Introduction & OS Structures Chapters 1 & 2

I. What is an Operating System?

A. One definition: A Resource Manager

1. What are the resources? Mostly **hardware**
   a) CPUs (and associate co-procs)
   b) Memory (registers, cache, RAM, disk)
   c) Devices (IO, specialized)
   d) Software (its own libraries, “critical” code)
2. How does it manage them? Through **services;** An environment in which to execute programs (Support getting “work” done)
   a) Program Execution (loading, running, monitoring, terminating)
   b) Performance (optimizing resources under constraints)
   c) Correctness (overseeing critical operations, preventing interference)
   d) Fairness (access to and allocation of resources)
   e) Error detection & recovery (network partition & media failure)
   f) Communication (inter-process, software-to-hardware, hardware-to-hardware, system-to-system, wide-area)
   g) IO: reading & writing, support for various mediums, devices, performance, and protections
   h) Data Organization (naming), Services (search) & Protection (access control)
   i) Security (isolation, enforcement, services, authentication, accounting and logging, trust)
   j) User interfaces (command-line, GUIs, multiple users)
3. Each service has associated **challenges** & **tensions**
   a) We’ll see a lot of, “How do we accomplish X?” for each of

the services listed above

b) Example 1: Disk drive performance characteristics; and probabilistic IO requests - How do service them (service them in order, service them by time, can we improve by waiting, for how long)?

c) Example 2: We have limited RAM, and we want to run more programs that can be stored, how do we allocate space (Who stays, who goes, in what amount do we expunge, what if we're wrong, what if the system is under extremely heavy load, is there a way to predict the future?)?

d) Example 3: We have two process (one that produces answers, one that consumes answers); how do they communicate (message passing or shared memory)? How do they synchronize (how to we prevent over-production, over-consumption, context-switching)?

B. It's just a program

1. One of the first programs to run
2. Has the highest privilege
3. Core: kernel -- supporting services (programs, daemons)

C. No common definition, and implementation's and capabilities vary greatly

II. Types of Operating Systems

A. Desktop (single user)

B. Time share/Mainframe (multi user)

C. Mobile

D. Web (Chrome)

E. Real-time & Soft Real-Time (Media)

F. Embedded

G. Virtual Machines

H. **Q:** How do their requirements differ?

III. OS Interfaces: System Calls

A. The OS provides **services** and manages **resources**
   1. From a user's perspective, we want to deal with services (**abstractions** of the hardware beneath);

B. How do we interface with the services? **System Calls**
   1. Give users abstracted, limited, and controlled access to low-level operations
   2. Gives OS ultimate control over these operations
   3. Comes at a cost (context switch, memory overhead, etc.)
   4. It's like a special function: one provided by the kernel for kernel-specific operations

C. **Types** of Calls
   1. Process Control (creating & duplicating (fork), exec, exiting, signal)
   2. Data Manipulation (create, read, write, truncate)
   3. Device Manipulation (ioctl, read, write)
   4. Information & Accounting (getpid, stat)
   5. Communication (pipe, mmap, socket)
   6. Security (chmod, chroot)

D. Standards
   1. POSIX
   2. Win32
   3. Commonly, OSes augment standards

E. System Call Execution
   1. Like a function call, we push arguments onto the stack, then we call into the library that provides the system call
   2. Each system call has a special number, placed into a register
   3. Executes a TRAP instruction (switch to kernel mode)
      a) A logical separation of memory space
   4. Kernel's system call handler is invoked, once done (but may block) may be returned to the process

IV. OS Paradigms: Microkernel vs.

Monolithic

A. Monolithic kernel
   1. Single piece of code in memory
   2. Can be made modular with “modules,” flexibility, customization, support
   3. Modules can be loaded dynamically
   4. Limited “information hiding”

B. Layered System (Dijkstra)
   1. Monolithic layers (rings)
   2. Each inner layer was more privileged; required a TRAP to move down layers
   3. Hardware-enforcement possible

C. Microkernel
   1. All non-essential components removed from the kernel, for modularization, extensibility, isolation, security, and ease of management
   2. A collection of OS services running in user space
   3. Heavy communication costs through message passing (marshaled through the kernel)

V. Virtual Machines

A. A software representation of hardware interfaces (virtualized devices) managed by a hypervisor (thin layer between **host** and **guest**)

B. Advantages
   1. Allows for multiple OSes (execution environments) to be run “at once”
   2. Partition resources: increase utilization (multi core), specialization (software “appliance”)
   3. Security: Isolation & Introspection
   4. Cheaper than duplicative hardware; hardware performance has far outpaced software performance

C. Not a new idea, but continual improvement in hardware (but not an increase in performance requirements) have made it an attractive paradigm again (foundation of cloud computing)