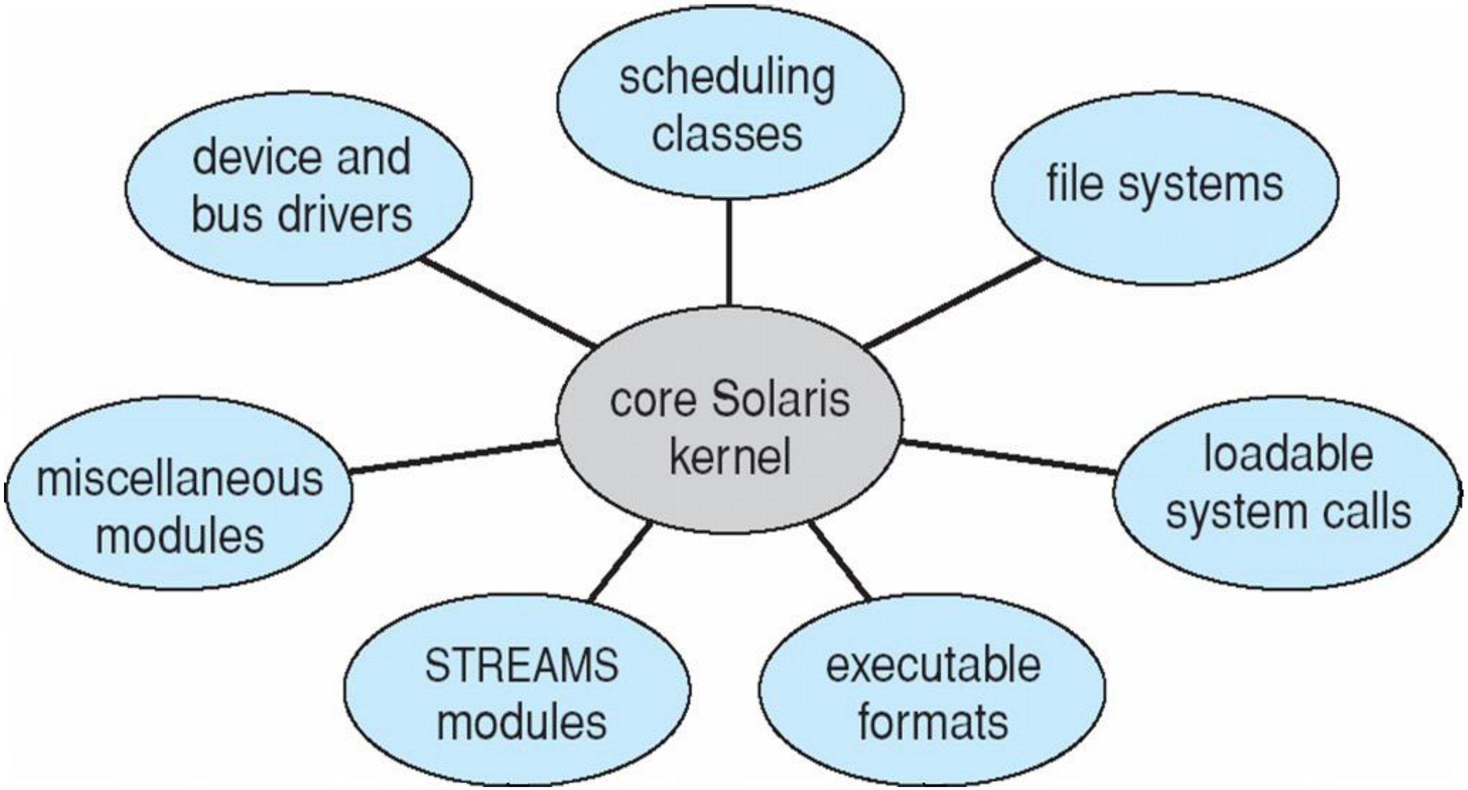
Modules

- Many modern operating systems implement **loadable kernel modules**
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc


電腦 本機磁碟 (C:) Windows System32 drivers
搜尋 drivers
組合管理
加入至媒體櫃
共用對象
燒錄
新增資料夾
en-US
etc
UMDF
zh-TW
1394bus.sys
1394ohci.sys
acpi.sys
acpipmi.sys
adp94xx.sys
adpahci.sys
adpu320.sys
afd.sys
agilevpn.sys
AGP440.sys
aliide.sys
amdide.sys
amdk8.sys
amdppm.sys
amdsata.sys
amdsbs.sys
amdxata.sys
appid.sys
arc.sys
arcsas.sys
ASACPI.sys
asyncmac.sys
atapi.sys
ataport.sys
b57nd60a.sys
battc.sys
beep.sys
blbdrive.sys
bowser.sys
BrFiltLo.sys
BrFiltUp.sys
bridge.sys
BrSerId.sys
BrSerWdm.sys
BrUsbMdm.sys
BrUsbSer.sys
bthmodem.sys
bxvbda.sys
cdfs.sys
cdrom.sys
circlass.sys
Classpnp.sys
cm11264.sys
CmBatt.sys
cmdide.sys
cng.sys
325 個項目


裝置管理員
檔案(F)
執行(A)
檢視(V)
說明(H)
IDE ATA/ATAPI 控制器
IEEE 1394 匯流排主機控制器
WSD 列印提供者
人性化介面裝置
存放控制器
系統裝置
音效，視訊及遊戲控制器
處理器
軟碟機
軟碟機控制器
通用序列匯流排控制器
連接埠 (COM 和 LPT)
滑鼠及其他指標裝置
電腦
監視器
磁碟機
WDC WD15EADS-00S2B0 ATA Device
網路介面卡
Realtek RTL8168C(P)/8111C(P) Family PCI-E Gigabit Ethernet NIC (NDIS 6.20)
VirtualBox Host-Only Ethernet Adapter
影像裝置
鍵盤
顯示卡
Intel(R) G45/G43 Express Chipset

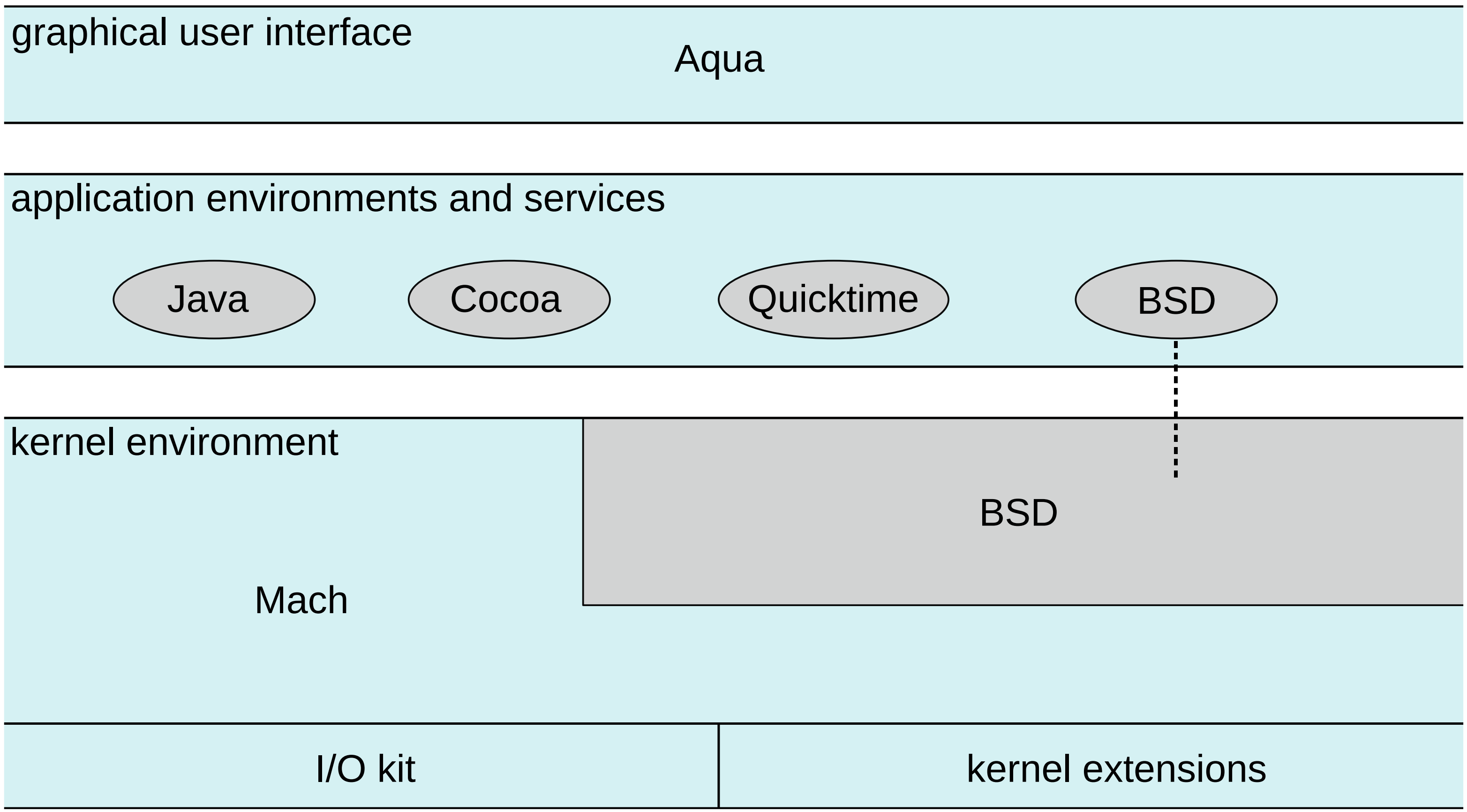
Solaris Modular Approach



Hybrid Systems

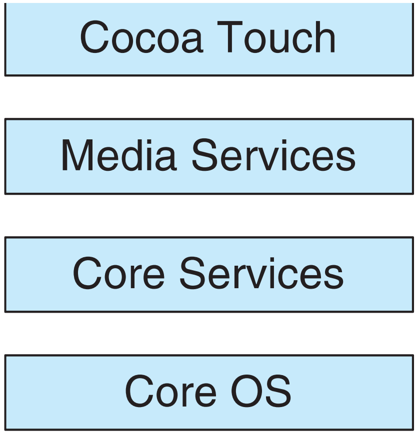
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

Mac OS X Structure



iOS

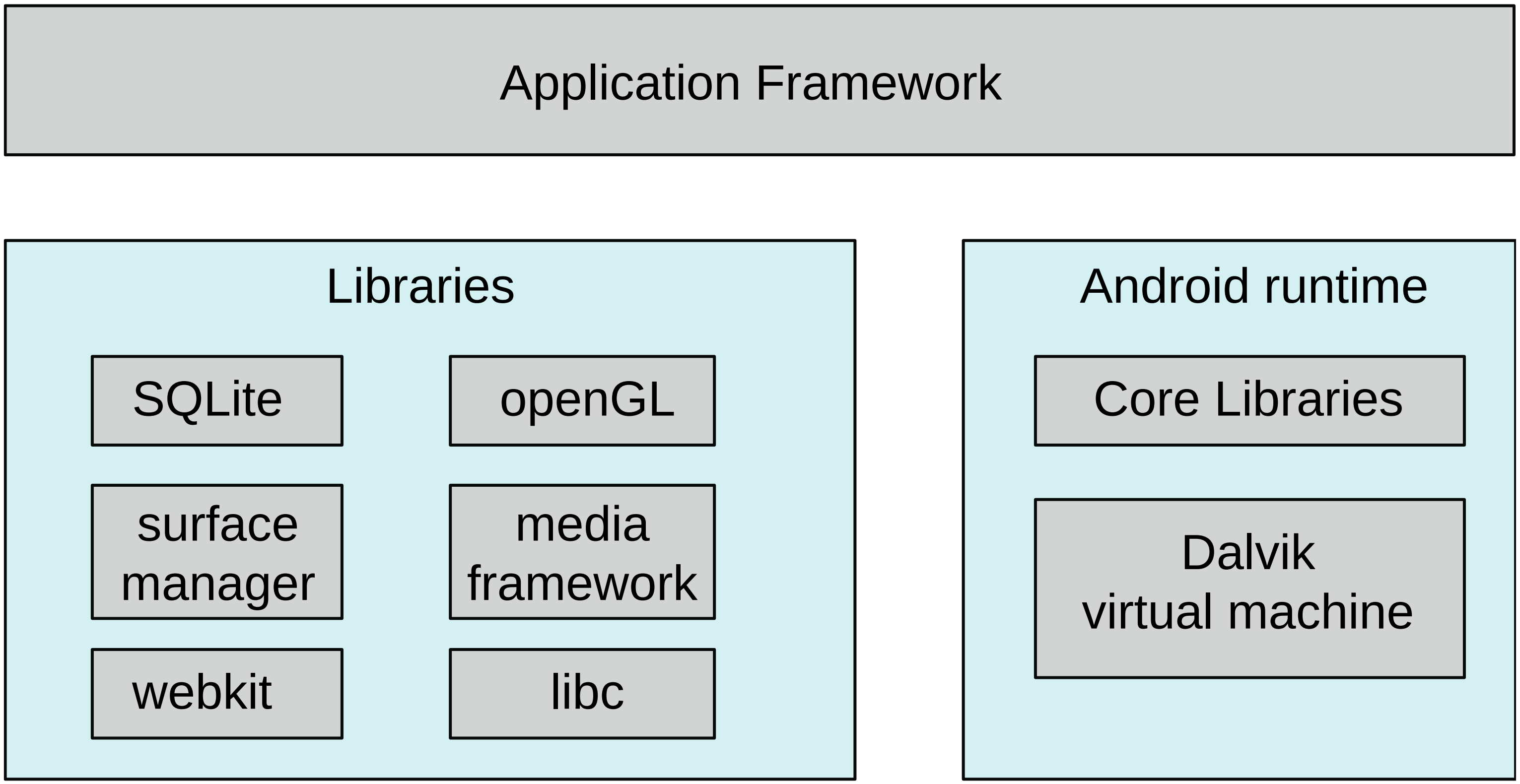
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



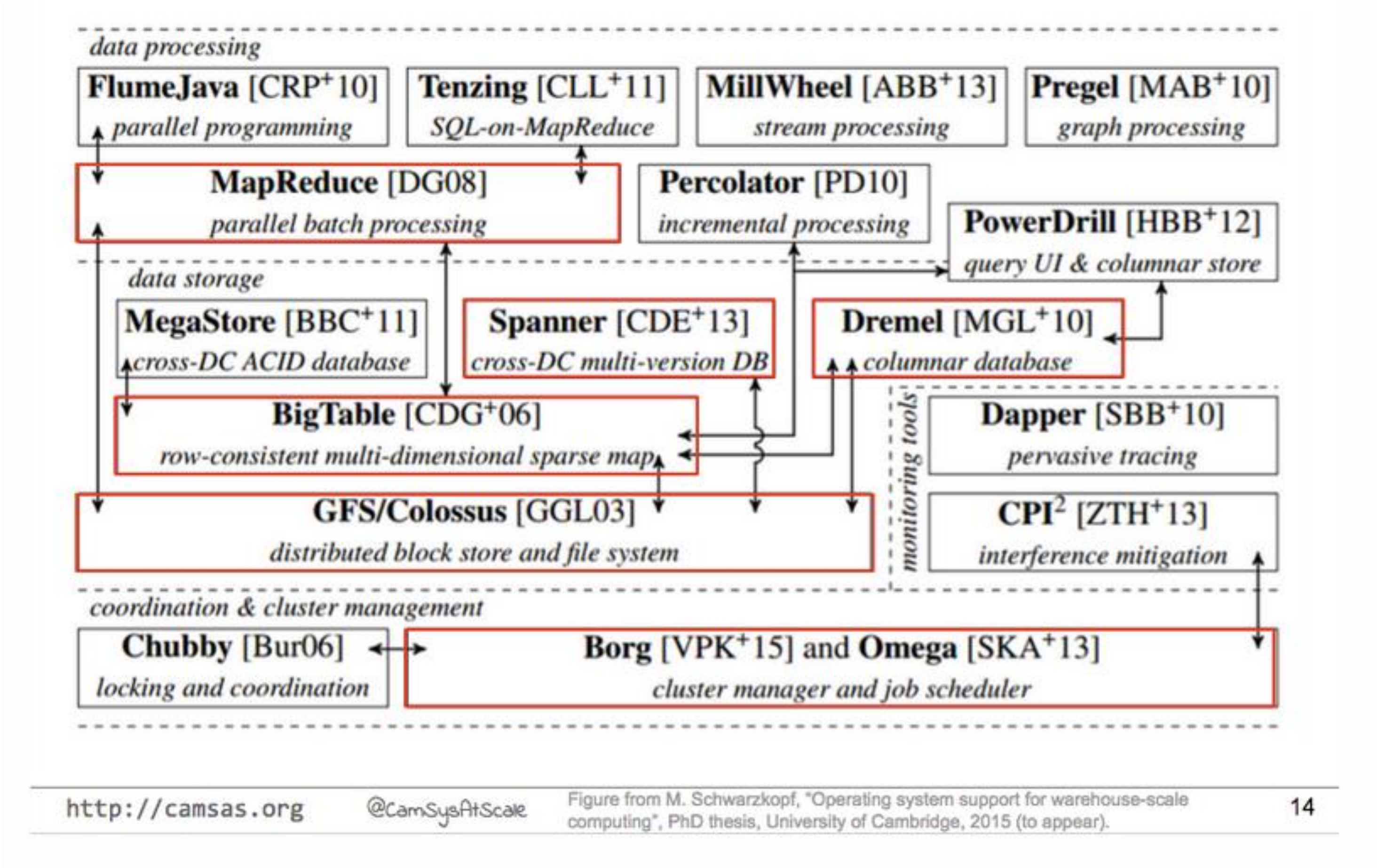
Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

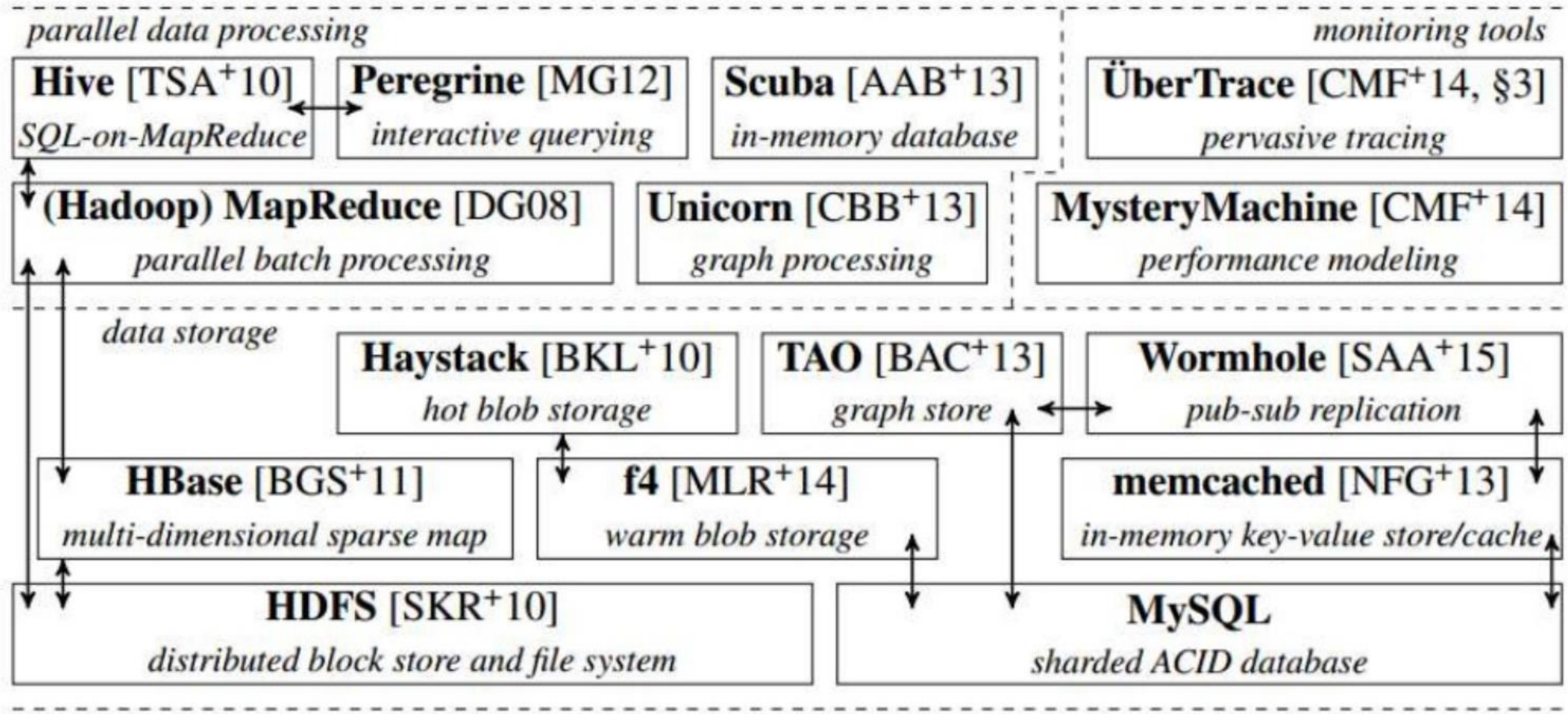
Android Architecture



The Google Stack



The Facebook Stack



(From: http://malteschwarzkopf.de/research/assets/facebook-stack.pdf)

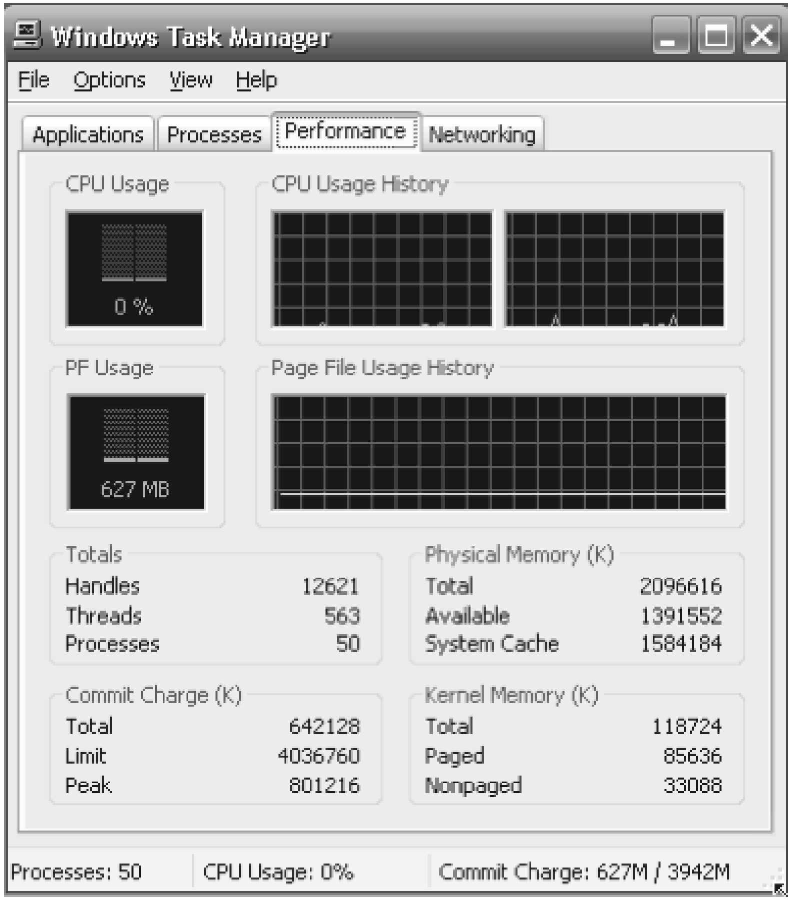
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



DTrace

DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems

Probes fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes

Example of following XEventsQueued system call move from libc library to kernel and back

```
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                          U
  0   -> _XEventsQueued                       U
  0     -> _X11TransBytesReadable             U
  0     <- _X11TransBytesReadable             U
  0     -> _X11TransSocketBytesReadable U
  0     <- _X11TransSocketBytesreadable U
  0     -> ioctl                              U
  0       -> ioctl                            K
  0         -> getf                           K
  0           -> set_active_fd                K
  0           <- set_active_fd                K
  0         <- getf                           K
  0         -> get_udatamodel                 K
  0         <- get_udatamodel                 K
...
  0         -> releasef                       K
  0           -> clear_active_fd              K
  0           <- clear_active_fd              K
  0           -> cv_broadcast                 K
  0           <- cv_broadcast                 K
  0         <- releasef                       K
  0       <- ioctl                            K
  0     <- ioctl                              U
  0   <- _XEventsQueued                       U
  0 <- XEventsQueued                          U
```

Dtrace (Cont.)

DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
   gnome-settings-d         142354
   gnome-vfs-daemon         158243
   dsdm                     189804
   wnck-applet              200030
   gnome-panel              277864
   clock-applet             374916
   mapping-daemon           385475
   xscreensaver             514177
   metacity                 539281
   Xorg                    2579646
   gnome-terminal          5007269
   mixer_applet2           7388447
   java                   10769137
```

Figure 2.21 Output of the D code.

Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- **SYSGEN** program obtains information concerning the specific configuration of the hardware system
  - Used to build system-specific compiled kernel or system-tuned
  - Can general more efficient code than one general kernel

System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

End of Chapter 2