Course Introduction

CS 239 — Advanced Operating Systems

Spring 1995

CS 239 - Advanced Operating Systems

Pre-requisites - CS 111

Schedule of class

Office hours

E-mail: reiher@cs.ucla.edu

Grading

Mid-term - 25%

Research paper - 25%

Final exam - 50%

Class Outline

- Review of basic OS concepts
- File Systems
- IPC Mechanisms
- OS Security
- Distributed Operating Systems
- OS Organizations
- New Topics in Operating Systems

Review of OS Concepts

- Overview
- Interrupts
- Basic OS Organization
- Processes and Scheduling
- Synchronization
- Virtual Memory
- Basic File Systems

Overview

Purposes of OS—

- Clean virtual machine
- Architecture independence
- Resource sharing and utilization management
- Long term data storage
- Protection, security, accounting
- Real time support, parallelism, human interface

Changing Roles of OS

What it does depends on available HW and SW

And on changing uses of machines

And on changing expectations of users

OS Structural Issues

- Interactivity?
- Multiple users?
- Multiple concurrent processes?
- Multiple processors?
- Multiple machines?
- Virtual memory?
- Real time?

Interrupts

Hardware ability to unexpectedly change current path of execution

Generally based on either

- An internal processor event
- An external stimulus

OS must determine what to do when an interrupt occurs

Purpose of Interrupts

To deal with high priority activities that cannot wait

To handle real time constraints

To maintain control over processor usage

To recover from errors

Interrupts and Context Switches

Interrupts usually lead to context switches

An entirely different piece of code starts running

For an entirely different purpose

Old context may or may not be returned to

Kinds of Interrupts

- Trap instructions
- Software errors or exceptions
- Hardware failures
- I/O interrupts
- Timers
- Page faults

Handling an Interrupt

Save processor state

Consult interrupt vector table

Transfer control to interrupt handler

Implications of Interrupts

Asynchrony

Effective parallelism

Non-determinism

Real time management possible

Organization of an OS

Components include:

- OS kernel
- Interrupt vectors
- Bootstrap program
- Device drivers
- Various servers

Major Divisions of a Basic OS

- Program execution and process management
- File system management and device support
- Network and communications management
- User interface support

OS Organization Strategies

- Monolithic organization
- Layered organization
- Virtual machine systems
- Microkernel systems
- Object oriented systems

Processes

What is a process?

Unit of processor scheduling

Sequential activity

Usually in control of private address space

Components of a Process

Process control block

Address space

Process Control Blocks

Contains all information about process OS needs to know

- Process name and type
- Process state image
- Scheduling info
- Address space location
- Process handling information
- Lots of other stuff

Process Address Space

A physical or virtual segment or segments of memory

Protected against access by other processes

May be stored in core or on disk

Process States

- Running
- Ready
- Blocked
- Swapped out

Typical Process Flow

Ready process reaches head of queue and starts running

Process runs till end of time slice

Process is switched out and moved to the end of queue

Alternately, process blocks and is moved to end of queue

Or process exits and is destroyed

Process Scheduling

How to handle needs of simultaneously active processes?

Need to provide —

- Fairness
- Responsiveness
- Effective utilization of resources

Every so often switch from one process to another

When to switch?

- When time slice used up
- When running process blocks for some reason
- When interrupt occurs
- When running process encounters a fault

Context Switching

Effectively, changing the attention of the processor

Requires—

- Stopping running process cleanly
- Saving context of stopped process
- Loading context of next process
- Starting up next process
- Accounting and scheduling administration

Scheduling Algorithms

- Round robin
- Preemptive priority
- Aging preemptive priority
- Shortest deadline first
- Shortest remaining time first
- Multilevel feedback

Synchronization

Real systems are non-deterministic

Due to concurrency

But we must hide much non-determinism from users

Synchronization used to provide determinism where important

Guarantees correct program execution regardless of interleaving

Sources of Non-determinism

- Device timing
- Code execution timing
- Effects of multiple active processes
- User interactions

Cobegin/Coend

Statements bracketed by cobegin/coend execute concurrently

None begin before statements preceding cobegin finish

No statements after coend start until bracketed statements end

Cobegin/Coend Example

```
x = 0;
cobegin
x = x+1;
y = x;
coend;
z = x +1;
```

Atomic Actions

Sequence of instructions

that acts as a single, uninterruptable instruction

At any instant, all actions have been performed

Or none have

Mutual Exclusion

No two processes in critical section of code simultaneously

Can be used to provide atomic actions

Can be implemented in various ways

Semaphores

Comprehensive synchronization mechanism

Can provide mutual exclusion

Three operations on semaphores:

- Create and Initialize(s,n)
- P(s)
- V(s)

P Semaphore Operation

Grants control of resource protected by semaphore

Or blocks waiting for current controller to finish

V Semaphore Operation

Releases control of protected resource

And wakes one waiting process, if any are waiting

Uses of Semaphores

Can provide many synchronization services

- Mutual exclusion
- Message rendezvous
- Coend
- Reader/writer on shared data

Deadlock

Set of processes wait on resources locked by each other

Can wait for—

- Critical code sections
- Resources

Deadlock Example

Dealing With the Deadlock Problem

- Ignore the problem
- Prevent them from occurring
- Detect them and recover from them

Ignoring Deadlocks

Often low-probability

Cost of preventing or detecting too high

Resolve by extra-system means

- Users kill processes, reboot, etc.

Preventing Deadlocks

Not always possible

Often expensive when possible

Generally overly conservative

Detecting and Solving Deadlocks

Not always possible

But possible for some important cases

Generally by detecting circularity in dependency graphs

Tradeoff—checking frequency vs. cost of mechanism

Memory Management

What if you don't have enough real memory to support processes' needs?

Process swapping

- But that doesn't work for big processes

- And it's slow

Virtual Memory

Allow processes to have large memory space

Divide main memory up into *page frames* of a fixed size

Divide processes' memory up into *pages*

  The same size as page frames

Only keep certain pages in main memory page frames

Keep other pages on disk

Page Faults

What happens when a process asks for a page that isn't in main memory?

Generate a page fault that:

- Halts the process
- Generates a request to bring page from disk
- Swaps in another process

Page Faults, Continued

When page request from disk is ready

- Halts running process
- Chooses where to put the page

  (possibly swapping a page out)
- Clears the faulted process
- Starts some process running

How Do You Do It?

Every address reference may generate a page fault

Pages may sit anywhere in physical memory

Need hardware to translate virtual address to real address

Must be very fast

Page Tables

List of physical locations for virtual pages

Typically, each process has its own page table

Kept in core or in special hardware

Translation Lookaside Buffer

Cache of page table entries

Typically associative memory

Ideally, all page requests satisfied here

Flushed on process switch

Typically LRU

TLB and Hit Ratio

Hit Ratio is the fundamental measure of TLB effectiveness

Fraction of memory accesses satisfied by TLB

Page Replacement Algorithms

When you get a page fault, what page gets thrown out?

Many algorithms—

- FIFO
- Optimal
- Least Frequently Used
- Least Recently Used
- Not Recently Used

Locality

Virtual memory can only work because of *locality*

References tend to be clustered in small number of pages

Over short periods of time

Most data access turns out to have good locality

But not all

Page Allocation Policies

How many page frames should a process get?

Too few and it faults all the time

Too many and other processes fault all the time

Processes with poorer locality need more frames

Working Set Theory

WS(w,t)—

at time t, set of pages accessed in last w memory references

w is window size; should be a multiple of time slice

Regularly measure process' WS

Increase or decrease allocation to match

Files and File Systems

Files are persistent collections of data

Organized into *file systems*

File systems provide generic way to access persistant data

File System Fundamentals

File systems should provide—

- Data organization mechanisms
- Data naming mechanisms
- Data access primitives
- Synchronization
- Protection and security

File Systems and Disks

Most typically, permanent file storage is on disk

FS must decide disk storage location for each data page

FS data page need not be same size as VM page

Must be able to find data later

Complex issues to achieve good performance

Directories

Typically, file systems are hierarchically organized

Non-directories are leaves in the tree

Non-leaves are called *directories*

Directories contain organizational information

And help file system locate data

Directories are also files, of course