

Part 6

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure – MS-DOS
 - More complex -- UNIX
 - Layered – an abstraction
 - Microkernel -Mach

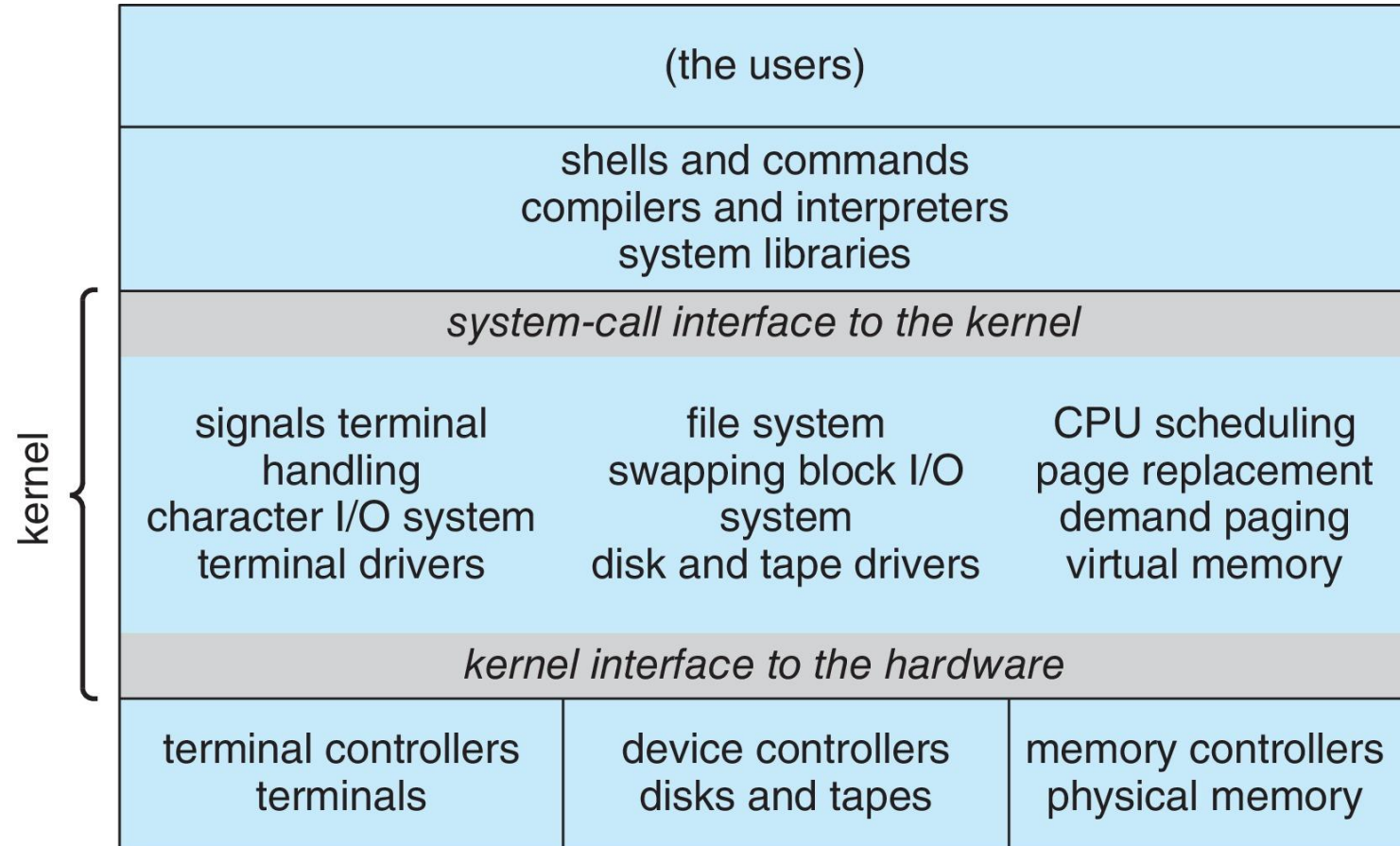
Monolithic Structure – Original UNIX

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

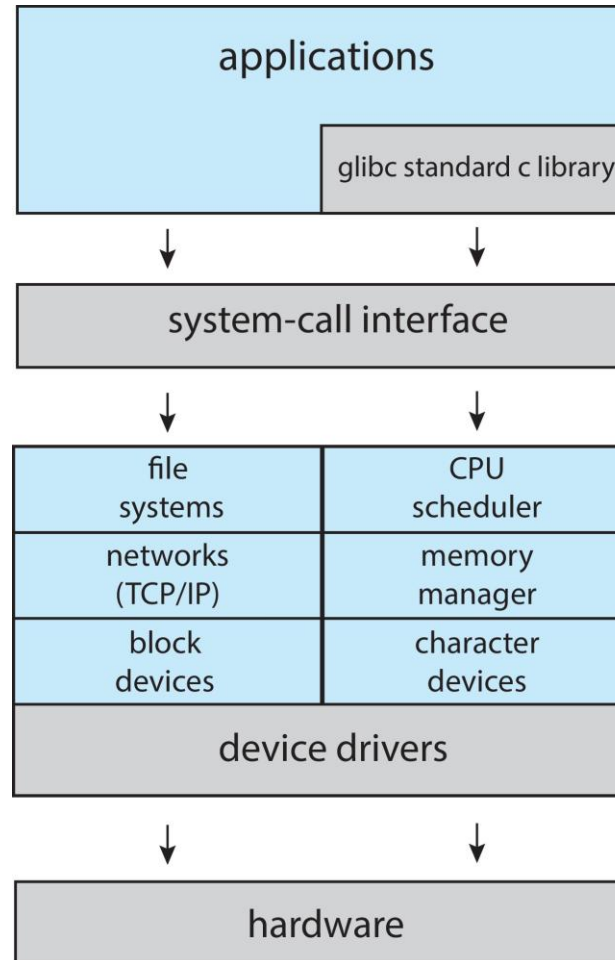
Traditional UNIX System Structure

Beyond simple but not fully layered

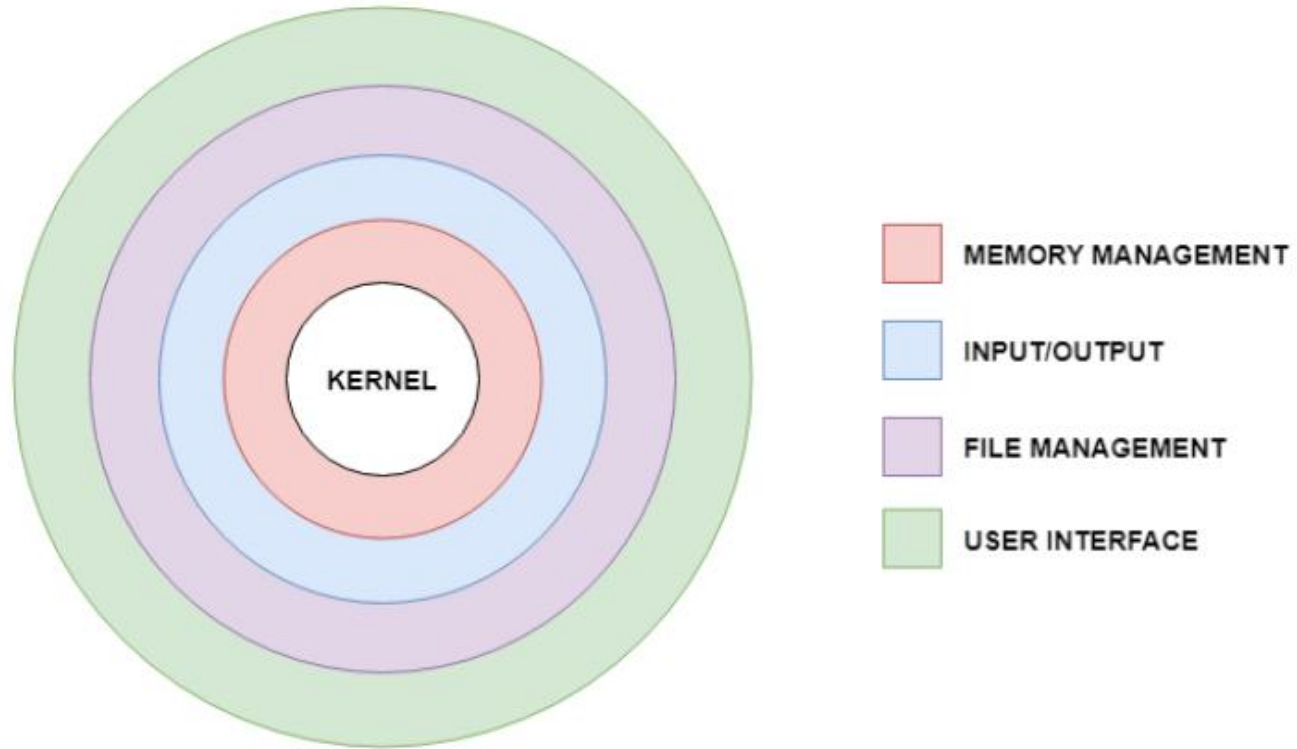


Linux System Structure

Monolithic plus modular design



Layered Approach

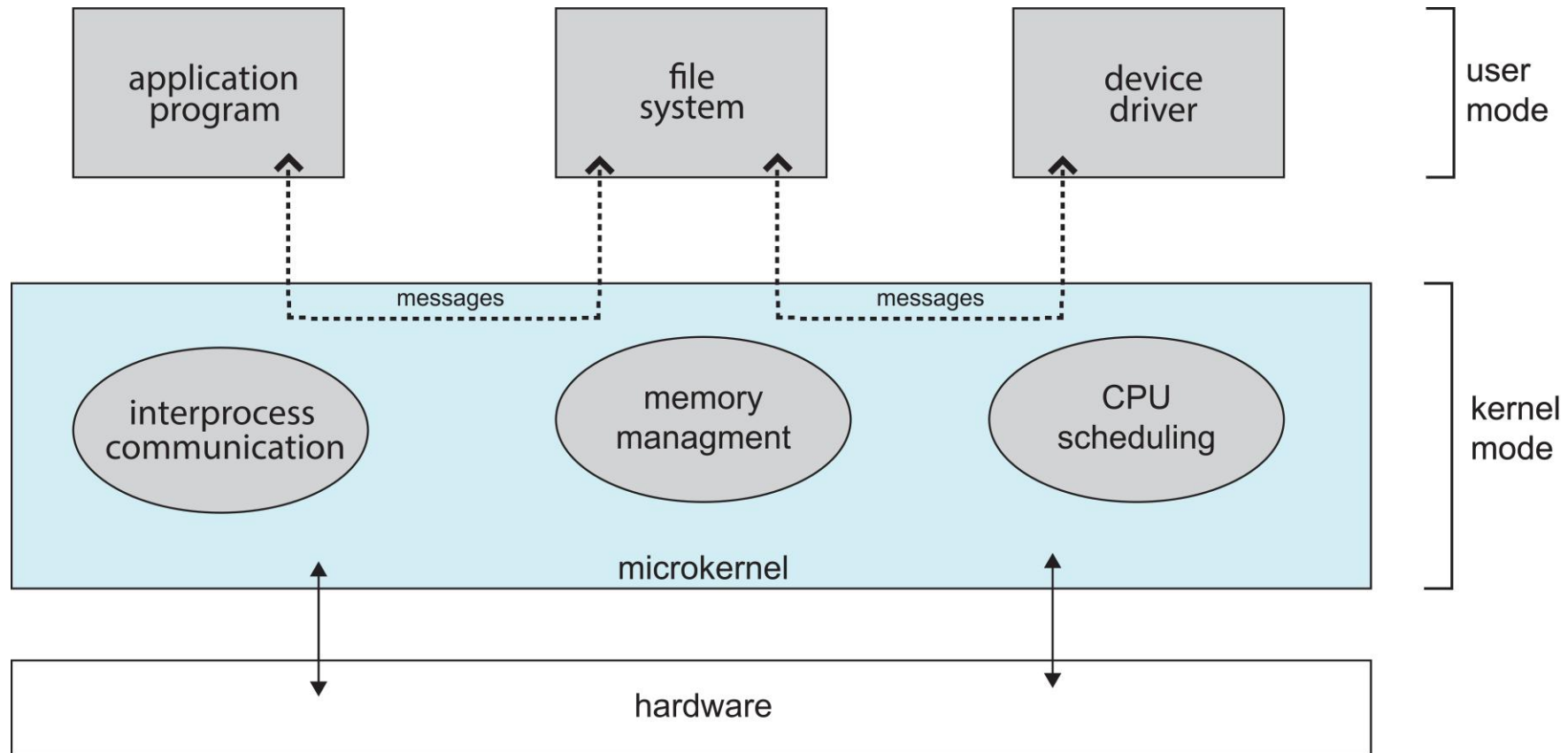


Layered Operating System Design

Microkernels

- ❑ 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

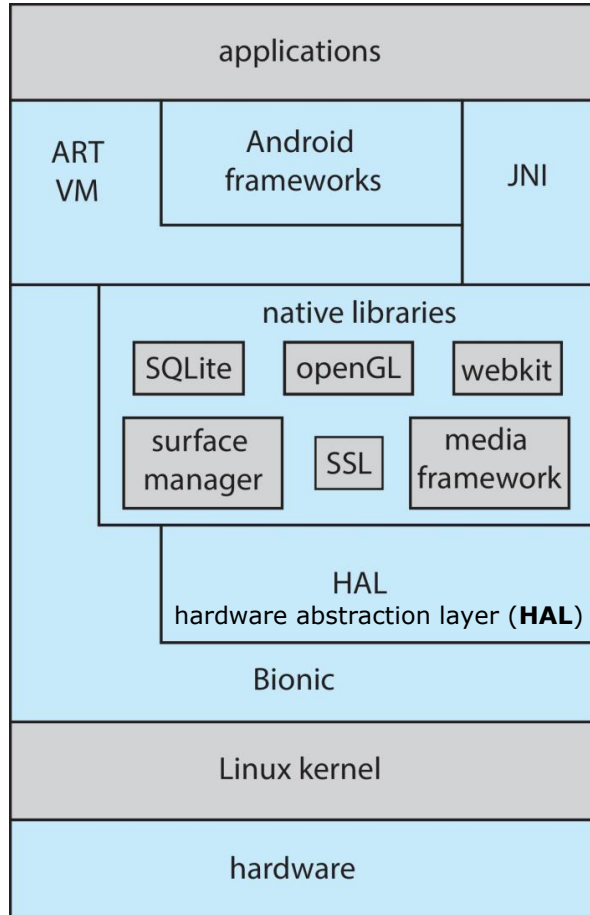
Microkernel System Structure



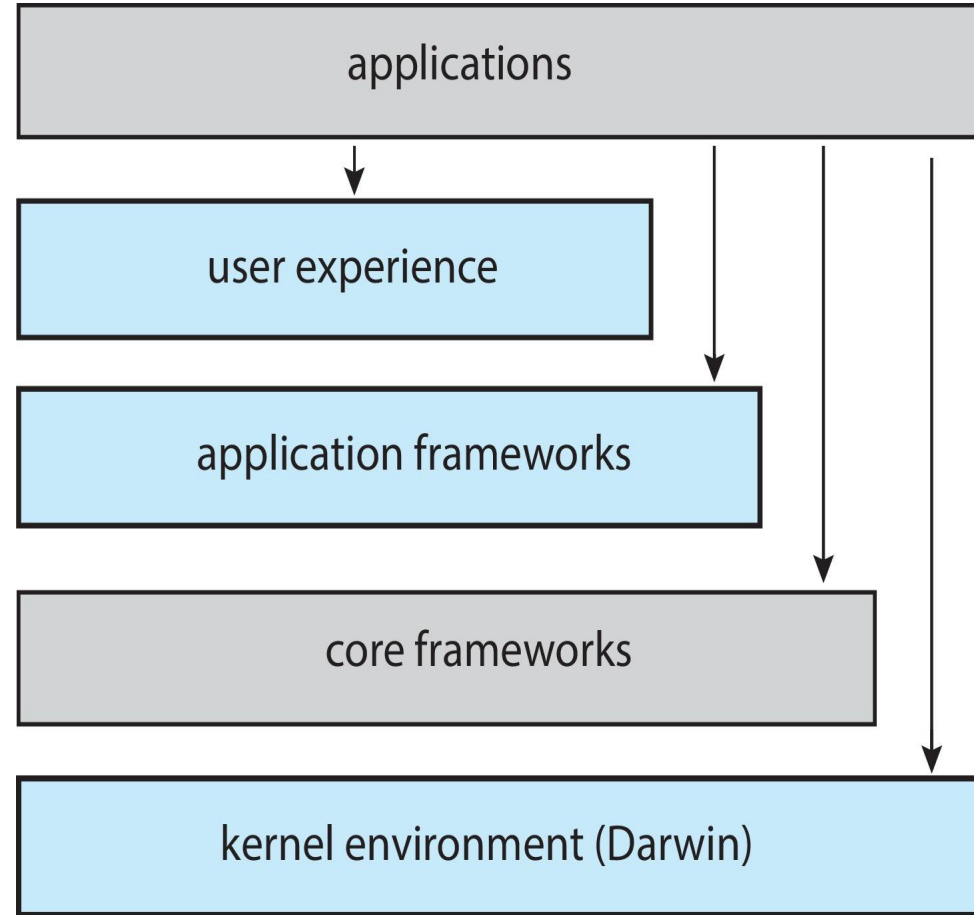
Modules

- Many modern operating systems implement **loadable kernel modules (LKMs)**
 - 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

Android vs iOS

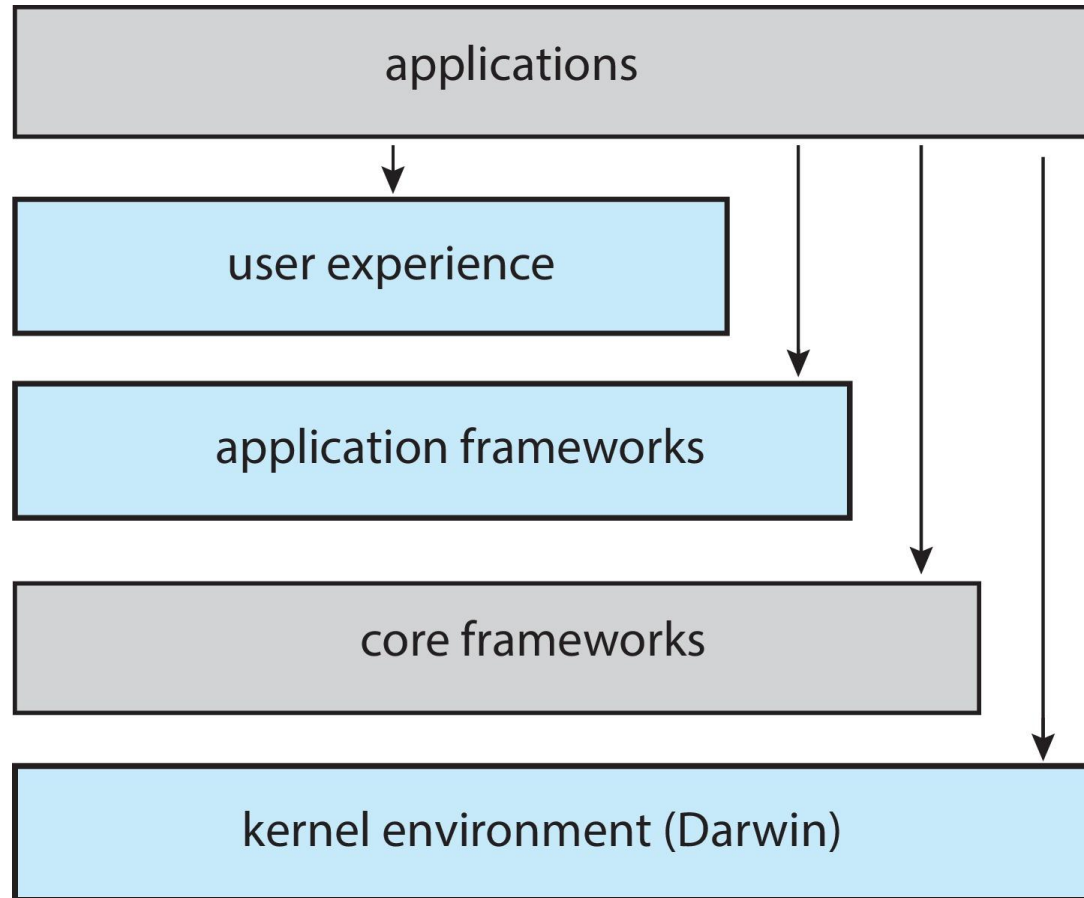


Android Architecture

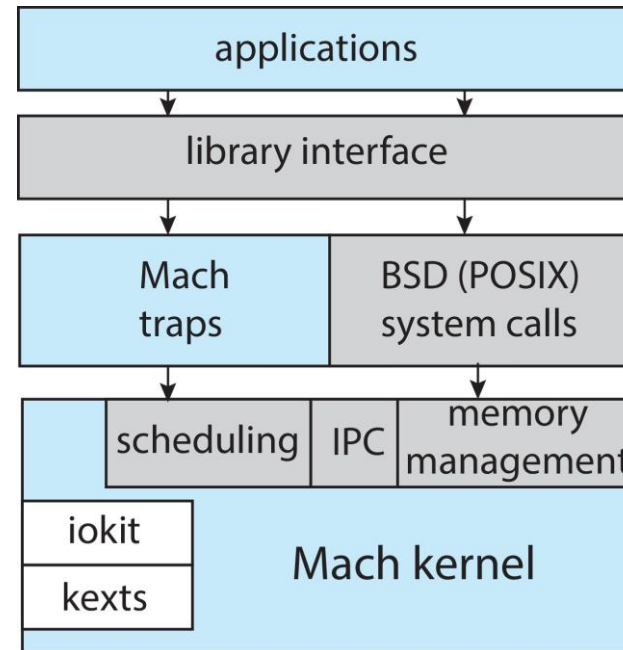


macOS and iOS Structure

macOS and iOS Structure



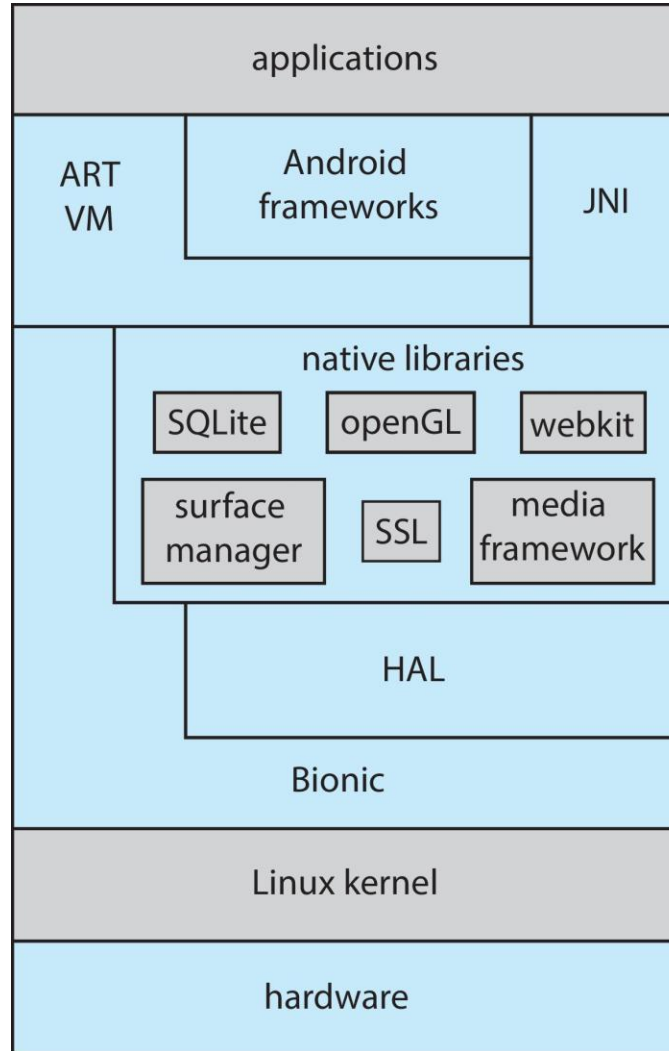
Darwin



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 then runs in Dalvik VM
- ❑ Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



Building and Booting an Operating System

- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
 - But can build and install some other operating systems
 - If generating an operating system from scratch
 - ▶ Write the operating system source code
 - ▶ Configure the operating system for the system on which it will run
 - ▶ Compile the operating system
 - ▶ Install the operating system
 - ▶ Boot the computer and its new operating system

Building and Booting Linux

- ❑ Download Linux source code (<http://www.kernel.org>)
- ❑ Configure kernel via `“make menuconfig”`
- ❑ Compile the kernel using `“make”`
 - ❑ Produces `vmlinuz`, the kernel image
 - ❑ Compile kernel modules via `“make modules”`
 - ❑ Install kernel modules into `vmlinuz` via `“make modules_install”`
 - ❑ Install new kernel on the system via `“make install”`

System Boot

- When power initialized on system, execution starts at **a fixed memory location**
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap loader**, **BIOS**, 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
 - Modern systems replace BIOS with **Unified Extensible Firmware Interface (UEFI)**
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**
- Boot loaders frequently allow various boot states, such as single user mode

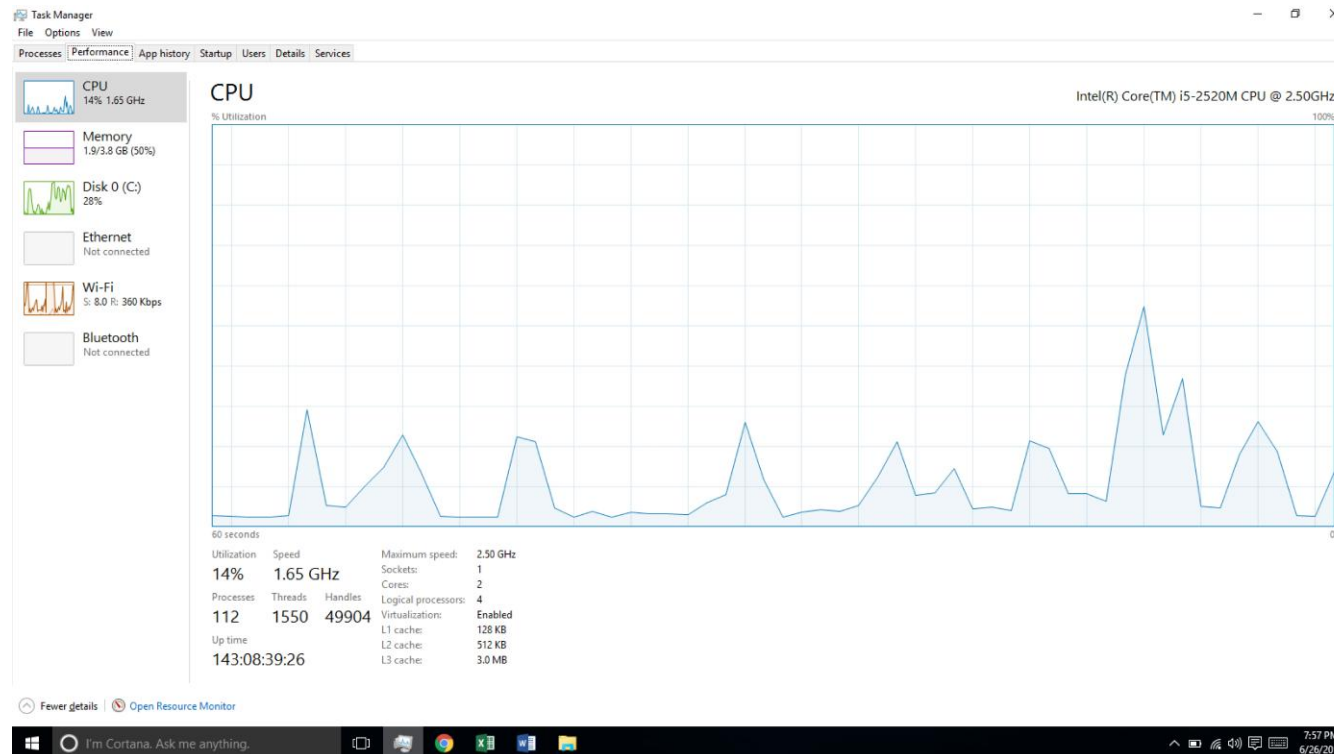
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- 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

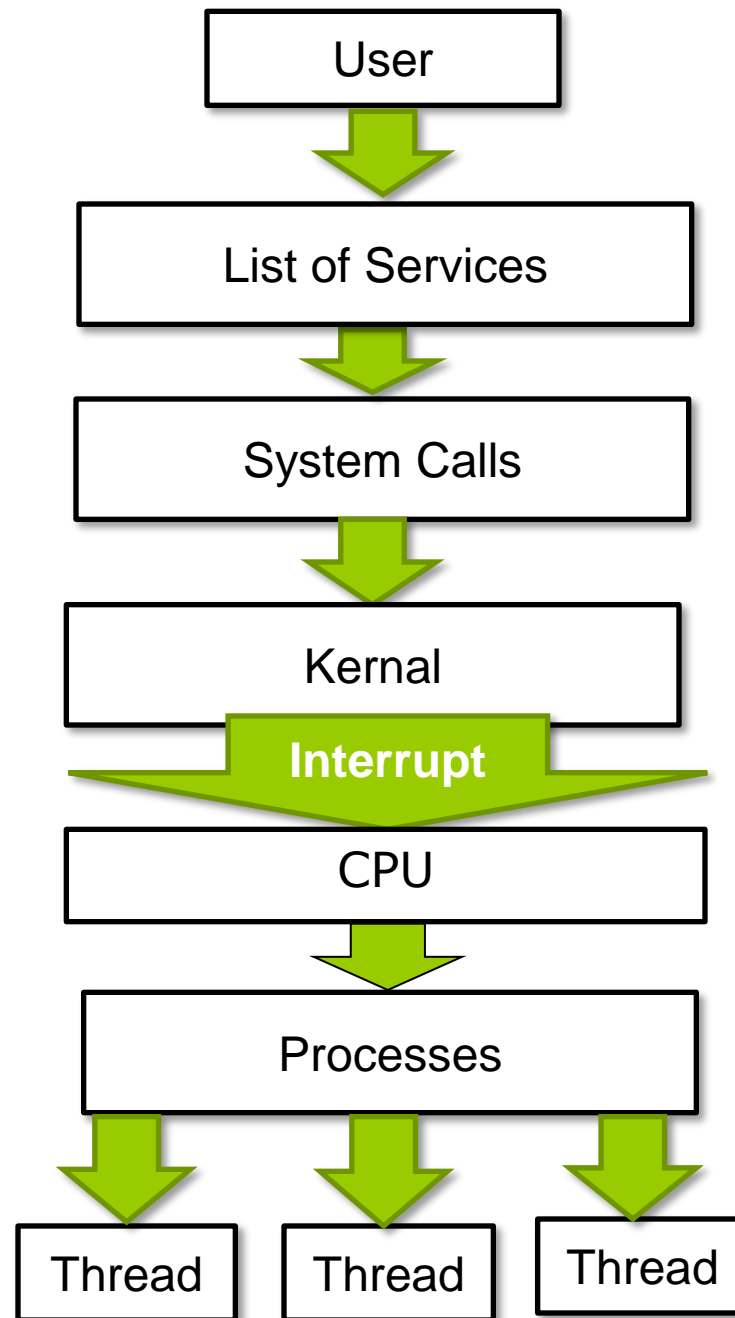
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

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



The Big Pic of OS



UX vs. UI



A list of UX/UI Words




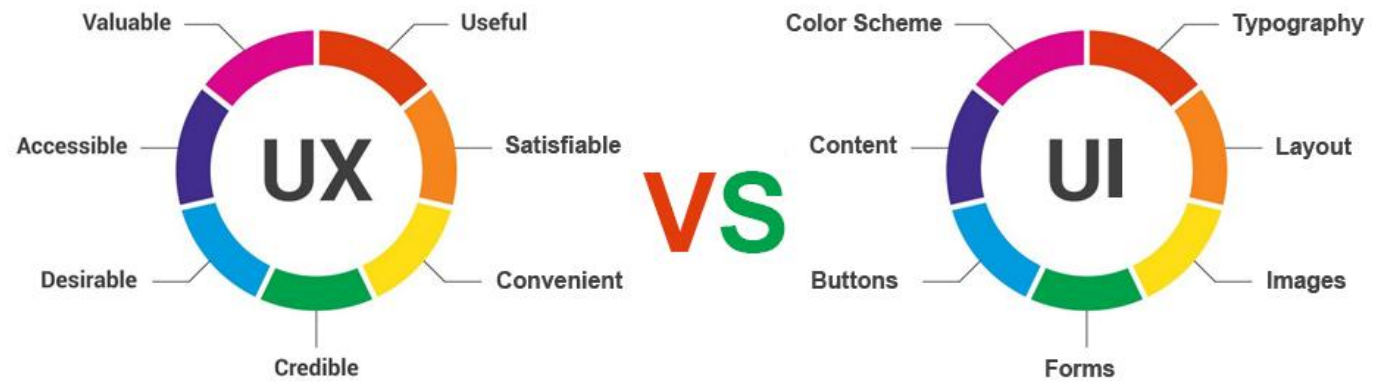
UX

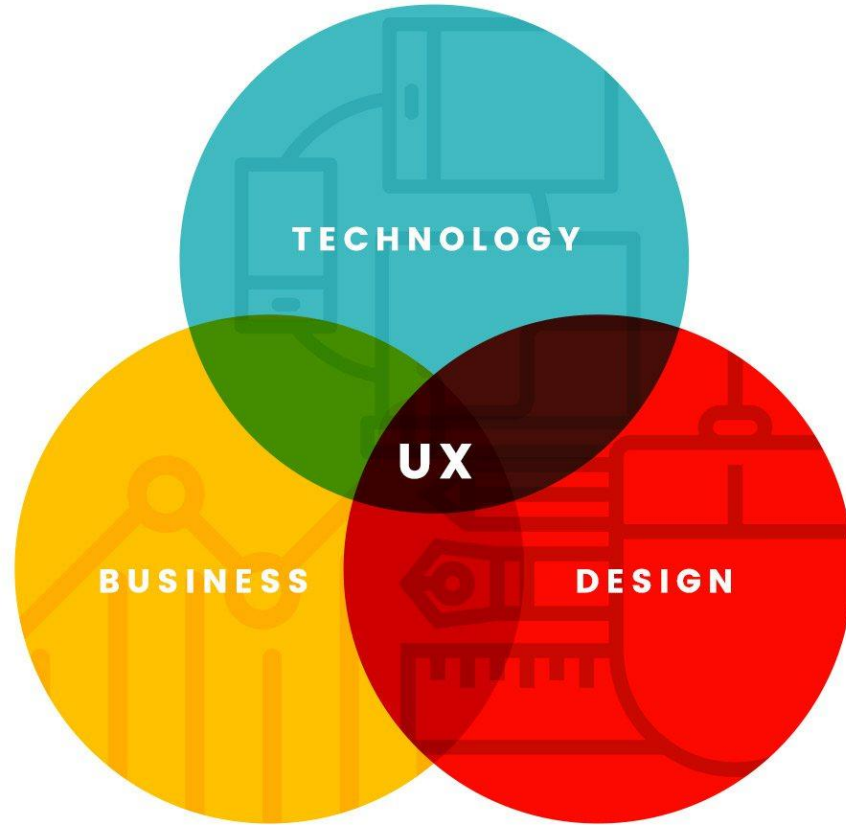
UX Psychology	Universal Design
UX Methods	Accessibility
UX Strategy	User Reserch
Design Process	User Interviews
Design Research	User Journey
Inclusive Design	User Personas
Mental Models	Affinity Diagram
Empathy Mapping	UX Microcopy
Dark Patterns	Gamification
	UX Deliverables

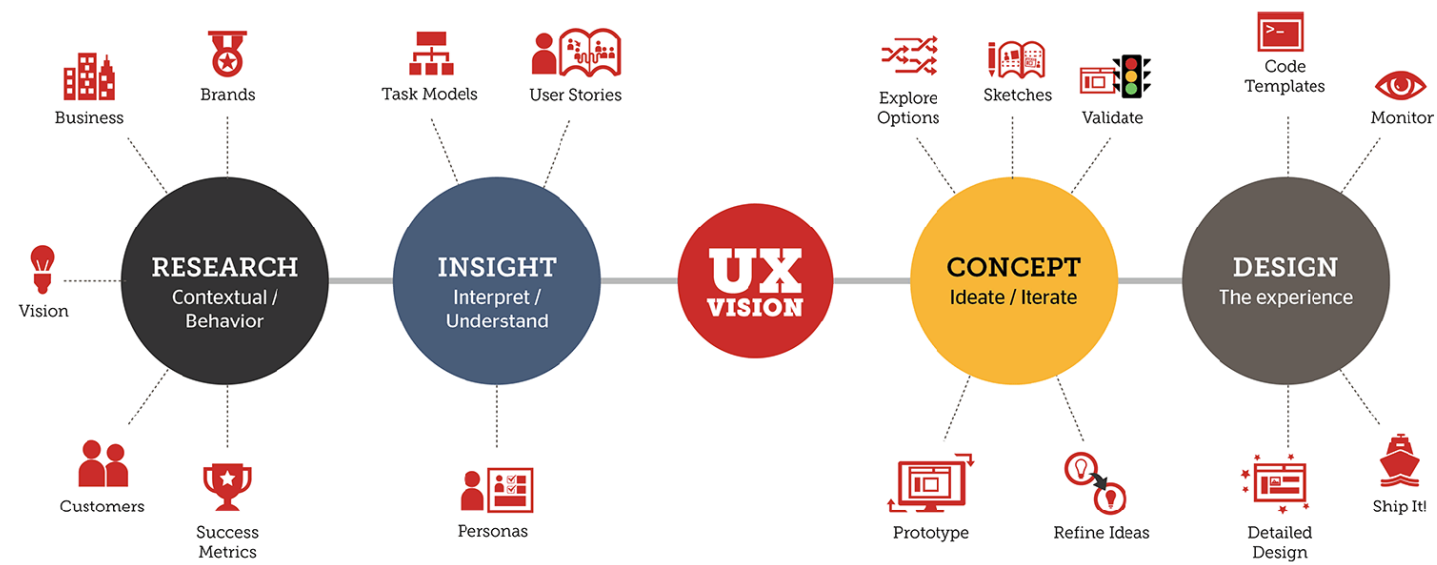
UI

Visual Design	Empty States
Design System	Motion Design
Grids	Prototyping
Information Architecture	App Design
Icons	Interaction Design
Typography	Heuristic Evaluation









When UX Meets CX

Customer Service

Advertising

Brand Reputation

Sales Process

Pricing Fairness

Product Delivery

User Experience

CX

UX

Usability

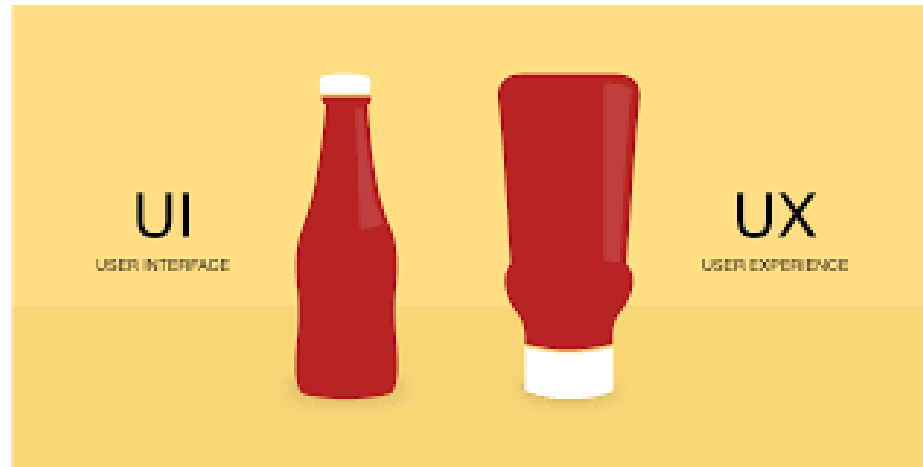
Information Architecture

Interaction Design

Visual Design

Content Strategy

User Research



Overview of Processes and Threads

Difference between Process and Thread

Process:

Process means *any program is in execution*.

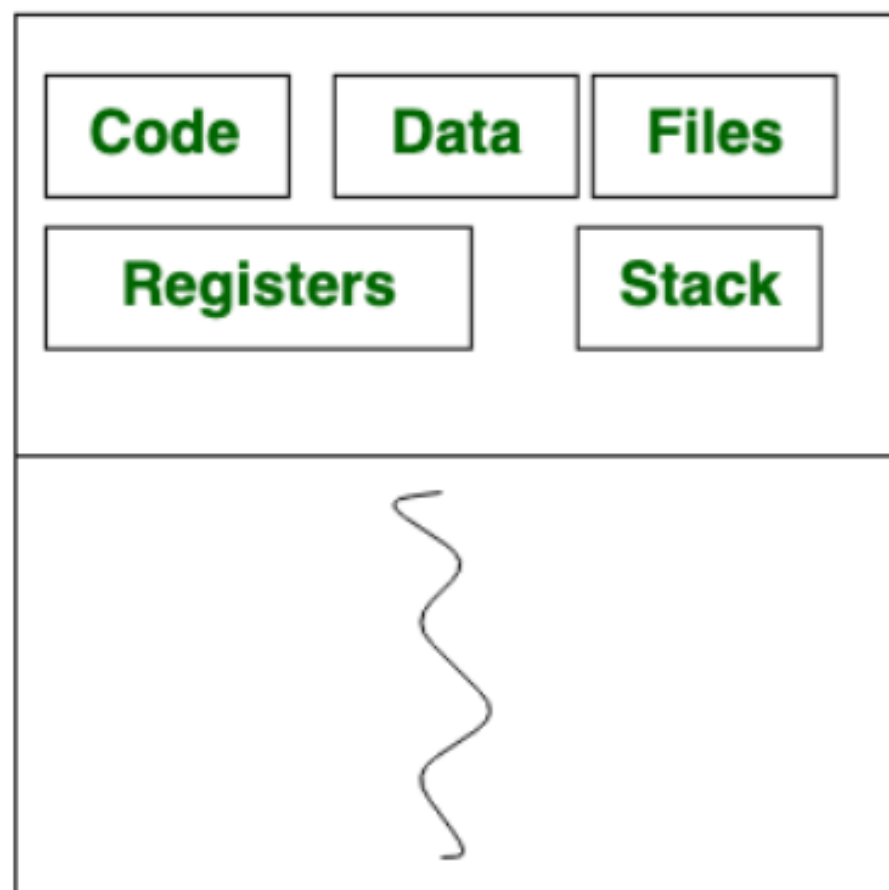
- ❑ **Process control block (PCB)** controls the operation of any process.
- ❑ **Process control block (PCB)** contains the information about processes for example: Process priority, process id, process state, CPU, register etc.
- ❑ A process can create other processes which are known as **Child Processes**.
- ❑ Process takes more time to terminate and it is isolated means **it does not share memory with any other process**.

Thread:

Thread is the segment of a process means a **process** can have **multiple** threads and these multiple threads are contained within a process.

- A thread have **3 states: running, ready, and blocked.**
- Thread takes less time to terminate as compared to process

Process



Thread

□ What is a Thread?

A thread is a **path of execution within a process**. A process can contain **multiple threads**.

- **Why Multithreading?**

A thread is also known as *lightweight process*.

- The idea is to achieve parallelism by dividing a process into multiple threads.
- For example, **in a browser, multiple tabs can be different threads.**
- MS Word uses multiple threads: **one thread to format the text, another thread to process inputs, etc.**

❑ **Process vs Thread?**

The primary difference is that threads within the same process run in a **shared** memory space, while processes run in **separate** memory spaces.

- ❑ Threads are **not independent** of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals).
- ❑ But, like process, a thread has its own program counter (PC), register set, and stack space.

Advantages of Thread over Process

1. *Responsiveness*: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
2. *Faster context switch*: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
3. *Effective utilization of multiprocessor system*: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. *Resource sharing*: Resources like **code, data, and files** can be shared among all threads within a process.

Note: **stack and registers** can't be shared among the threads. Each thread has its own stack and registers.

5. *Communication*: Communication between multiple threads is easier, as the threads **shares common address space**. while in process we have to follow some specific communication technique for communication between two process.

6. *Enhanced throughput of the system*: If a process is divided into multiple threads, and each **thread function is considered as one job**, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Types of Threads

There are **two types** of threads.

- **User Level Thread**
- **Kernel Level Thread**

Difference between Process and Thread:

S.NO	PROCESS	THREAD
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.

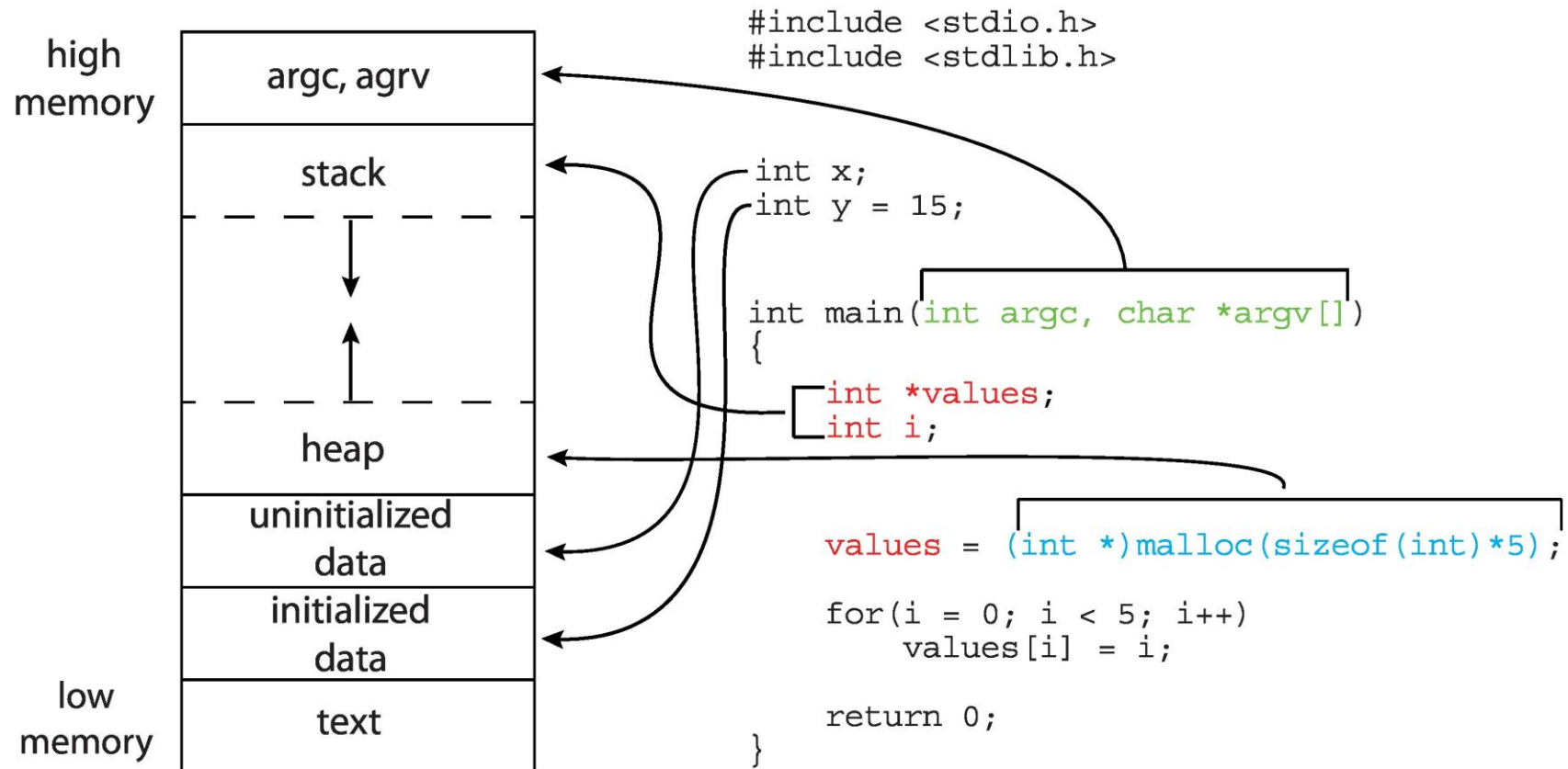
Process Concept

- ❑ An operating system executes a variety of programs that run as a process.
- ❑ **Process** – a program in execution; process execution must progress in **sequential fashion**
- ❑ Multiple parts
 - ❑ The program code, also called **text section**
 - ❑ Current activity including **program counter**, processor registers
 - ❑ **Stack** containing temporary data
 - ▶ Function parameters, return addresses, local variables
 - ❑ **Data section** containing global variables
 - ❑ **Heap** containing memory dynamically allocated during run time

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**); process is ***active***
 - **Program becomes process when executable file loaded into memory**
- **Execution of program started via GUI mouse clicks, command line entry of its name, etc**
- One program can be **several processes**
 - **Consider multiple users executing the same program**

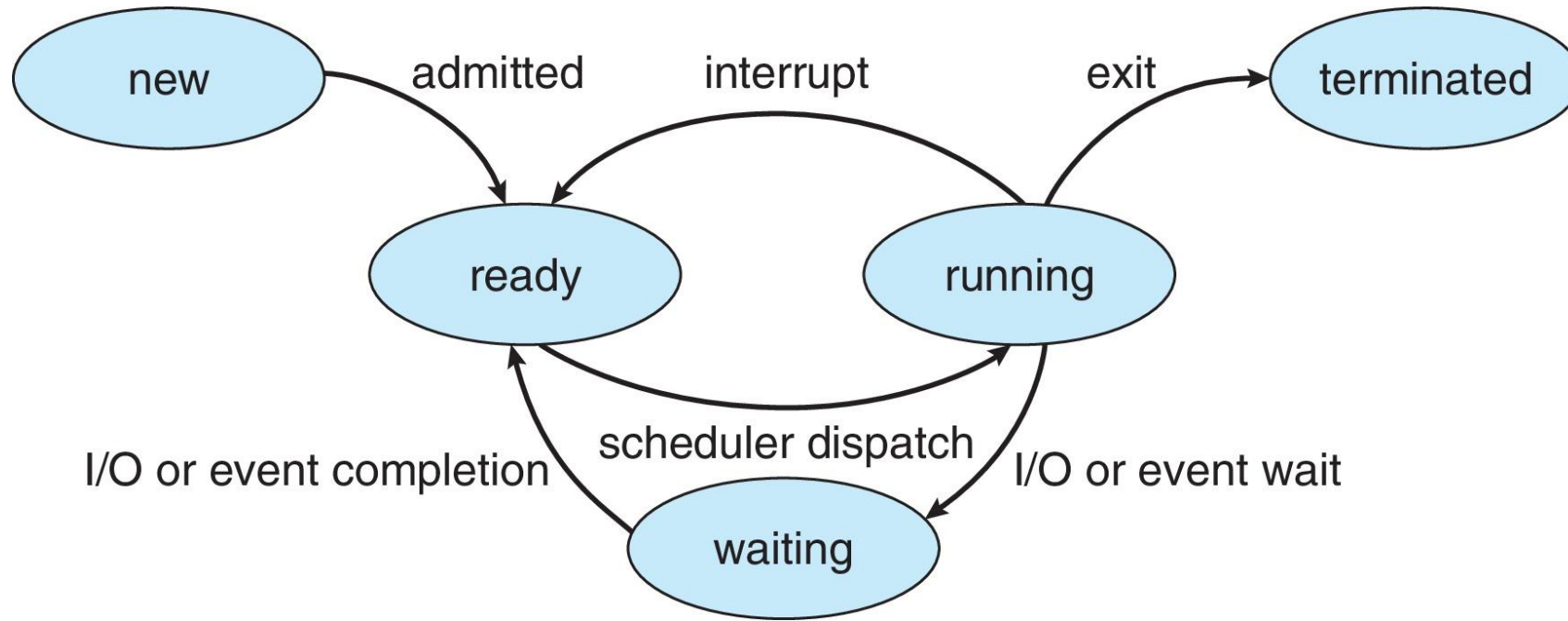
Memory Layout of a C Program



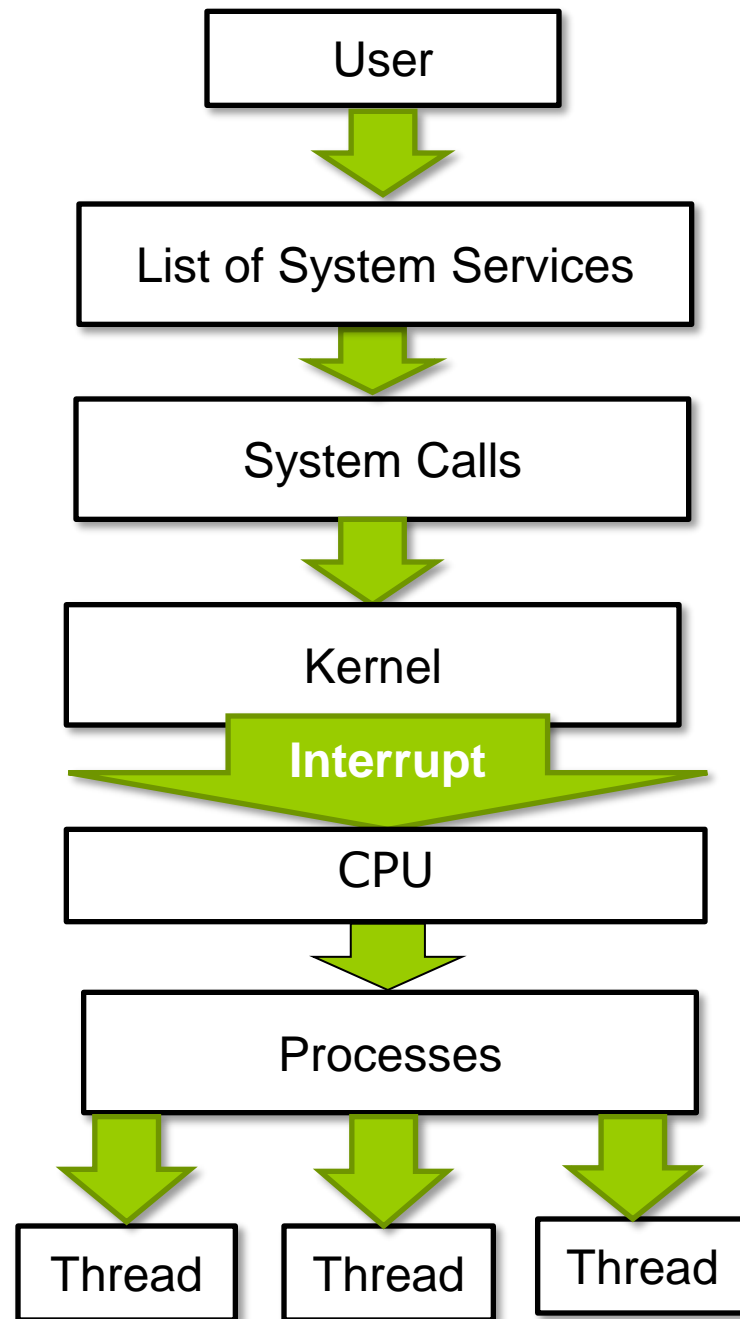
Process State

- As a process executes, it changes **state**
 - **New**: The process is being created
 - **Running**: Instructions are being executed
 - **Waiting**: The process is waiting for some event to occur
 - **Ready**: The process is waiting to be assigned to a processor
 - **Terminated**: The process has finished execution

Diagram of Process State



The Big Pic of OS



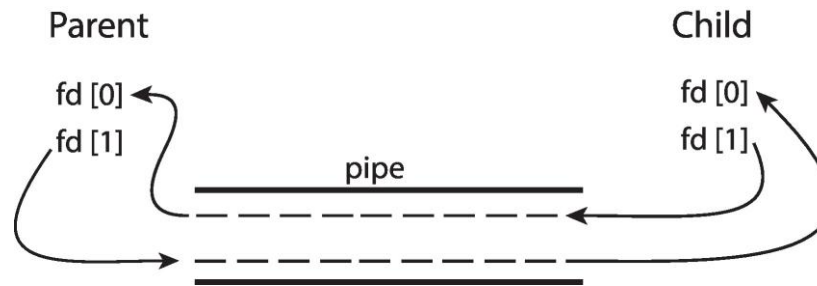
Threads & Concurrency

Pipes

- Acts as a conduit allowing two processes to communicate
- **Ordinary pipes** – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- **Named pipes** – can be accessed without a parent-child relationship.

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these **anonymous pipes**

Named Pipes

- ❑ Named Pipes are more powerful than ordinary pipes
- ❑ Communication is bidirectional
- ❑ No parent-child relationship is necessary between the communicating processes
- ❑ Several processes can use the named pipe for communication
- ❑ Provided on both UNIX and Windows systems

Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls

Sockets

- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are **well known**, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running