Theme 2

Operating systems

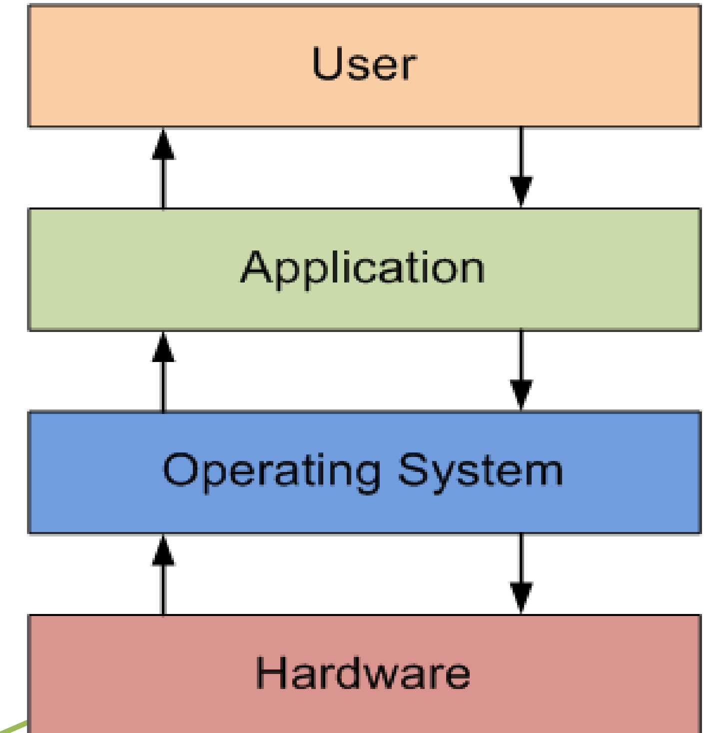
Subjects:

-Basic concepts
-User interface
-High level structure
-System primitives
-Kernel architecture

Duration - 4 ac.h.

Operating systems basic concepts

Operating System (OS) is an interface between hardware and user which is responsible for the management and coordination of activities and the sharing of the resources of the computer that acts as a host for computing applications run on the machine.



Linux

Google Chrome OS

Mac

MAC OS

FreeBSD

Windows Seven

Microsoft Windows

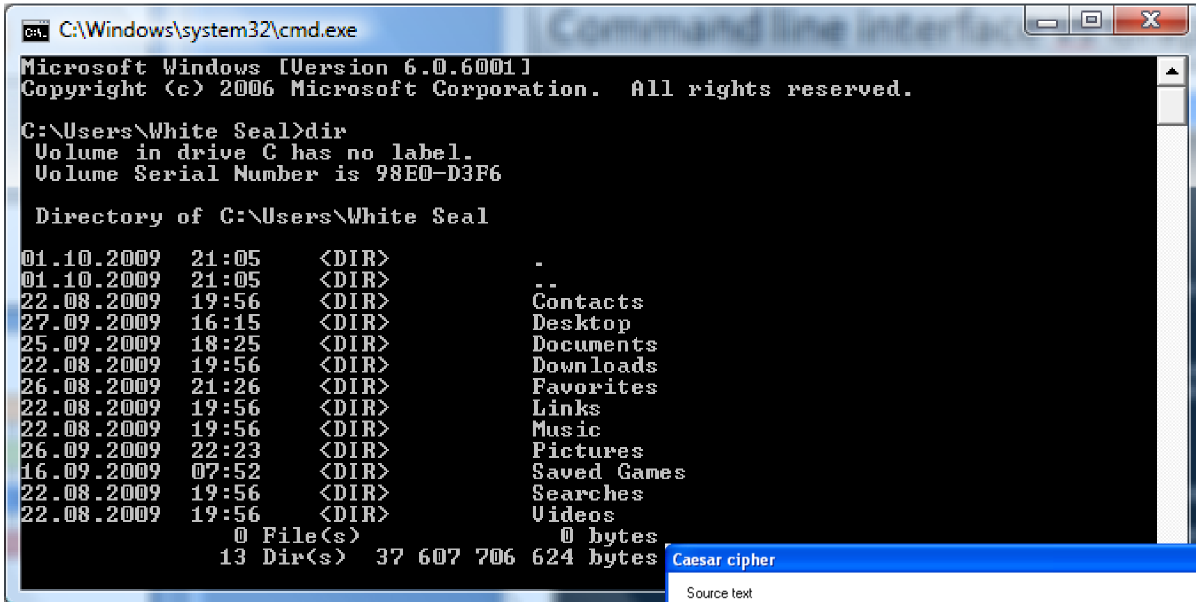
solaris

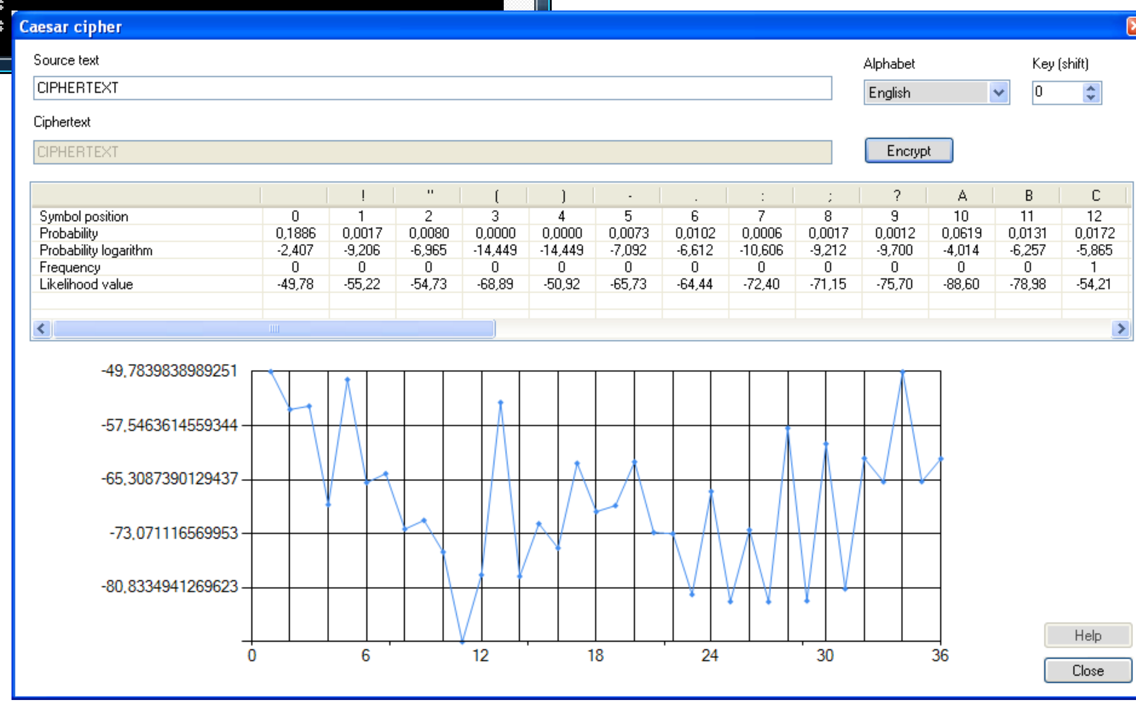
SolarisOS

OpenSolaris

Darwin

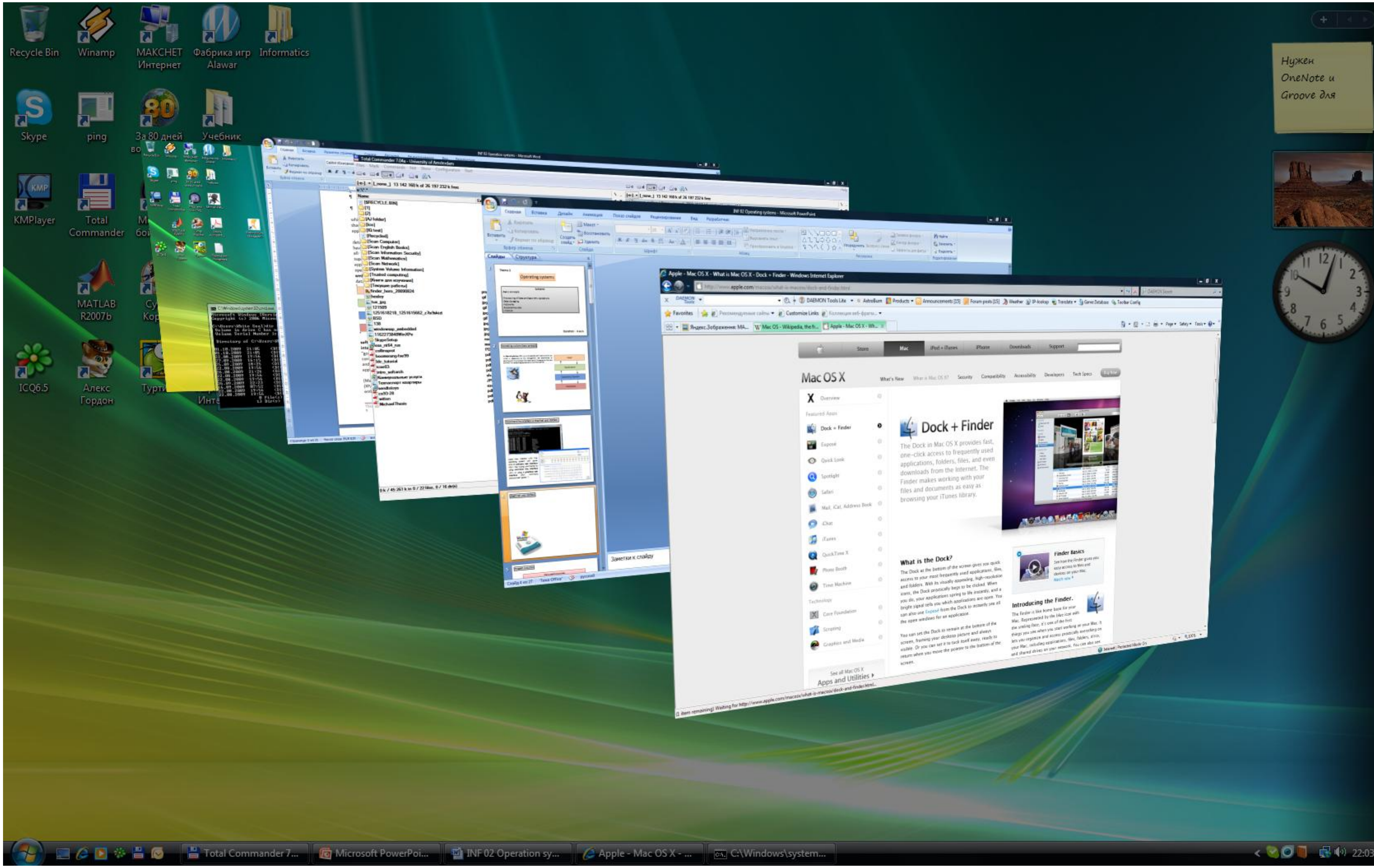
Command line interface vs. Graphical user interface





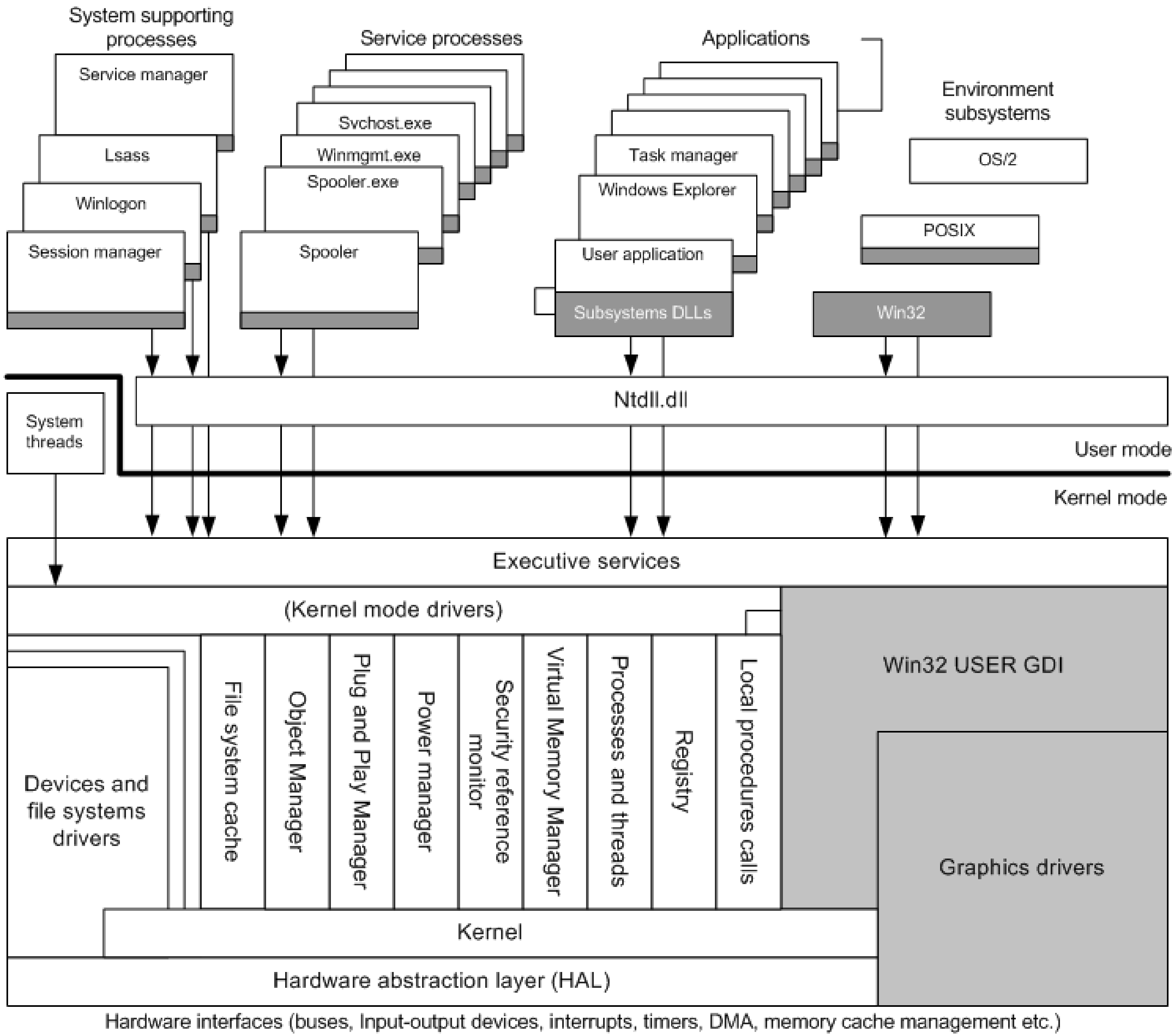
Users may interact with the operating system with some kind of **software user interface** (SUI) like typing commands by using **command line interface** (CLI) or using a **graphical user interface** (GUI, commonly pronounced “gooey”).

Graphical user interface



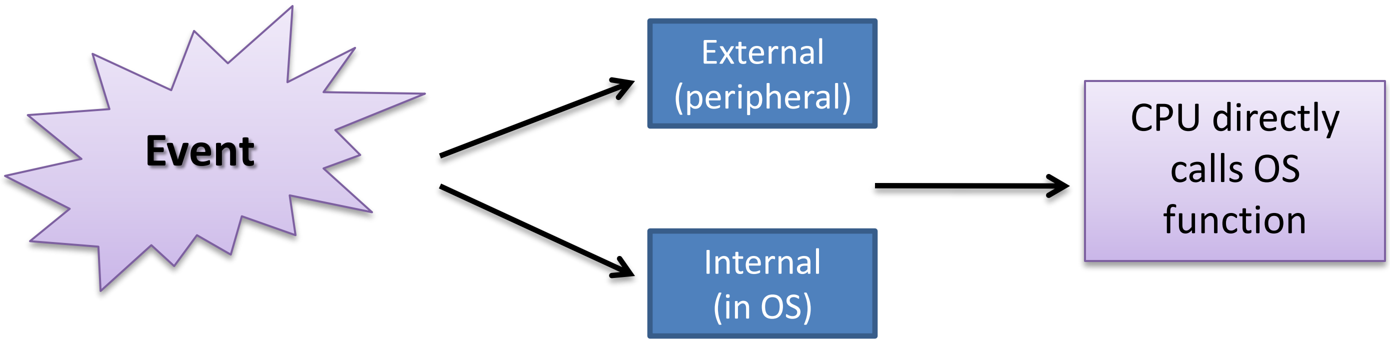
GUI is more preferred !!!

Windows architecture



Operating system modes

Interrupts. Interrupt-based programming is directly supported by most CPUs. Interrupts provide a computer with a way of automatically running specific code in response to events.



When a computer first starts up, it is automatically running in **supervisor mode**. The first few programs to run on the computer, being the **BIOS**, **bootloader** and the **operating system** have unlimited access to hardware.

In **protected mode**, programs may have access to a more limited set of the CPU's instructions. A user program may leave protected mode only by triggering an **interrupt**, causing control to be passed back to the kernel. In this way the operating system can maintain exclusive control over things like access to hardware and memory.

Program execution

The operating system acts as an interface between an application and the hardware; this system is a set of services which simplifies development of applications. Executing a program involves the creation of a **process** by the operating system.

Start creation of process

Assigning memory and supporting resources

Establishing priority for the process

Loading program code into memory

Executing program

Kernel full control

Processes and Threads

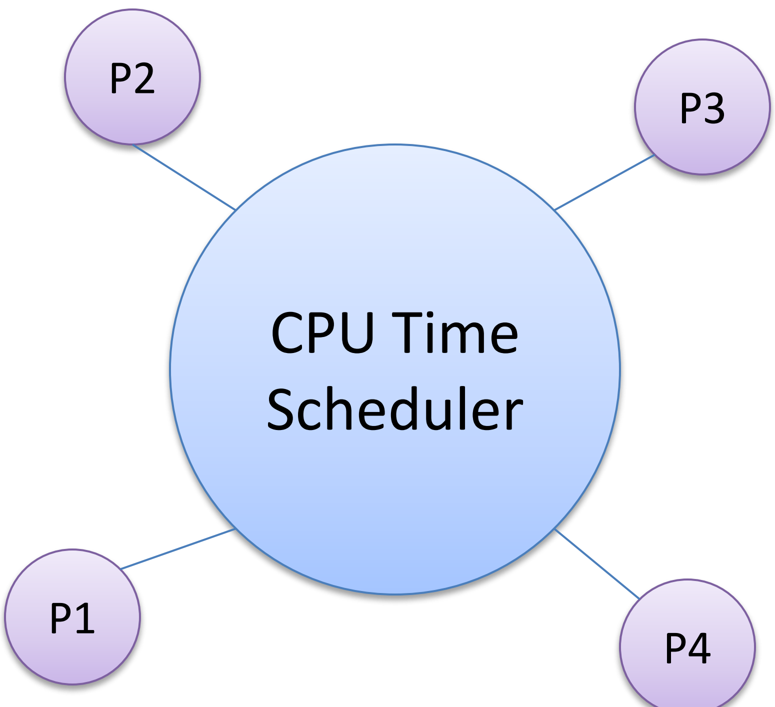
What is process?

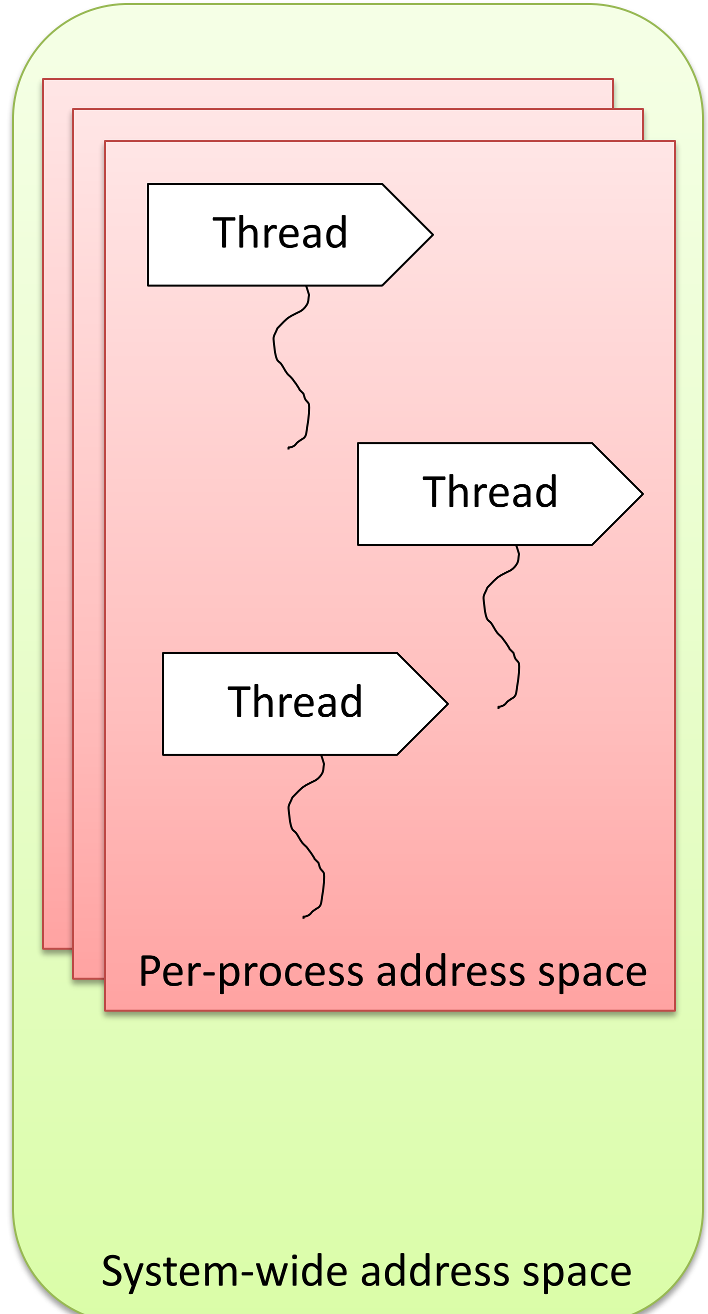
Represents an instance of a running program

- You create a process to run a program
- Starting an application creates a process

Process defined by

- Address space
- Resources (e.g., open handles)
- Security profile (token)





Processes and Threads

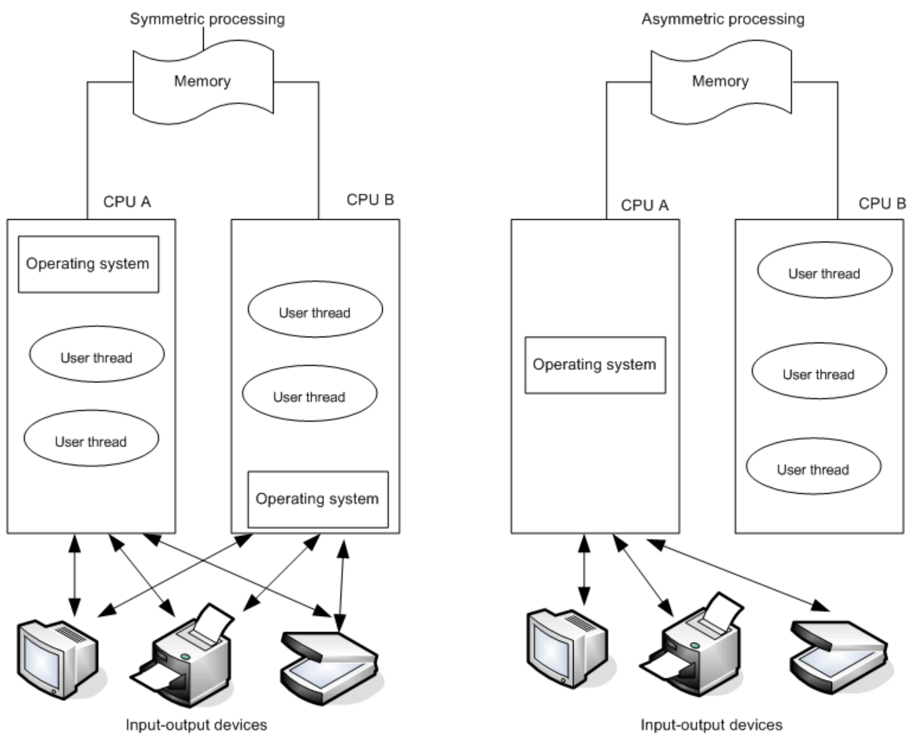
What is thread?

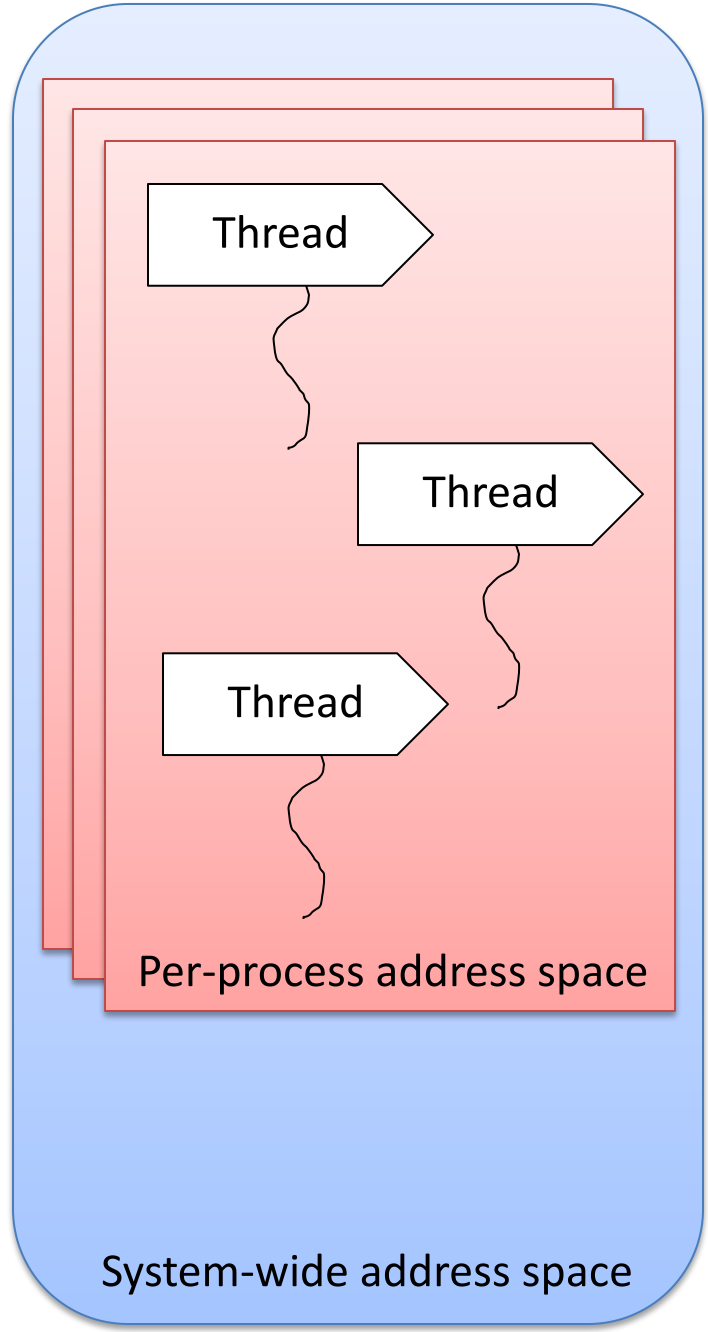
Represents an instance of a running program

- An execution context within a process
- Unit of scheduling (threads run, processes don't run)

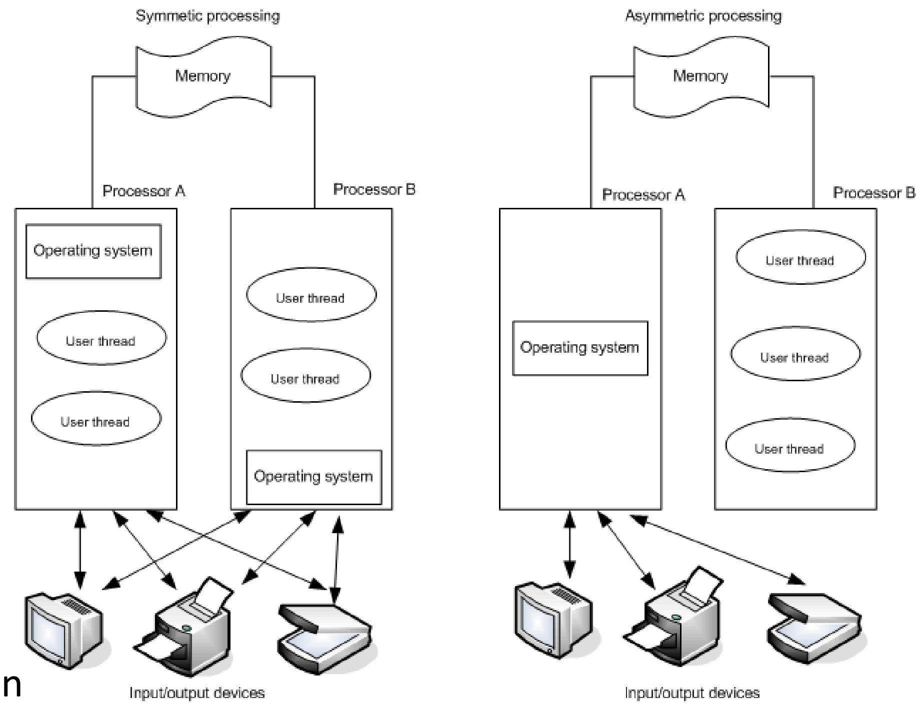
All threads in a process share the same per-process address space

All threads in the system are scheduled as peers to all others, without regard to their "parent" process





Processes And Threads



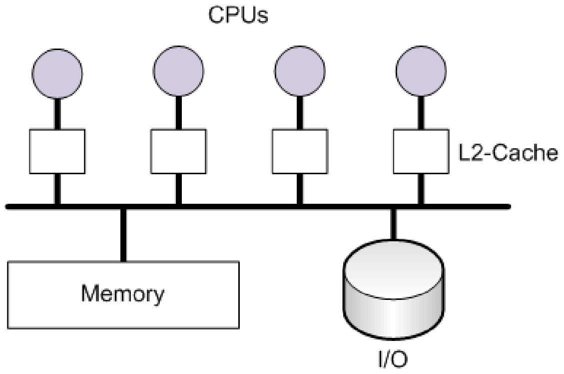
Every process starts with one thread

- First thread executes the program's "main" function
  - Can create other threads in the same process
  - Can create additional processes

Why divide an application into multiple threads?

- Perceived user responsiveness, parallel/background execution
  - Examples: Word background print – can continue to edit during print
- Take advantage of multiple processors
  - On an MP system with n CPUs, n threads can literally run at the same time
  - Question: Given a single threaded application, will adding a second processor make it run faster?
- Does add complexity
  - Synchronization
  - Scalability well is a different question…
    - Number of multiple runnable threads versus number CPUs
    - Having too many runnable threads causes excess context switching

Symmetric Multiprocessing (SMP)



No master processor

- All the processors share just one memory space
- Interrupts can be serviced on any processor
- Any CPU can cause another CPU to reschedule what it's running

Hyperthreading support

CPU fools OS into thinking there are multiple CPUs

Example: dual Xeon with hyperthreading can support 2 logical processors

XP, Vista & Windows Server are hyperthreading aware

Logical processors don't count against physical CPU limits

Scheduling algorithms take into account logical vs physical processors

Dual Core

Processor licensing is per-socket

NUMA (non uniform memory architecture) – supports only in Server versions

Groups of physical processors (called "nodes") that have "local memory"

Still an SMP system (e.g. any processor can access all of memory)

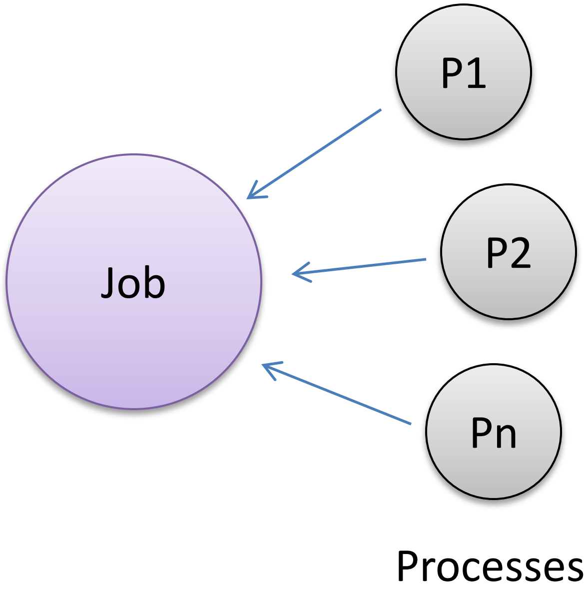
But node-local memory is faster

Scheduling algorithms take this into account

Jobs

It is kernel object to **manage groups of processes**.

- Set limits on a process or group of processes.



A job object allows control of certain attributes and provides limits for the process or processes associated with the job. It also records basic accounting information for all processes associated with the job and for all processes that were associated with the job but have since terminated. In some ways, the job object compensates for the lack of a structured process tree in Windows – yet in many ways it is more powerful than a UNIX-style process tree.

Quotas and restrictions:

Quotas: total CPU time, # active processes, per-process CPU time, memory usage

Run-time restrictions: priority of all the processes in job; processors threads in job can run on

Security restrictions: limits what processes can do

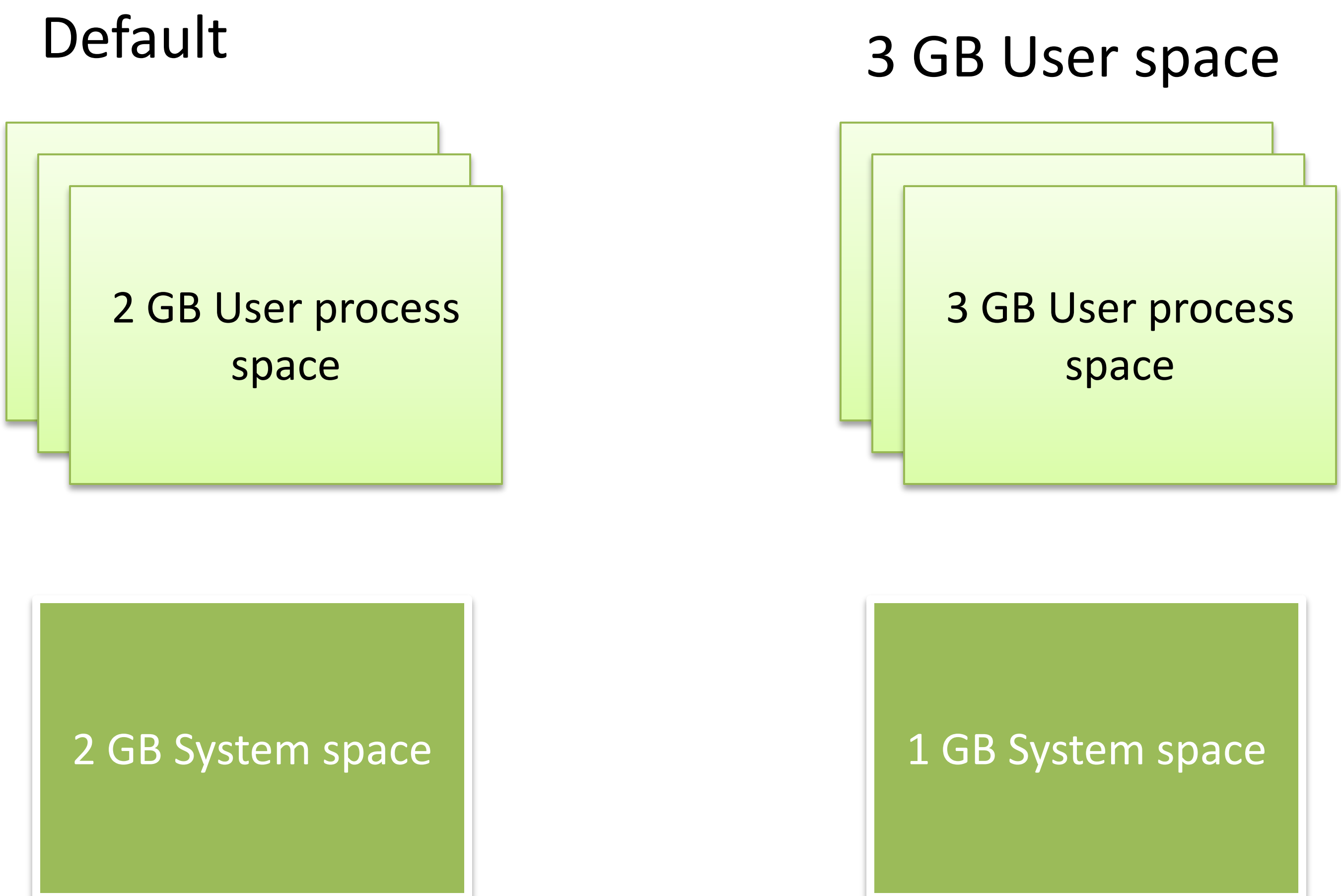
- not acquire administrative privileges
- not accessing windows outside the job, no reading/writing the clipboard

Scheduling class: number from 0-9 (5 is default) - affects length of thread timeslice (or quantum); e.g. can be used to achieve “class scheduling” (partition CPU)

Only Datacenter Server version has a built-in tool to take advantage of jobs

32-bit x86 Address Space

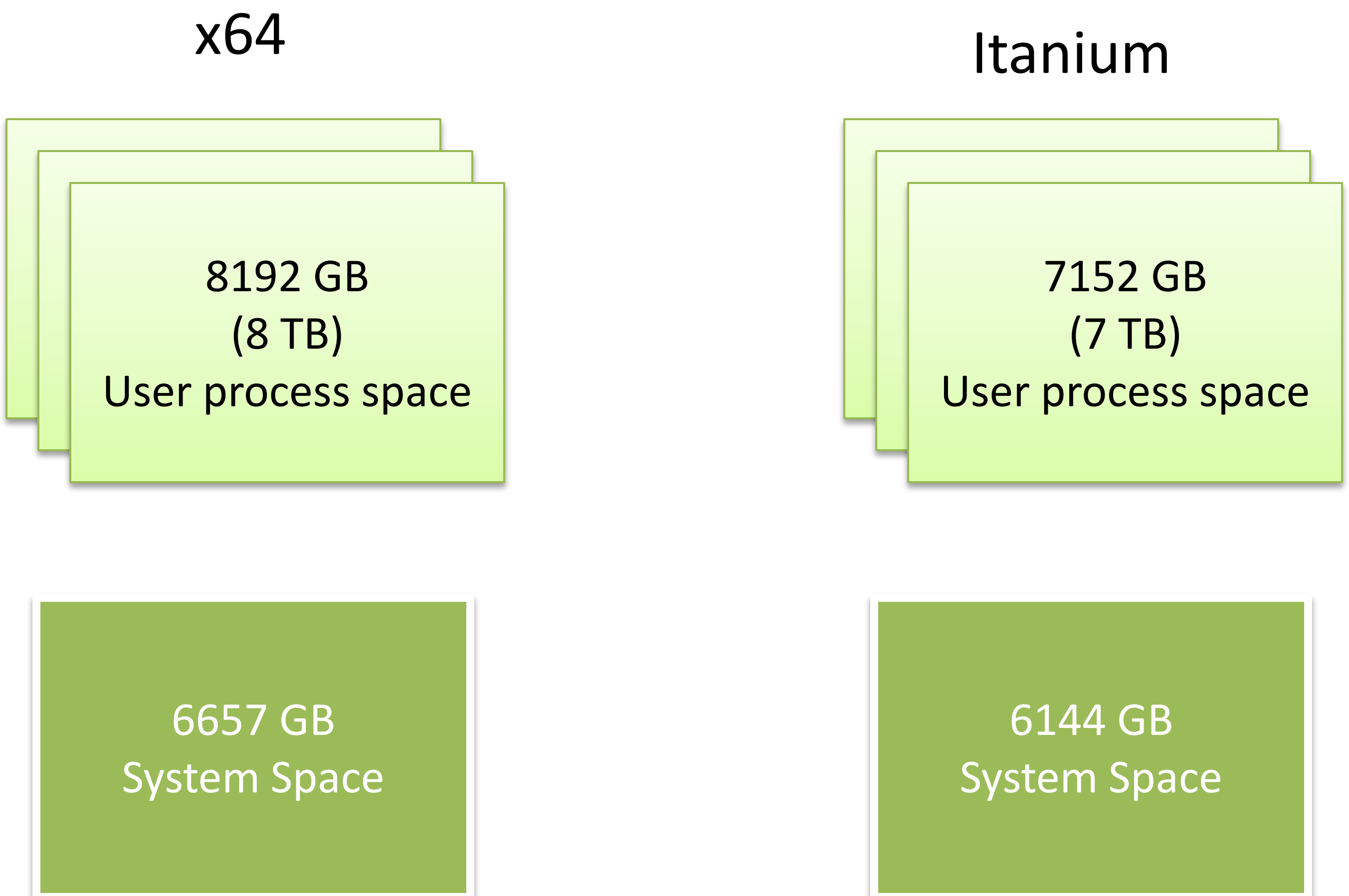
32-bits = 4 GB



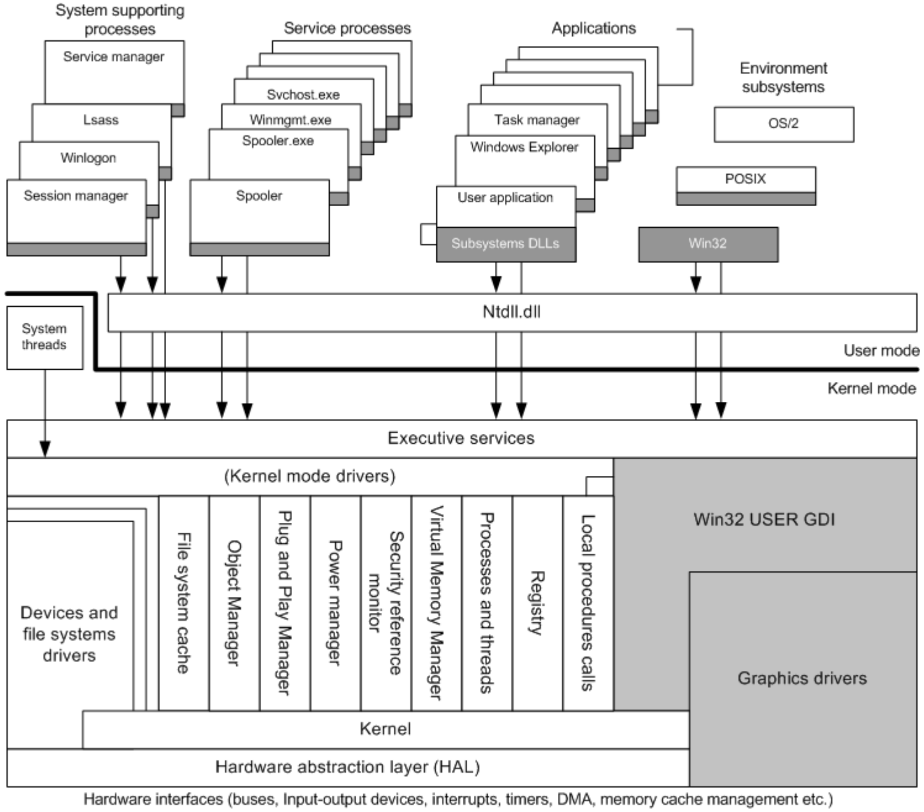
64-bit Address Spaces

64-bits = 17,179,869,184 GB

- x64 today supports 48 bits virtual = 262,144 GB
- IA-64 today support 50 bits virtual = 1,048,576 GB



Windows Kernel



Lower layers of the operating system

Implements processor-dependent functions (x86 versus Itanium, etc.)

Also implements many processor-independent functions that are closely associated with processor-dependent functions

Main services

- Thread waiting, scheduling, and context switching
- Exception and interrupt dispatching
- Operating system synchronization primitives (different for MP versus UP)
- A few of these are exposed to user mode

Not a classic “microkernel” (shares address space with rest of kernel-mode components)

Windows Kernel Evolution

Basic kernel architecture has remained stable while system has evolved

- Windows 2000: major changes in I/O subsystem (plug & play, power management, WDM), but rest similar to NT4
- Windows XP & Server 2003: modest upgrades as compared to the changes from NT4 to Windows 2000

Internal version numbers confirm this:

- Windows 2000 was 5.0
- Windows XP is 5.1
- Windows Server 2003 is 5.2

- Windows Vista is 6.0 (the same for SP1 and SP2)
- Windows 2008 is 6.1 (Build 7600)
- Windows 7 is 6.1 (build 7600)

Windows Kernel

Is Windows NT/2000/XP/2003 a microkernel-based OS?

- No – not using the academic definition (OS components and drivers run in their own private address spaces, layered on a primitive microkernel)
- All kernel components live in a common shared address space
  - Therefore no protection between OS and drivers
- But it does have some attributes of a microkernel OS
  - OS personalities running in user space as separate processes
  - Kernel-mode components don't reach into one another's data structures
    - Use formal interfaces to pass parameters and access and/or modify data structures
- Therefore the term "modified microkernel"

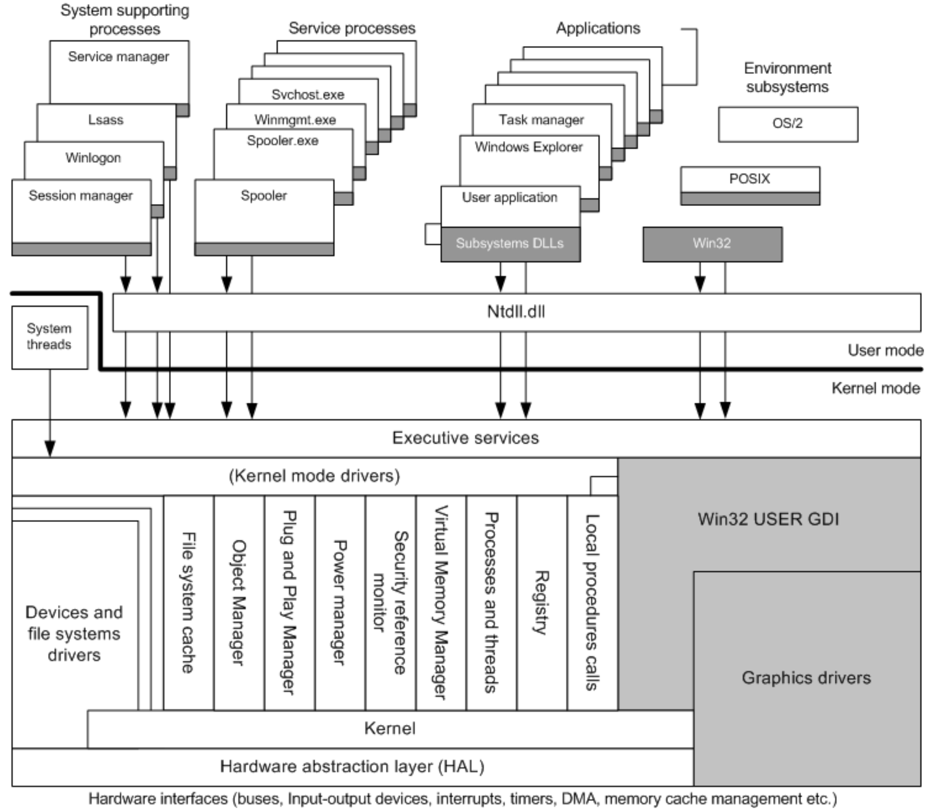
Why not pure microkernel?

- Performance – separate address spaces would mean context switching to call basic OS services

Linux has the same monolithic kernel architecture

- So do most Unix's, VMS, …

Hardware abstraction layer



Reduced role since Windows 2000

Bus support moved to bus drivers

Majority of HALs are vendor-independent

Responsible for a small part of "hardware abstraction"

Components on the motherboard not handled by drivers

System timers, Cache coherency, and flushing

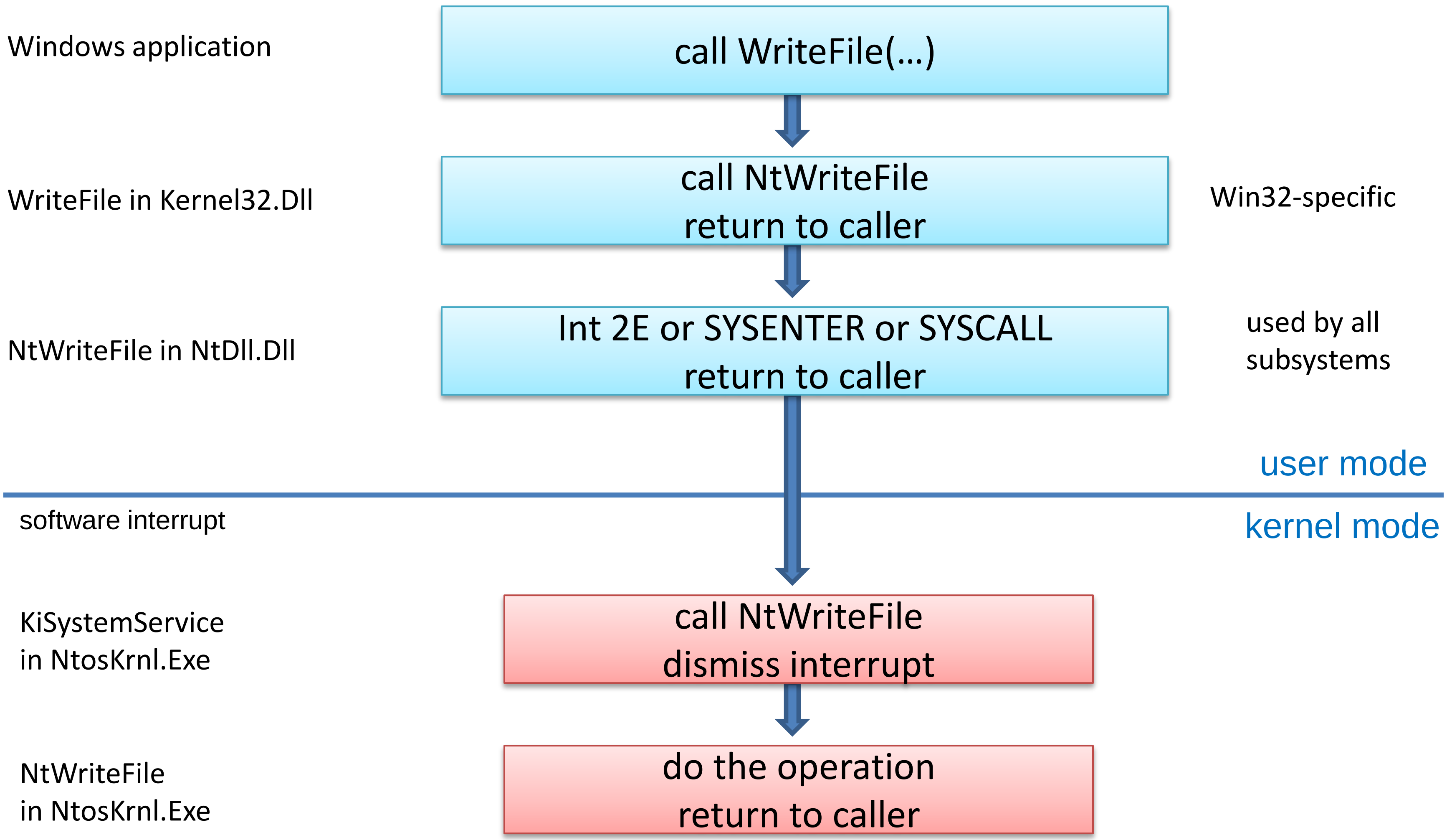
SMP support, Hardware interrupt priorities

Subroutine library for the kernel and device drivers

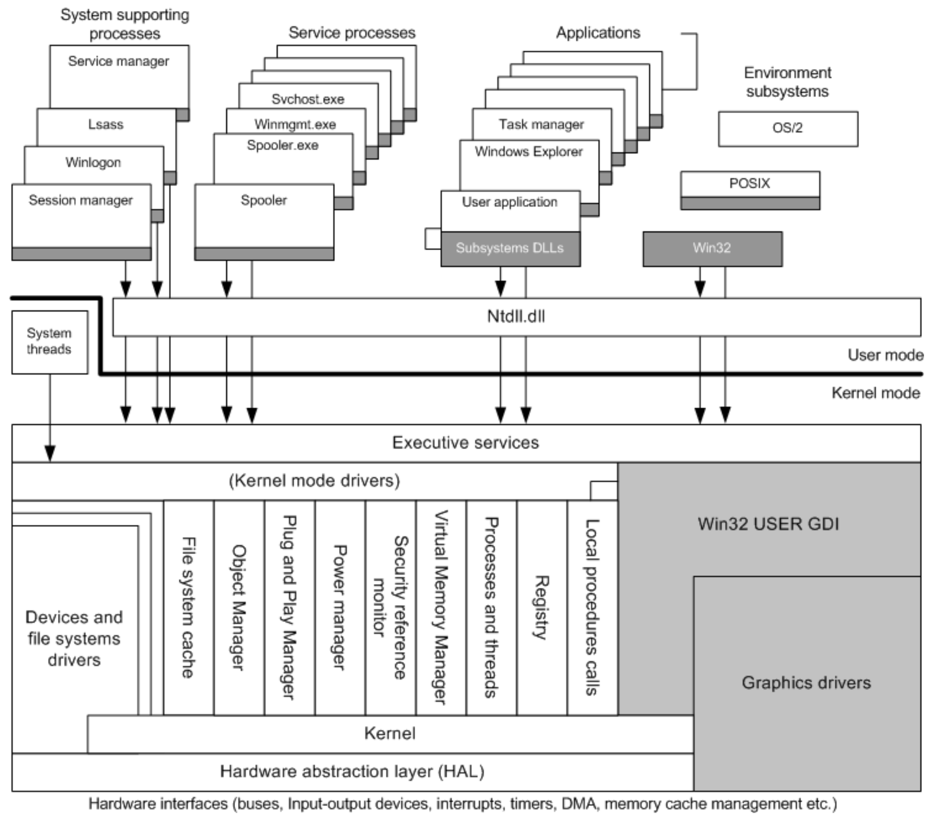
Isolates OS & drivers from platform-specific details

Presents uniform model of I/O hardware interface to drivers

Internal function call (Windows API translation)



Executive subsystem



- **Upper layer of the operating system**
- **Provides "generic OS" functions**
  - ❑ Process Manager
  - ❑ Object Manager
  - ❑ Cache Manager
  - ❑ LPC (local procedure call) facility
  - ❑ Configuration Manager
  - ❑ Memory Manager
  - ❑ Security Reference Monitor
  - ❑ I/O Manager
  - ❑ Power Manager
  - ❑ Plug-and-Play Manager
- **Almost completely portable C code**
- **Runs in kernel ("privileged", ring 0) mode**
- **Most interfaces to executive services not documented**

Thanks for attention