Overview and History of Operating Systems

These are the notes for lecture 1.

Please review the “Syllabus” notes before these.

Overview / Historical Developments

An Operating System...

- **Sits between hardware and users**
- **Provides “environment” to execute programs**
- **Like a government**
  - **No useful work**
  - **Regulates workers**
- **Manages, allocates resources**
  - **CPU (execution time)**
  - **Memory Space**
  - **Disk / File storage**
  - **I/O Devices**
- **Control**
  - **Prevent incorrect use of hardware**
  - **Security / Protection**

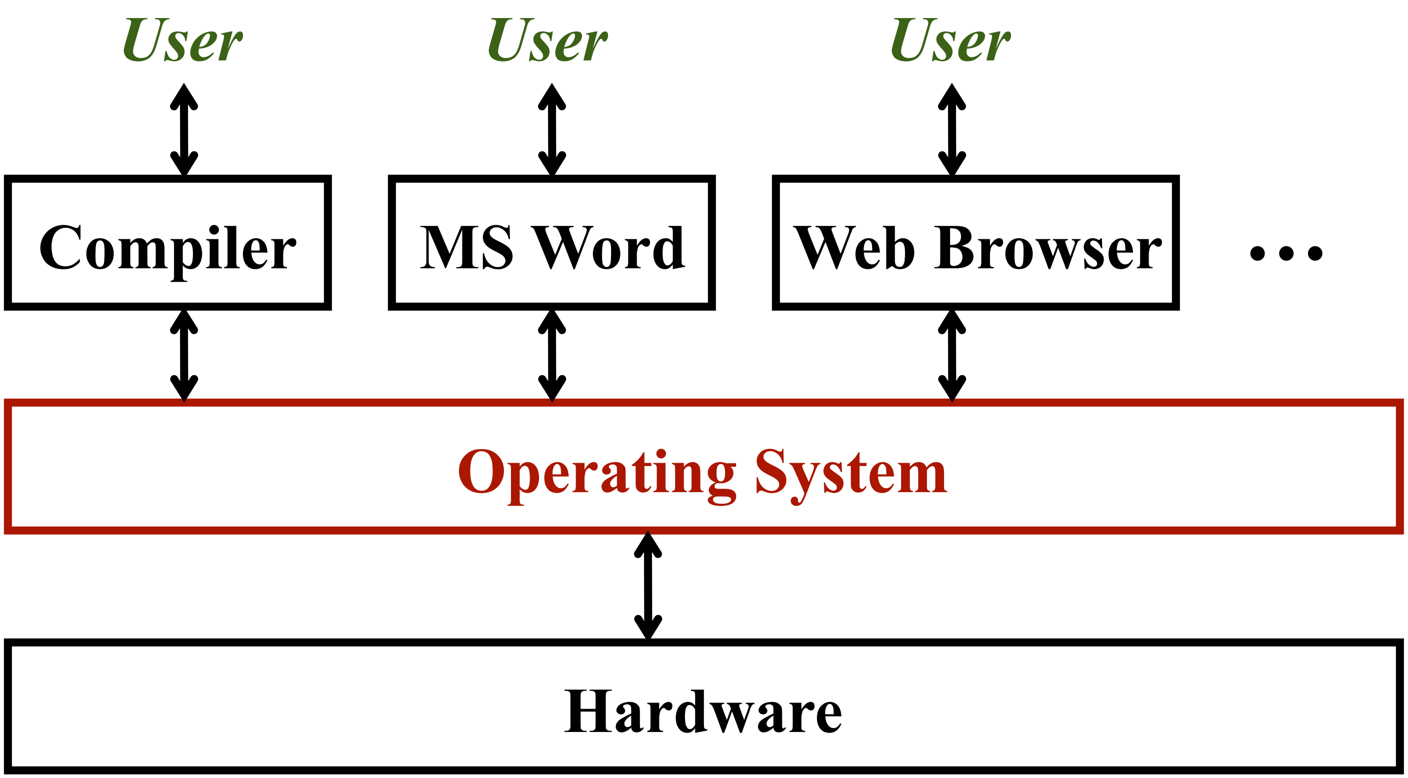
Goals

Make computer *easy* to use
Make computer more *efficient*
Help user *solve problems* / *do work*

Ease of use
Efficiency

} ***Often in conflict***

Overview



User
User
User
Compiler
MS Word
Web Browser
• • •
Operating System
Hardware

Early Computers

Punch card

Input Devices:
- Card Reader

Output Devices:
- Printer
- Card Puncher

(No disk, no secondary storage)

A "*job*"

User prepares input cards (Program, Data)
User gets time on the machine
Loads the program
Executes it
Study the output and come back tomorrow

Early Computers

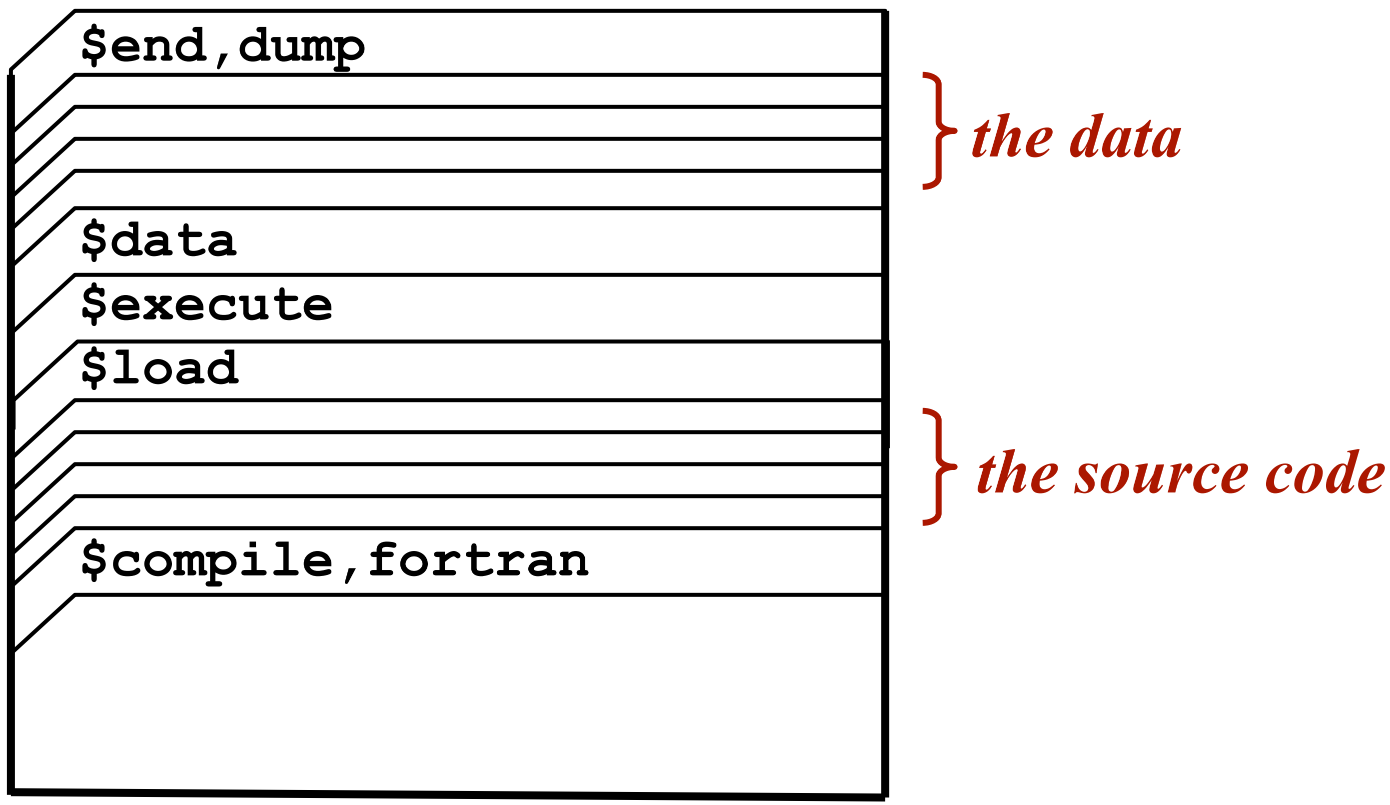
The O.S....

(Simply a "*control program*")

- **Read cards**
- **Load memory**
- **Transfer control to user code**
- **Print out contents of memory ("*core dump*")**
- **Loop to next "*job*"**

Batch Operating System

Read jobs from cards
Some jobs are "control cards"



New Technology: Magnetic Tape

Idea: **Read cards onto tape**

To output a card...

Write to tape & punch it later

Same for printing

New Technology: Magnetic Tape

Idea: **Read cards onto tape**

To output a card...

Write to tape & punch it later

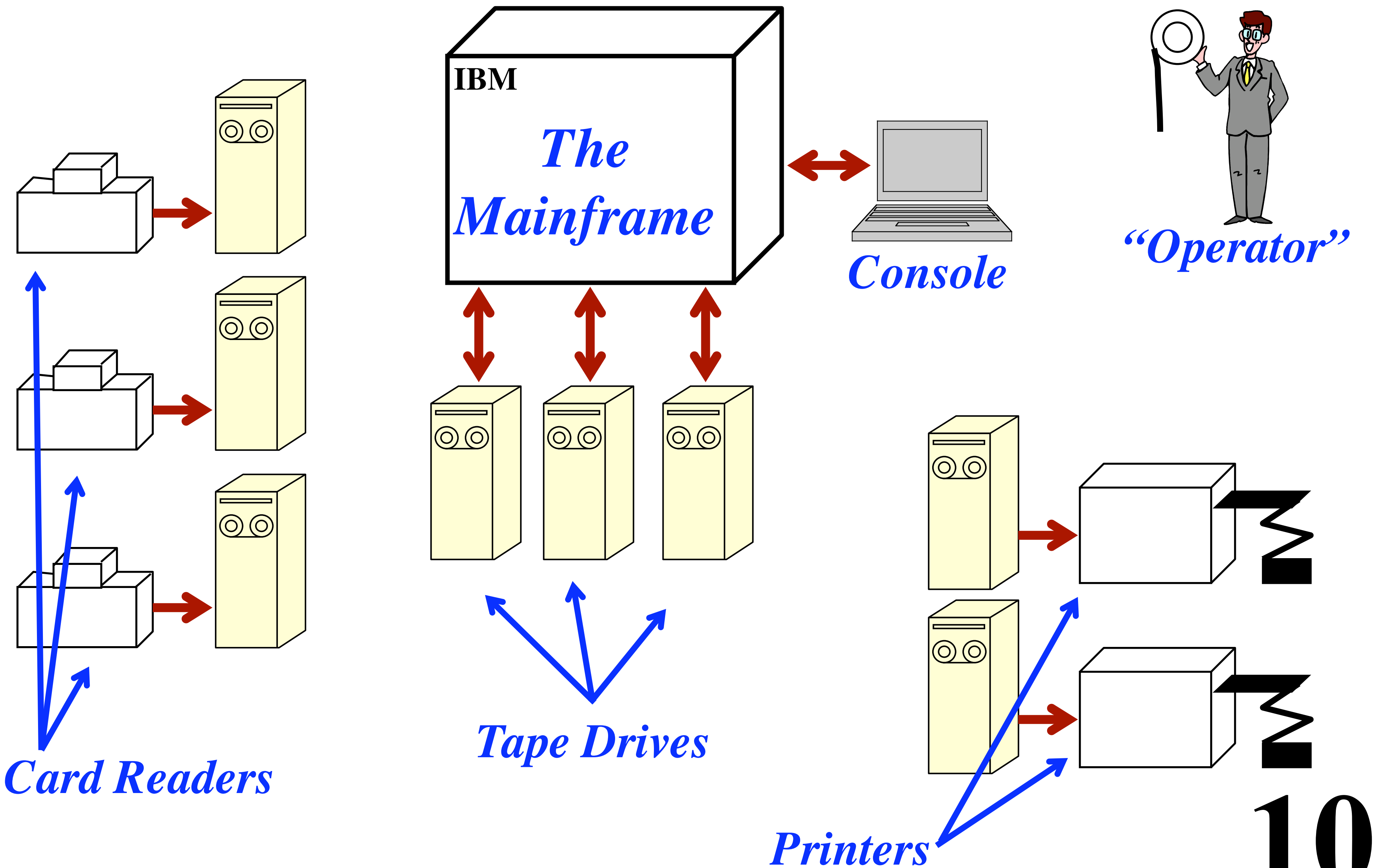
Same for printing

Concept: ***SPOOLING***

- **Use tape as a "buffer".**
- ***Allows I/O from one job to overlap the computation from another job!***

(When disks were invented, they were first used like this.)

A Mainframe System (circa 1960)



New Technology: The Disk

The first disks were used for spooling.
(The “file” was invented later.)

Concept: The Job Pool

Several jobs are waiting to be executed

- **One job in memory**
- **Future jobs sitting on disk**

New Technology: The Disk

The first disks were used for spooling.
(The "file" was invented later.)

Concept: The Job Pool

Several jobs are waiting to be executed

- **One job in memory**
- **Future jobs sitting on disk**

The O.S. can make its first decision!

Which job to run next?

First-come, first served

vs.

Job Scheduling

Multi-programming

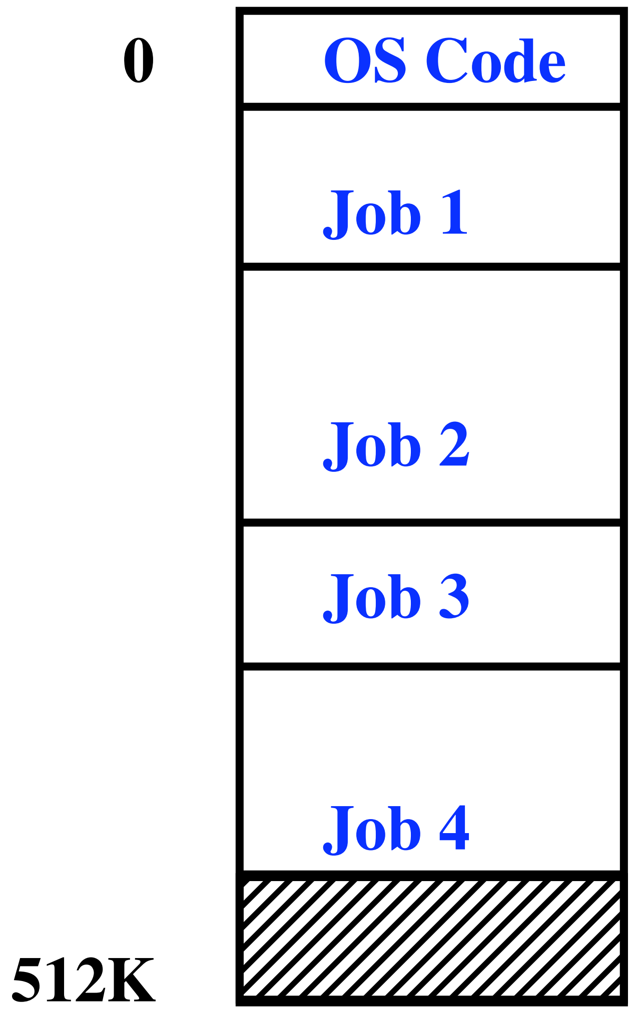
***Idea:* Keep several jobs in memory at once!**

When one job waits on I/O...
another jobs can use the CPU!

Increases CPU utilization

Don't keep entire job pool in memory
(just select a few)
Job X starts I/O...
OS selects another job to run.
CPU does not sit idle.
When job X finishes...
OS selects another job from job pool
and loads it into memory.

Main Memory



Main Memory

Simplest Approach:

- **The entire job is laoded into a contiguous range of memory**
- **No protection between programs**
- **A job runs until it requests I/O**
  No "*time-slicing*"
  If it loops... Oh, well...
- **When complete, it transfers back to the OS.**

Memory Management

A big topic (a chapter in textbook)

Job Scheduling

When job X blocks (due to I/O)...
... which job will be run next?
"Process Management" (chapter 2)

New Concept: Terminals

Combines with multiprogramming!
Users want to interact with the programs
while they are executing!

Initial motivations for interaction:

- **Deal with contingencies during job execution.**
- **Debugging programs.**

The good 'ole days...

- **Long turn-around times.**
- **Program were smaller.**
- **Written in assembly / FORTRAN.**
- **Programmers were very careful.**
- **Bugs were intolerable.**

Batch Operating Systems

Each job runs non-stop
(until it decides to perform I/O)

Problems when used with interactive terminals:

- **User response times are...**
  - **Long**
  - **Unpredictable**
- **Bugs in one program...**
  - **Crash the entire system**
  - **Really annoyed people using terminals**
  - **(With batch jobs, just restart the programs)**

New Concept: Time-Sharing

"Multi-tasking"

Goal:

Make it seem like every user has a dedicated computer!

The Idea:

Switch the CPU rapidly between jobs so that every running job seems to be making regular progress.

New hardware support required:

Periodic "timer interrupts"

Ways to protect one program from the other programs.

Goal: isolate bugs / loops to just that program!

Historical Context

Each user has a Cathode Ray Tube (CRT) terminal.
Each user types a command & it is executed.
“Response time” -- should be < 1 second.
Many users “online”, sharing the CPU.
Demonstrated in 1960, common in 1970s.
Files kept on disks...
but lots of implementation details still visible.
(blocking, file formats, sectors, etc...)

UNIX (1969, spreading in mid-1970’s)

Time-Sharing

Several processes in memory.
Memory management & protection are required.
If size of user job is really large...
- **Swap other users' jobs out to disk (temporarily)**
- ***"Virtual Memory"***
  - **Don't load the entire job into memory**

Disk Management:
- **Directory Structures**
- **Ease-of-use**
- **Protection from other users**

Communication between users:
- **Still an active area**
- **Synchronization, coordination**

Personal Computers

Began appearing in the 1970s.

Shift of emphasis:

- **Hardware utilization is less important**
- **Maximize: User convenience**
  Fast response time
  Ease of programming
- **Security vs. Communication**

Early opinions: Protection is unimportant
Each person's computer is separate & isolated.

Trends

Hardware costs will continue to fall.

OS's will appear embedded in more devices!
More kinds of OS's will be needed!

Computer users will become more sophisticated.

Better OS's will be demanded!

Features on research / high-end systems will become common on small, inexpensive systems.

Parallel processing / multi-processing
Real-time control
Distributed systems

Programs will become more complex.

The OS will need to promote ease of programming / use!

More malicious programs will be created.

More security will be needed!

Kinds of Operating Systems

Personal Computers
Large, super-computers
  Parallel multi-processors
Embedded computers
Real-time computers
Distributed systems
Highly reliable systems
Super-low cost
Research platforms

Applications

- Chess playing
- Dishwasher / microwave
- Flight control / space shuttle
- Military command
- Automobile control
- Lab / factory automation
- Assembly robot control
- Corporation management
- Web server
- Web search engine
- Nuclear reactor control
- Toys
- Artificial Intelligence

... etc ...

Parallel Systems

Single processor vs. multi-processor systems

Goals:

- **Increase *"Throughput"***
  Get more work done, per hour
- **Utilize small, inexpensive processors**
  ... To get more horsepower (giga-flops)
  Example: Graphics co-processor
- **Save money by sharing expensive peripheral devices.**
- **Reliability**
  Graceful degradation
  Fault-tolerance

Example: Tandem System

Two identical processors

- **Primary**
- **Backup**

Each has its own memory

Operating in lock-step

Failure detected?

Backup becomes the primary

"hot backup"

Types of Multiprocessing Systems

Tightly-Coupled Systems

- **Share the system bus**
- **Share peripheral I/O devices**
- **Share memory (sometimes)**
- **Share a common clock (sometimes)**

Example: Graphics Co-processor

Lossely-Coupled Systems

Types of Tightly-Coupled Systems

"Symmetric Multiprocessing"

- **Each processor runs identical copy of OS**
- **OS code resides in shared memory**
- **Shared data structures (for concurrency control)**

"Asymmetric Multiprocessing"

- **Master-slave relationship**
- **One processor assigns tasks to others**
- **Increased specialization**
  ---> decreased reliability
- **Trend: cheap processors**
  Offload tasks to slaves or back-ends
    Graphics processor
    Disk processor
    Processor in keyboard / mouse
- **Communication/interfacing becomes paramount.**

Distributed Systems

Processors do not share
- Memory, Clock, System bus, Devices

"Loosely-coupled systems"

Communication
- Slow (internet) ⟷ Fast (specialized bus)

Motivations
- Communication & sharing
  - Email, shared data, group work
- Reliability
  - No single failure should crash the entire system
  - Data should remain accessible
- Computation speed-up
  - Load-sharing
- Recource sharing
  - Access to specialized or unique hardware

Real-Time Operating Systems

Used to control a physical process
Sensors collect data ("input"?)
Actuators control the physical process ("output"?)

Examples:

- **Avionics (space shuttle control system)**
- **Medical systems (heart monitor)**
- **Industrial system (oil refinery)**
- **Military (anti-missile laser control)**
- **Consumer products (automobile controller)**

The key...
Rigid, well-defined, fixed time contraints

Time-sharing systems *should* repond quickly!
Real-time systems *must* respond quickly!

Real-Time Operating Systems

Soft Real-Time Systems

Some processes are given higher "*priorities*."
Example: video & music playback

Common in many OS's.

Adequate for many applications,
... but too risky for some!

Even though response is often fast enough,
it is never guaranteed!

Real-Time Operating Systems

Hard Real-Time Systems

Guarantees that task will complete by their deadlines.

All potential delays are *bounded*.
Want to avoid:

- **Disks (highly variable latencies)**
- **Virtual Memory (complex & unpredictable)**

OS tends to be low-level, minimal, close to hardware.

A specialized sub-field of OS.