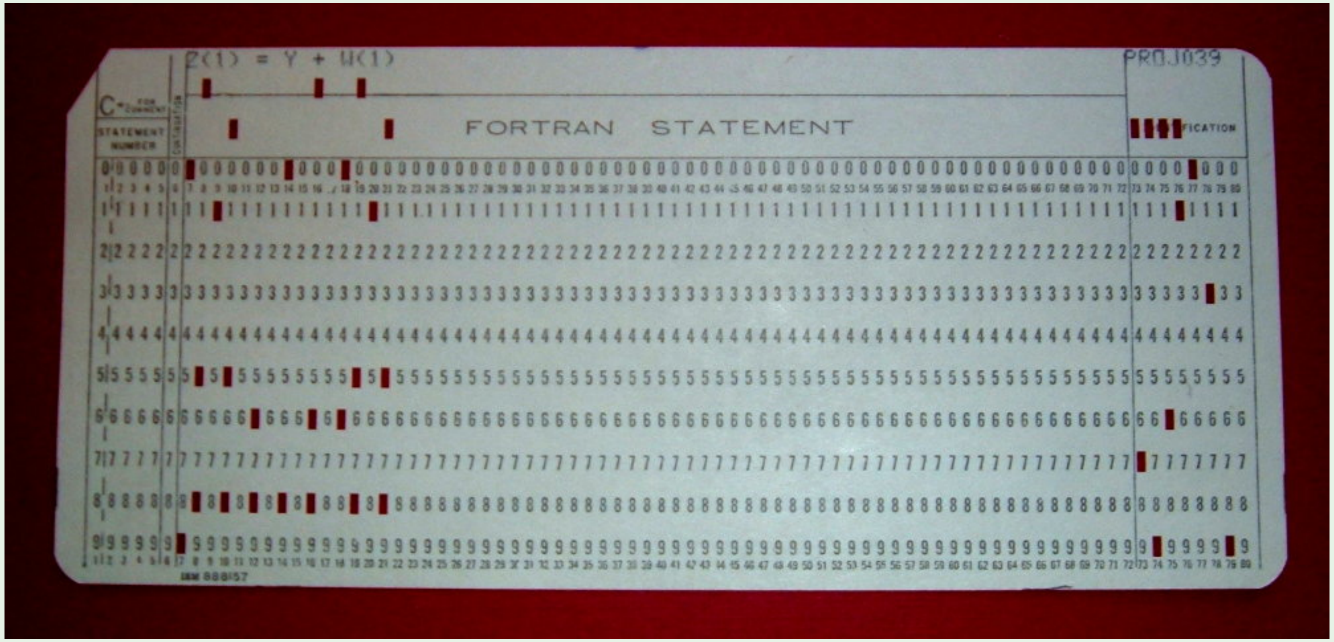
2. An Operating System, What For?

- Operating System Tasks
- Survey of Operating System Principles

Batch Processing

Punched Cards



Is it Enough?

There exist more interactive, complex, dynamic, extensible systems!

They require an *Operating System* (OS)

Operating System Tasks and Principles

Tasks

- Resource management
- Separation
- Communication

$\leftrightarrow$

Principles

- Abstraction
- Security
- Virtualization

2. An Operating System, What For?

- Operating System Tasks
- Survey of Operating System Principles

The Role of the Kernel: Separation, Communication

- The kernel is a *process manager*, not a process
- It runs with higher privileges (enforced by the microprocessor)
  - *User mode*: restricted instructions and access to memory
  - *Kernel mode*: no restriction
- User processes switch to kernel mode when requesting a service provided by the kernel
  - *Context switch*
  - *System call*

The Role of the Kernel: Resource Management

Control

- Bootstrap the whole machine
  Firmware, BIOS, EFI, boot devices, initialization sequence
- Configure I/O devices and low-level controllers
  Memory-mapped I/O, hardware interrupts
- Isolate and report errors or improper use of protected resources
  Kernel vs. user mode, memory protection, processor exceptions

Allocate

- Distribute processing, storage, communications, in time and space
  Process/task, multiprocessing, virtual memory, file system, networking ports
- Multi-user environment
  Session, identification, authorization, monitoring, terminal
- Fair resource use
  Scheduling, priority, resource limits

2. An Operating System, What For?

- Operating System Tasks
- Survey of Operating System Principles

First OS Principle: Abstraction

Goal

- Simplify, standardize
  - Kernel portability over multiple hardware platforms
  - Uniform interaction with devices
  - Facilitate development of device drivers
  - Stable execution environment for the user programs

Main Abstractions

1. Process
2. File and file system
3. Device
4. Virtual memory
5. Naming
6. Synchronization
7. Communication

Process Abstraction

Single Execution Flow

- Process: *execution context of a running program*
- Multiprocessing: *private address space* for each process
  - Address spaces isolation enforced by the kernel and processor (see *virtual memory*)

Multiple Execution Flows

- Within a process, the program "spawns" multiple execution flows operating within the same address space: the *threads*
- Motivation
  - Less information to save/restore with the processor needs to switch from executing one thread to another (see *context switch*)
  - Communication between threads is trivial: shared memory accesses
- Challenge: threads need to *collaborate* when they *concurrently* access data
  - Pitfall: looks simpler than distributed computing, but hard to keep track of data sharing in large multi-threaded programs, and even harder to get the threads to collaborate correctly (non-deterministic behavior, non-reproducible bugs)

File and File System Abstractions

- *File*: *storage* and *naming* in UNIX
- *File System* (FS): repository (specialized database) of files
- Directory tree, absolute and relative pathnames
  `/` `.` `..` `/dev/hda1` `/bin/ls` `/etc/passwd`
- File types
  - Regular file or hard link (file name alias within a single file system)
    `$ ln pathname alias_pathname`
  - Soft link: short file containing a pathname
    `$ ln -s pathname alias_pathname`
  - Directory: list of file names (a.k.a. hard links)
  - Pipe (also called FIFO)
  - Socket (networking)
- Assemble multiple file systems through *mount points*
  Typical example: `/home` `/usr/local` `/proc`
- Common set system calls, independent of the target file system

Device Abstraction

- Device special files
  - *Block*-oriented device: disks, file systems
    `/dev/hda /dev/sdb2 /dev/md1`
  - *Character*-oriented device: serial ports, console terminals, audio
    `/dev/tty0 /dev/pts/0 /dev/usb/hiddev0 /dev/mixer /dev/null`

Virtual Memory Abstraction

- Processes access memory through *virtual addresses*
  - Simulates a large *interval* of memory addresses
  - Expressive and efficient address-space protection and separation
  - Hides kernel and other processes' memory
  - Automatic *translation* to *physical addresses* by the CPU (MMU/TLB circuits)
- *Paging* mechanism
  - Provide a protection mechanism for memory regions, called *pages*
  - The kernel implements a *mapping* of physical pages to virtual ones, different for every process
- *Swap* memory and file system
  - The ability to suspend a process and virtualize its memory allows to store its pages to disk, saving (expensive) RAM for more urgent matters
  - Same mechanism to migrate processes on NUMA multi-processors

Naming Abstraction

- Hard problem in operating systems
  - Processes are separated (logically and physically)
  - Need to access *persistent* and/or *foreign* resources
  - Resource *identification* determines large parts of the programming interface
  - Hard to get it right, general and flexible enough
- Good examples: /-separated filenames and pathnames
  - Uniform across complex directory trees
  - Uniform across multiple devices with *mount points*
  - Extensible with *file links* (a.k.a. aliases)
  - Reused for many other naming purposes: e.g., UNIX sockets, POSIX Inter-Process Communication (IPC)
- Could be better
  - INET addresses, e.g., 129.104.247.5, see the never-ending IPv6 story
  - TCP/UDP network ports
- Bad examples
  - Device numbers (UNIX internal tracking of devices)
  - Older UNIX System V IPC
  - MSDOS (and Windows) device letters (the ugly `C:\`)

Concurrency Abstraction

Synchronization

- Interprocess (or interthread) synchronization interface
  - Waiting for a process status change
  - Waiting for a signal
  - Semaphores (IPC)
  - Reading from or writing to a file (e.g., a pipe)

Communication

- Interprocess communication programming interface
  - Synchronous or asynchronous signal notification
  - Pipe (or FIFO), UNIX Socket
  - Message queue (IPC)
  - Shared memory (IPC)
- OS interface to network communications
  - INET Socket

Second OS Principle: Security

Basic Mechanisms

- Identification
  /etc/passwd and /etc/shadow, sessions (login)
  UID, GID, effective UID, effective GID
- Isolation of processes, memory pages, file systems
- Encryption, signature and key management
- Logging: /var/log and syslogd daemon
- Policies:
  - Defining a security policy
  - Enforcing a security policy

Enhanced Security: Examples

- SELinux: http://www.nsa.gov/selinux/papers/policy-abs.cfm
- Android security model: http://code.google.com/android/devel/security.html

Third OS Principle: Virtualization

"Every problem can be solved with an additional level of indirection"

Third OS Principle: Virtualization

"Every problem can be solved with an additional level of indirection"

Standardization Purposes

- Common, portable interface
- Software engineering benefits (code reuse)
  - Example: Virtual File System (VFS) in Linux = superset API for the features found in all file systems
  - Another example: drivers with SCSI interface emulation (USB mass storage)
- Security and maintenance benefits
  - Better isolation than processes
  - Upgrade the system transparently, robust to partial failures

Third OS Principle: Virtualization

"Every problem can be solved with an additional level of indirection"

Compatibility Purposes

- Binary-level compatibility
  - Processor and full-system virtualization: emulation, binary translation (*subject of the last chapter*)
  - Protocol virtualization: IPv4 on top of IPv6
- API-level compatibility
  - Java: through its virtual machine and SDK
  - POSIX: even Windows has a POSIX compatibility layer
  - Relative binary compatibility across some UNIX flavors (e.g., FreeBSD)