Lecture Notes for Lecture 2 of CS 5600 (Computer Systems) for the Fall 2019 session at the Northeastern University Silicon Valley Campus.

Operating Systems

Philip Gust,
Clinical Instructor
Department of Computer Science

Operating Systems

Review of Lecture 1

- In this first lecture we began to explore the physical characteristics of a computer systems.
- We examined the hardware components, and how they are organized, how information is represented, how the computer operates on information, and input/output.
- Finally, we learned about how the computer is programmed, including the instructions it can carry out, and how instructions are executed by the computer.

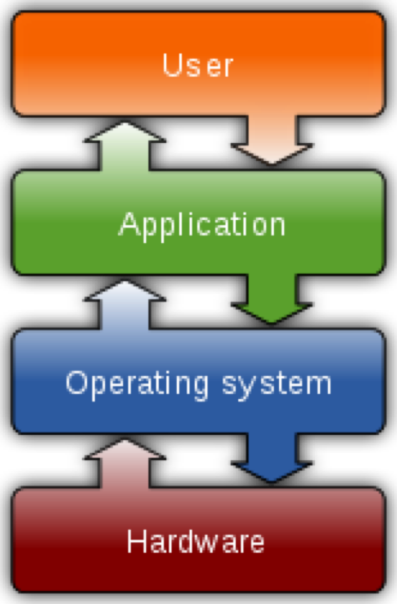
Operating Systems

Review of Lecture 1

- We learned that a *stored-program* computer stores data and instructions in memory, and can change programs over time by modifying the instructions as a result of computations.
- The computer architecture developed by John von Neumann is still the way computers are organized today,
- It consists of a CPU with registers, a control unit (CU) and an arithmetic logic unit; main memory with a memory controller and an array of memory locations; and I/O controllers.
- Instructions are a sequence of bytes that encode fields for the operation, address mode. Operations include loading/storing data, arithmetic, branching, and calling subroutines.

Operating Systems

Introduction



- An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.
- For functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware
- Although the application code is usually executed directly by the hardware it frequently makes system calls to an OS function or is interrupted by it.

Operating Systems

Introduction

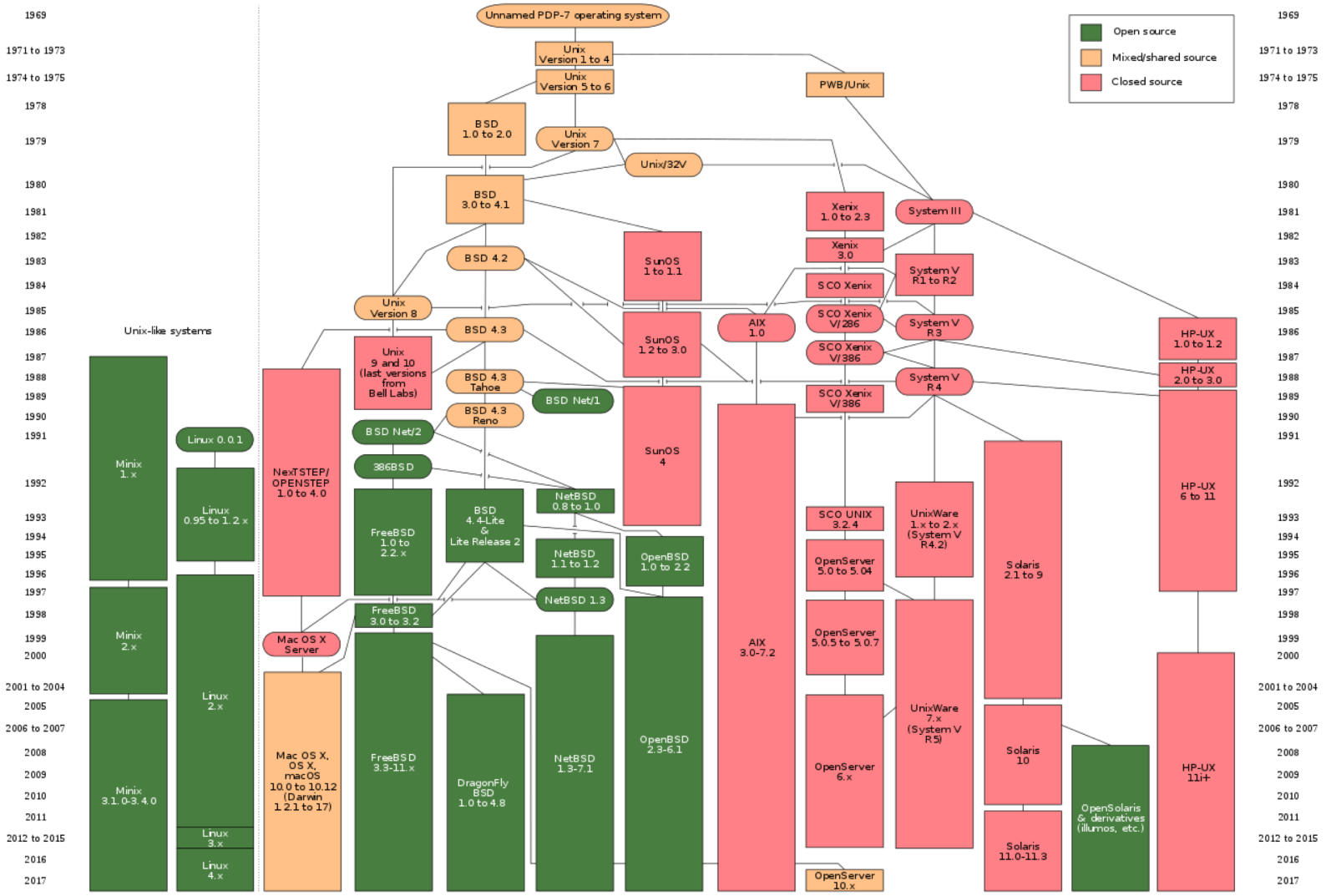
- Early computers were built to perform a series of single tasks. Programs were loaded into the computer memory, started, and once finished, the computer halted.
- Basic operating system features were developed in the 1950s, such as resident monitor functions that could automatically run different programs in succession to speed up processing.
- Operating systems did not exist in their modern form until the mid-1960s, when IBM's OS/360 pioneered the concept that the operating system keeps track of all system resources.
- This included program and data space allocation in main memory and file space in secondary storage, file locking during update, and reclaiming resources on termination.

Operating Systems

Introduction

- A breakthrough in operating systems architecture was the development of Unix at AT&T Bell Laboratories in the early to mid 1970s by Ken Thompson, Dennis Richie and others.
- Originally written in assembly language, in 1973 it was rewritten in a new systems language C, and became the first truly portable operating system.
- It was widely distributed to universities starting in the mid-1970s, and licensed commercially starting in the late 1970s.
- There have been many implementations of Unix, and many operating systems inspired by its design since then.

Operating Systems



Operating Systems

Introduction

- Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy".
- The operating system provided applications with a coherent and unified library of services and system calls for interacting with the operating system.
- A simplified, file model treats files as simple byte streams. The file system hierarchy also includes services and devices such as printers, terminals, or disk drives).
- Unix also provides a set of commands that perform limited, well-defined functions, and a scriptable shell language for combing them to perform complex workflows.

Operating Systems

Introduction

- In the late 1980s, an IEEE standardization effort known as POSIX (Portable Operating System Interface), provided a common baseline for all operating systems.
- The IEEE based POSIX around the common structure of the major competing variants of the Unix system. The first POSIX standard was published in 1988.
- POSIX covers all levels of the operating system, including core services, APIs, command shells, and commands that can be supported across a wide variety of operating systems.
- Many operating systems are POSIX compliant including MacOS, Linux, AIX (IBM), HP-UX (HP). Cygwin provides a largely POSIX-compliant environment for Microsoft Windows.

Operating Systems

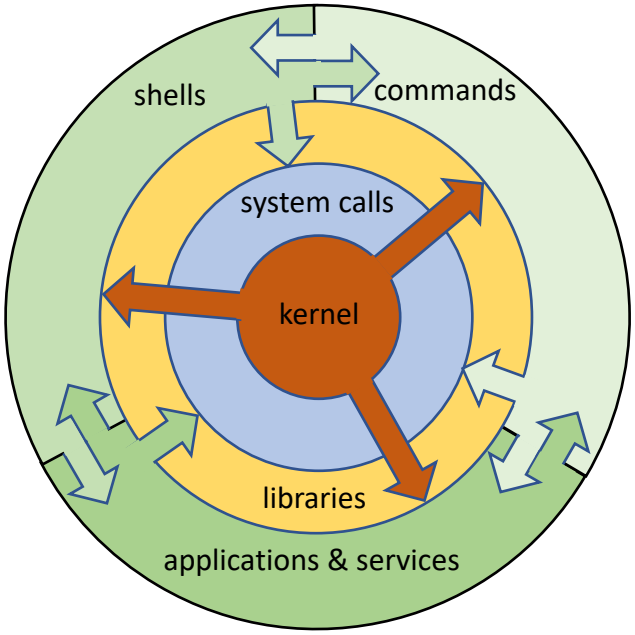
Operating System Architecture

- An POSIX operating system consists of standardized software abstraction layers for the underlying hardware.
- It provides interfaces at all levels that are implemented by the underlying operating system.
- We will begin by looking at the elements of the model at each layer and see how they interact with each other.
- Next, we will survey the inner-most layer, known as the kernel in more detail to understand the services it provides and the resources it manages.
- In future lectures we will look at many of these elements in greater depth.

Operating Systems

Operating System Architecture

- This diagram illustrates the layers of a POSIX operating system.



Operating Systems

Operating System Architecture

- The outer-most layer consists of programs that are run by a user, or operate autonomously to provide services.
- The program can be broadly classified as one of three types:
  1. *Shells* that provide an interactive environment for running and managing the execution of other programs. Also provides the ability to script operations performed by the shell.
  2. *Commands* are used by the shell to carry out its functions. Commands are software tools that extend the built-in commands provided by the shell to provide more advanced capabilities.
  3. *Applications and services* are either interactive programs attached to a display and input devices, or non-interactive programs that run in the background and provide services to other programs.

Operating Systems

Shells

- A *shell* is an interactive user interface that provides access to an operating system's services. It is named a shell because it is the outermost layer around the operating system kernel.
- In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation.
- POSIX includes a *shell command language* for a CLI. It describes the interactive features and syntax, and specifies built-in operations it can perform.
- The standard also describes features that make the language suitable as a scripting language, including values, variables, conditionals, control structures and defined functions.

Operating Systems

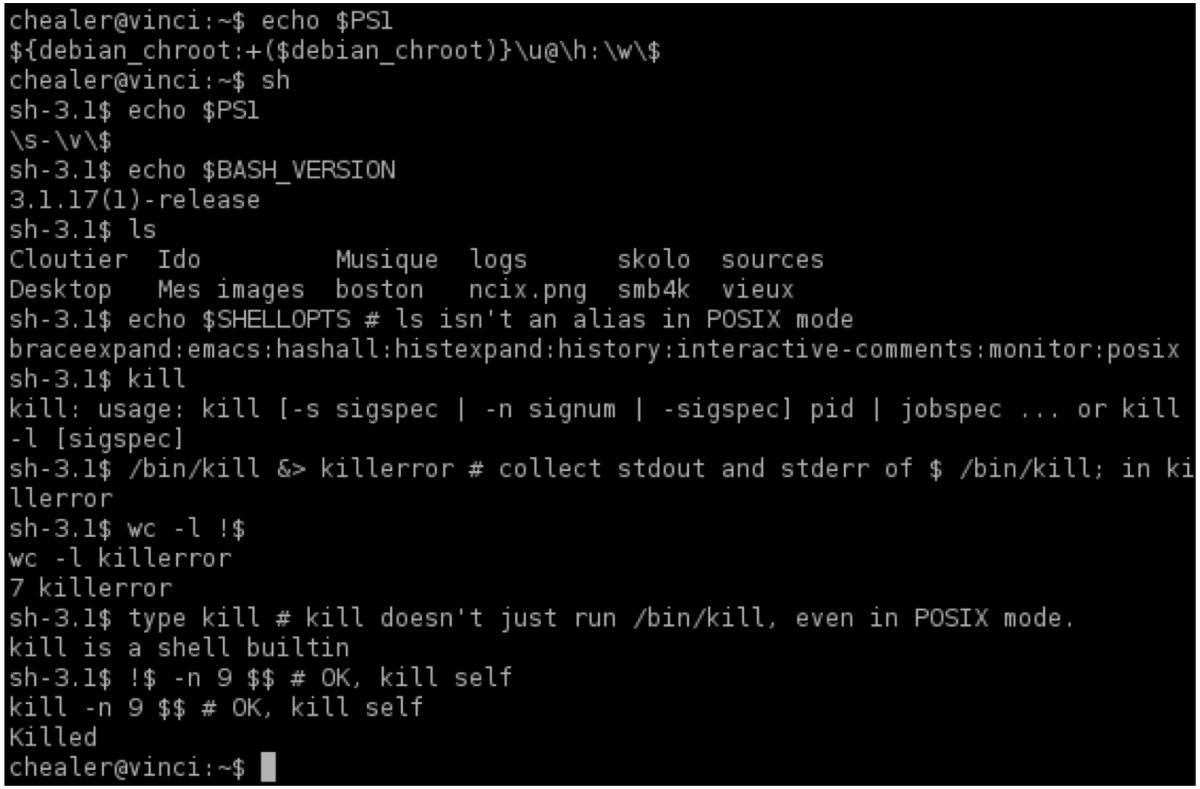
Shells

- The CLI standard was based on the Unix System V shell that was the most historically portable across a wide variety of platforms.
- The most commonly-used CLI shell is *bash*, which has numerous extension to the standard but also provides an option to run in strict POSIX-compliant mode.
- We will study the shell, its language, and its built-in commands in more detail in the next lecture.

Operating Systems

Shells

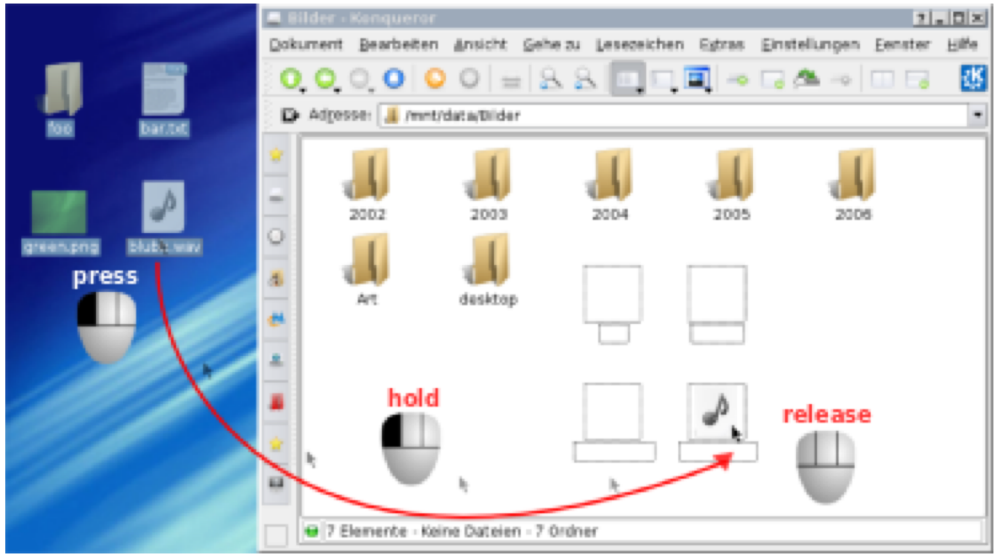
- Bash command line shell running in a terminal.



Operating Systems

Shells

- There is no POSIX standard for graphical user shells, although many POSIX-compliant systems provide the X11 window system and one of several graphical shells by default.
- MacOS X has a proprietary window system and GUI.



Operating Systems

Commands

- The commands built in to a shell provide only the most basic functionality. To supplement them, POSIX systems provide a standard collection of supplementary commands.
- These commands are built on the “Unix philosophy” that each one performs a specific function, with a number of options to customize its behavior.
- To perform a given task, the shell provides a way to “compose” commands to achieve the desired functionality.
- For example, many commands rely on being combine with the *sort* command to to produce sorted output rather than providing their own sorting functionality.

Operating Systems

Commands

- The POSIX standard specifies several hundred commands, primarily oriented towards text processing because much of the POSIX environment is text-based.
- The standard also specifies a set of options to customize the behavior of each command, and the syntax for each option.
- For example, the standard specifies options for the sort command, including specifying order (increasing, decreasing).
- Standardizing commands and their options makes it possible to write scripts that can run across various POSIX systems.
- We will look at the set of commands and useful ways that they can be combined within the shell in the next lecture.

Operating Systems

Applications and Services

- The primary role of the operating system is to enable running programs that perform some useful task. Programs can be divided into two categories: *applications* and *services*.
- An application is any program that is started by a user or an automation established by a user.
- Applications are typically associated with a display, and input devices such as a keyboard, mouse, touch screen, tablet, etc.
- Applications either run to completion once started, or are designed for interactive use by issuing commands through an input device and producing results to an output device.
- When the user is finished with the application, the program terminates under user control.

Operating Systems

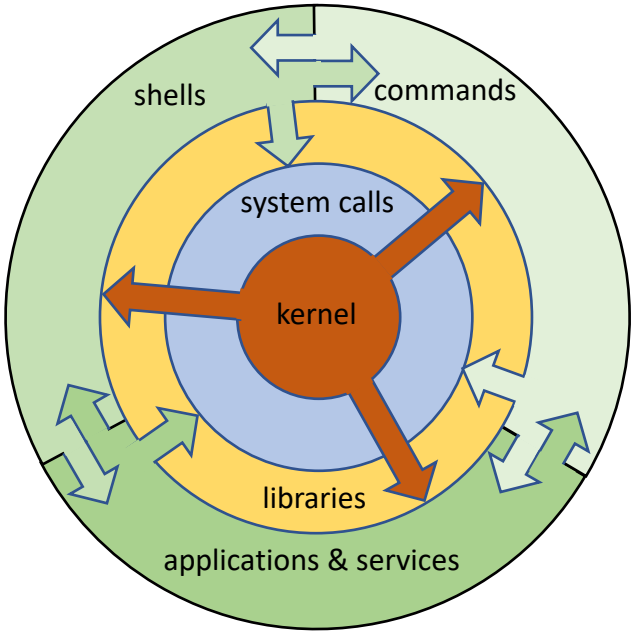
Applications and Services

- A service is a program that is configured to start automatically either when the operating systems begins running, or when some external event occurs.
- The service is typically not associated with a display or any input devices. Instead, it runs autonomously and provides a service that can be utilized by other programs.
- Services typically continue running until the system shuts down or some external event occurs.
- Examples of services are a web server, a print server, or a network file server that serves files to remote computers.
- We will look at how services are created and used in a later lectures.

Operating Systems

Libraries

- Applications and services, commands, and shells are built on a set of *libraries* that provide application-level functionality.



Operating Systems

Libraries

- Libraries are software components that can be combined with application code to produce program that can utilize the functions provided by the libraries.
- Various programming languages provide their own libraries, but POSIX focuses on those provided by the C programming language.
- POSIX specifies hundreds of library functions and their parameters that can be used to develop applications, services, commands, and even shells.
- In fact, most operating systems are built with many of the same libraries that are called by user-level programs.

Operating Systems

Libraries

- These functions do not communicate directly with the operating system kernel. Instead, they call lower-level system functions that represent operations the kernel can perform.
- Examples of library functions include mathematical functions, string functions, memory management functions, and higher-level input-output operations such as output formatting.
- We will look at how programs are translated from languages like C, and combined with library functions to create complete programs in lecture 4.
- We will also learn about the structure of libraries, how they are built, and ways that they can be combined with programs.

Operating Systems

System Calls

- The layer just outside the kernel is a collection of special functions known as *system calls* that provide a way for a program to request a service from the kernel.
- This includes hardware-related services (e.g. accessing a disk drive), creating and executing processes, and communicating with kernel services such as process scheduling.
- System calls provide an essential interface between a process and the operating system.

Operating Systems

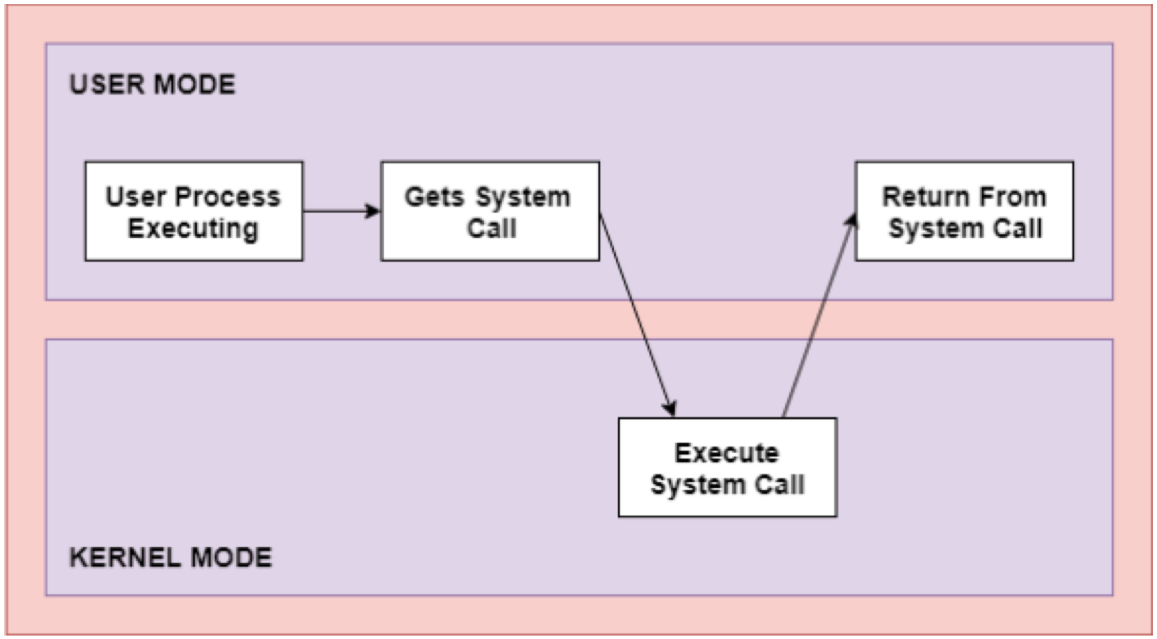
System Calls

- To protect operating system integrity, the kernel operates in a privileged mode, separated from user-level programs. System calls bridge the user and kernel spaces.
- Implementing system calls requires a transfer of control from user space to kernel space
- This is typically implemented with special CPU hardware operations that switch the processor from user to kernel mode with elevated privileges.
- When the kernel completes its operation, the processor switches to user mode again, and the system call returns the result to the program that called it.

Operating Systems

System Calls

- A system call requires switching from user mode to kernel mode and back. These switches are much slower than a normal function call.



Operating Systems

Kernel

- The role of the kernel is to provide managed access to a pool of system resources that are being managed on behalf of the programs and services running on the operating system.
- The kernel ensure that managed resources are utilized by programs and services in away that meets the resource management criteria of the operating system.
- The architecture of the kernel dictates what resources are managed and how the management policies are implemented.

Operating Systems

Kernel Functions

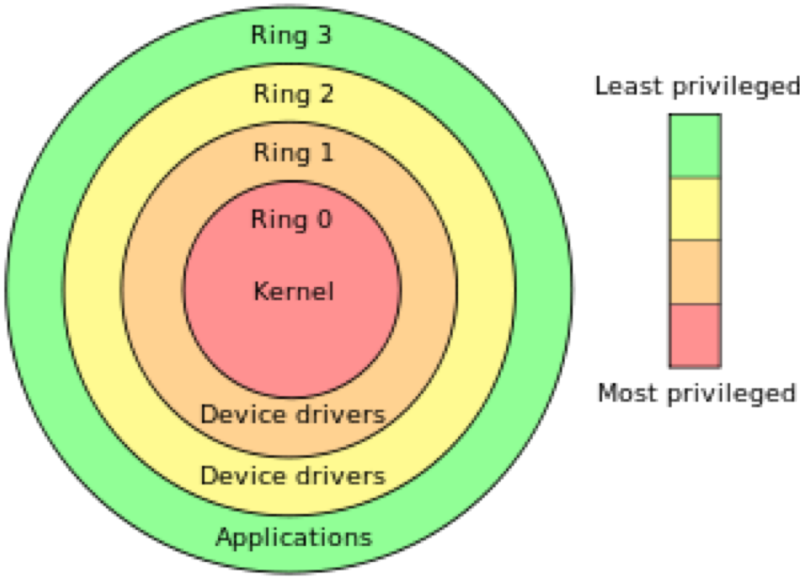
- The kernel provides access to these managed resources:

Category	Examples
Process control	These system calls deal with processes such as process creation, process termination etc.
Memory management	These system calls are used to control access to areas of memory and mapping and unmapping areas of memory.
File management	These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
Device management	These system calls are responsible for device manipulation such as reading and writing device buffers
Information maintenance	These system calls handle information and its transfer between the operating system and the user program.
Communication	These system calls perform inter-process communication, and creating and delete communication connections.

Operating Systems

Kernel Protection Domains

- Kernels control access to resources using a concept called *protection domains*. Since these are generally hierarchical, the are often referred to as protection rings.



Operating Systems

Kernel Protection Domains

- Programs like applications and services generally run in the least-privileged ring. This ensures that actions by an application or service cannot adversely impact other ones.
- Rings are arranged in a hierarchy from most privileged (most trusted, usually numbered zero) to least privileged (least trusted, usually with the highest ring number).
- On most operating systems, Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory.

Operating Systems

Kernel Protection Domains

- Special gates between rings allow an outer ring to access an inner ring's resources in a predefined manner, as opposed to allowing arbitrary usage.
- Correctly gating access between rings can improve security by preventing programs from one ring or privilege level from misusing resources intended for programs in another.
- For example, spyware running in Ring 3 is prevented from turning on a web camera without user permission, since hardware in Ring 1 is reserved for device drivers.
- Programs such as web browsers running in higher numbered rings must request access to the network, a resource restricted to a lower numbered ring.

Operating Systems

Kernel Protection Domains

- Many modern CPU architectures (including x86 architecture) include some form of ring protection. The original Multics system had eight rings, but many modern systems have fewer.
- The hardware remains aware of the current ring of the executing instruction thread at all times, with the help of a special machine register.
- The hardware restricts the ways in which control can be passed from one ring to another, and enforces restrictions on memory access that can be performed across rings.
- These hardware restrictions limit opportunities for accidental or malicious breaches of security.

Operating Systems

Kernel Architecture

- There are two major approaches to kernel architecture: *monolithic kernels*, and *micro-kernels*.
- While monolithic kernels execute all of their code in the same address space (kernel space), microkernels try to run most of their services in user space.
- Most kernels do not fit exactly into one of these categories, but are rather found in between these two designs. These are called *hybrid kernels*.

Operating Systems

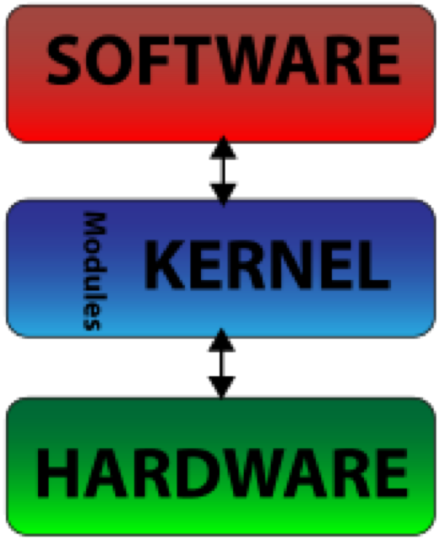
Kernel Architecture

- In a monolithic kernel, all OS services run along with the main kernel thread, thus also residing in the same memory area. This approach provides rich and powerful hardware access.
- Monolithic kernels, have traditionally been used by Unix-like operating systems, and contain all the operating system core functions and the device drivers.

Operating Systems

Kernel Architecture

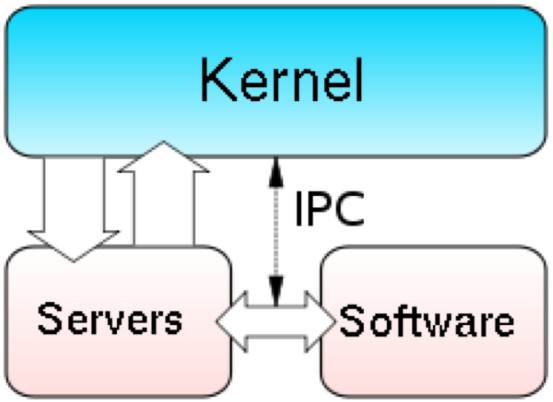
- A monolithic kernel is one single program that contains all of the code necessary to perform every kernel related task.
- Every part that is to be accessed by most programs which cannot be put in a library is in the kernel space: Device drivers, Scheduler, Memory handling, File systems, Network stacks.



Operating Systems

Kernel Architecture

- Microkernel (abbreviated µK or uK) is the term describing an approach to operating system design by which the functionality of the system is moved out of the traditional "kernel", into a set of "servers" that communicate through a "minimal" kernel



- This leaves as little as possible in "system space" and as much as possible in "user space".

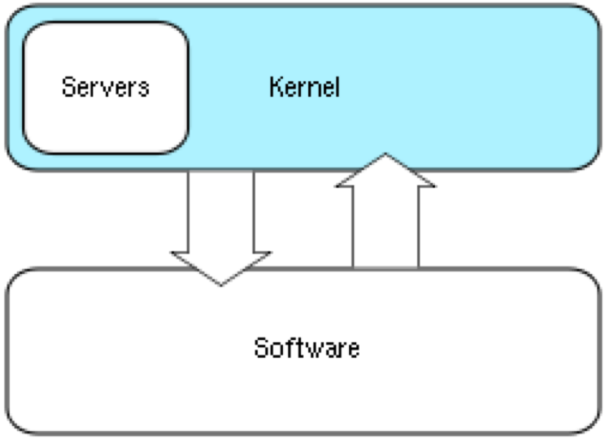
Operating Systems

Kernel Architecture

- The microkernel approach consists of defining a simple abstraction over the hardware, with a set of primitives or system calls to implement minimal OS services.
- These include memory management, multitasking, and inter-process communication.
- Other services, including those normally provided by the kernel, such as networking, are implemented in user-space programs, referred to as servers.
- Microkernels are easier to maintain than monolithic kernels, but the large number of system calls typically generate more overhead than plain function calls.

Operating Systems

Kernel Architecture



- Hybrid kernels are similar to micro-kernels, except they have additional code in kernel-space to increase performance.
- Hybrid kernels are micro kernels that have some "non-essential" code in kernel-space in order for the code to run more quickly than it would were it to be in user-space.
- Hybrid kernels are a compromise between the monolithic and microkernels