

# 15-410

*“Anything else is just the rest of your life”*

## Operating System Overview Jan. 16, 2004

**Dave Eckhardt**

**Bruce Maggs**

# Synchronization

## Syllabus!

- Please read the syllabus!

## Project 0

- Questions?
- Please don't forget about the bboard

# Synchronization

## Reading

- Today – Chapter 1, more or less
- Upcoming
  - Chapter 4 (Process) – Skip 4.5, 4.6
  - Chapter 5 (Thread)
  - Chapter 7 (Synchronization) – Skip 7.9

# Outline

## What is an OS?

- “A home for a process”
- Brief history
- Special topics for special hardware

# What is an OS?

## **PalmOS**

- 1 user, 1 task

## **IBM VM/CMS**

- 1000 users, 1 (DOS box) task apiece

## **Capability-based OS**

- What do you mean by “user”?

# What is an OS?

## Size

- 16 kilobytes?
- 16 megabytes?

## Portable:

- “Of course!!!”
- “Why would you want portability???”

## Consensus elusive

- “The stuff between the hardware and the application”

# Common Features

## Abstraction layer

- People want files, not sectors
- People want I/O, not interrupts
- People want date & time, not "ticks since boot"
- Or: *Obstruction* layer
  - See: Exokernel

# Common Features

## Virtualization

- Give everybody “their own” machine
- IBM's VM/SP is “strong” virtualization
  - Your own 3081!
  - PC-XT/370!
- Unix process is like a virtual machine too
  - Upcoming lecture



# Common Features

## Protected Sharing (*Controlled* Interference)

- Shared disk
  - space-sliced
- Shared CPU
  - time-sliced
- Shared keyboard/display
  - Hmm...
- Shared memory
  - Hmm...

# Single-process OS

## Examples

- DEC's RT-11
  - moment of silence
- CP/M (and its clone, MS-DOS)
- Apple DOS
- UCSD p-system

# Single-process OS

## Typical features

- One active program
- Some memory management
- A "file system"
- A command interpreter
  - “Built-in” commands
    - DIR, SET, ^C
  - “External” commands
    - compiler, editor

# Mainframe “Batch” OS

## Examples

- IBM HASP?

## Typical features

- One active program
- I/O library
  - Card reader, tape drive, printer
- Load next program
  - (completion or “abend”)

## Wasteful

- Usually much of machine is idle

# Multiprogramming Batch OS

## Key insight

- Sometimes *two* programs fit in memory
- Each program is often waiting for I/O
- Two for the price of one!

# Multiprogramming Batch OS

## Typical features

- **Job scheduling**
  - Semi-ordered entry to memory
  - No longer a hot research topic
- **Processor scheduling**
  - Multiplexing CPU somehow
- **Input/Output stream abstraction**
  - Virtual card reader/punch
  - JCL!

# Multiprogramming Batch OS

## Typical features

- Memory mapping or linkage discipline
- (Hopefully) crash isolation

## Examples

- IBM MVT, MVS

# Timesharing

## Key Insight

- (none)

## Timesharing = *Interactive* Multiprogramming

- Memory cheap enough for lots of processes
- Terminals cheap enough for lots of users



# Timesharing

## Examples

- CTS, ITS, TENEX
- VM/CMS
- MVS/TSO
- Multics
- Unix

# Timesharing

## Typical features

- Swapping processes out of memory
- Virtual memory
- Fancy process scheduling (priorities, ...)

## Inter-user/inter-process *communication!*

- Why not? You're all logged in all day...

# Shared-memory Multiprocessors

## Requirements

- cheap processors
- shared memory with some coherence

## Advantages

- Throughput
  - linear if you're lucky
- Resource sharing efficiency (one box, one net port)
  - but maybe: resource hot-spot inefficiency
- Machine can keep running if one processor dies

# Asymmetric Multiprocessing

## Typical

- One processor runs the OS kernel
- Other processors run user tasks

## Cheap hack

- Easy to adapt a 1-processor OS

## Downside

- Kernel is a “hot spot”
  - Eventually that processor is 100% busy
  - Then more processors can't increase user throughput

# Symmetric Multiprocessing

**“Ideal” multiprocessing**

**Re-entrant multi-threaded kernel**

**Fascinating problems**

- **TLB shoot-downs**

# Distributed Applications

## Concept

- Modeling from one mountain peak to another
- Standage, Victorian Internet

## Client-server

- WWW
- File service

# Distributed Applications

## Message passing / “Peer-to-peer”

- e-mail
- USENET
- Music/movie “sharing”
- “Ad-hoc networking”
- “Sensor” nets

# Loosely-Coupled Distributed Applications

## Sample Challenges

- Time delays may be large
  - Vinge, [Fire Upon the Deep](#)
  - Clarke, [Songs of Distant Earth](#)
- Group membership generally un-knowable
- Temporal coherence often very weak
- Messages must be somewhat self-contained
- No authority to trust



# Loosely-Coupled Distributed Applications

## Advantages

- Large systems can grow with minimal central planning
- Large, *useful* systems
  - e-mail, USENET, WWW
- Aggregate throughput can be enormous
- Systems can keep working despite damage
  - “The Net interprets censorship as damage and routes around it” – John Gilmore

# Distributed File Systems

## Typical features

- **Single global namespace**
  - Everybody agrees on mapping between files & names
- **Many servers, but invisible**
  - Server name not part of file name
  - File motion among servers is transparent
- **Authentication across administrative boundaries**
- **Some client autonomy**
  - Avoid server hot spots

# Distributed File Systems

## Examples

- AFS
- OpenAFS
- Arla
- Coda

## “Storage” is hot

- NAS, SAN
- So maybe the time has come

# Distributed Operating Systems

## Intuition

- Mixture of remote and local resources

## Interactive process

- Local memory, processor, display, keyboard, mouse
- Remote file system

## Server process

- Local memory, processor (maybe disk)

# Distributed Operating Systems

## Examples

- Locus
- Amoeba
- Sprite
- Plan 9
- ~Mach

# Distributed Operating Systems

## Common emphases

- “Capabilities” for objects
  - Same format, behavior for remote or local objects
  - (non-forgable handles require cryptography)
- User-centric namespaces
  - My `"/tmp"` is *mine*

## *One* namespace:

- files, processes, memory, devices

# Real-time Systems

## Sometimes time matters

- Music
  - “small” glitches sound *bad*
- Gaming
  - must match hand/eye coordination
- Factory process control
- Avionics

# Real-time Systems

## Hard real-time

- Glitch means something goes *boom*
- Avoid things with unpredictable timing
  - Virtual memory, disks
- Seriously over-engineer

## Soft real-time

- Ok to do it right “most of the time”
- Minor changes to existing OS help a lot
- Fancy scheduler, fancy mutexes, memory locking



# Mobile Computing

## Examples

- PDAs
- Laptops
- “Sensor” networks

## Standard resources are tight

- Memory
- Processor speed
- Screen size

# Mobile Computing

## New worries

- Intermittent connectivity
- Self-organization
- *Power*

# Summary - 1

## Resource abstraction

- **Packets  $\Rightarrow$  reliable byte streams**
- **Disk sectors  $\Rightarrow$  files**
- **Resource naming**

# Summary - 2

## Resource sharing/protection

- CPU time slicing
- Memory swapping/paging
- Disk quotas

# Summary - 3

## Communication & Synchronization

- Messaging
- Synchronizing & coherence

# Closing

## Upcoming

- Hardware (in one easy lecture!)
- The Process