OS Overview

David A. Eckhardt
School of Computer Science
Carnegie Mellon University

de0u@andrew.cmu.edu

Administrivia

"Book reports"

- Suggestions: de0u+412books@andrew.cmu.edu

Project 1

- Expect: individual
- Probably: out Monday (for two weeks)

PC Survey

- Who would prefer to work *primarily* on non-128.2 machine?

Office Hours, Syllabus

- The web site should be less vacuous Thursday or Friday

A word about C++/...

- The word is "tilt!"

Outline

“OS Concepts”, Ch. 1

What is an OS?

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

Next time

- Hardware, meet Software
    - Including one “important conceptual contribution”

What is an OS?

Consensus elusive

- PalmOS: 1 user, 1 task
- IBM VM/CMS: 1000 users, 1 (DOS box) task apiece
- Capability-based OS: user?
- Size: 16 kilobytes? 16 megabytes?
- Portable: “yes!” (or: “why?”)

...but I know one when I see one!

- (like other obscenities)

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”

Virtualization

- Give everybody “their own” machine
- VM/SP is “strong” virtualization
- Unix process is not far off

Protected Sharing (*Controlled* Interference)

- Shared keyboard/display I/O
- Shared memory

Single-process OS

Examples

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

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 on completion or abort

But...

- Wasteful: often 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!

OS requirements

- 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!
- 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

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/process *communication*!

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

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: only one processor runs the OS kernel
- other processors run user tasks
- cheap hack: easy to adapt a 1-processor OS
- lose: kernel is a "hot spot"

Symmetric Multiprocessing

- re-entrant multi-threaded kernel

Distributed Applications

Concept

- Yodeling from one mountain peak to another

Client-server

- WWW

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
- “messages” must be somewhat self-contained
- temporal coherence often very weak
- no authority to trust

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

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)

Examples

- AFS
- OpenAFS, Arla

“Storage” is hot

- 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)
- Amoeba, Plan 9, ~Mach

Common emphases

- “capabilities” for objects (remote or local)
    - (non-forgeable 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

"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

New worries

- intermittent connectivity
- self-organization
- *power*

Summary

Resource abstraction

- packets -> reliable byte streams
- disk sectors -> files
- resource naming

Resource Sharing/protection

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

Communication & Synchronization

- Messaging
- Synchronizing & coherence