Computer Systems II
Gordon College

Operating System Overview

Class Intro

• Operating System Class
• Two Directions:
  – Practical
    • Linux+ Guide to Linux Certification and Lab Manual
    • Lab Experience
  – Theoretical
    • Operating Systems (3rd Edition Gary Nutt)
    • Lecture and Projects
Why Study Operating Systems?

- Understand the *model of operation*
  - Easier to see how to use the system
  - Enables you to write *efficient* code
- Learn to design an OS
- Even so, OS is pure overhead of real work
- Application programs have the real value to person who buys the computer

Perspectives of the Computer

(a) End User View
(b) Application Programmer View
(c) OS Programmer View
System Software

- Independent of individual applications, but common to all of them

- Examples
  - C library functions
  - A window system
  - A database management system
  - Resource management functions
  - The OS

Using the System Software

Application Programmer

Software API

Command Line Interpreter

Loader

Window System

Compiler

Libraries

OS

System Software

Database Management System

Hardware
The OS as Resource Manager

• **Process**: An executing program

• **Resource**: Anything that is needed for a process to run
  – Memory
  – Space on a disk
  – The CPU

• “An OS creates resource abstractions”

• “An OS manages resource sharing”
Resource Abstraction

```c
load(block, length, device);
seek(device, 236);
out(device, 9)
```

```c
write(char *block, int len, int device,
     int track, int sector) {
    ...
    load(block, length, device);
    seek(device, 236);
    out(device, 9);
    ...
}
```

```c
write(char *block, int len, int device, int addr);
```

```c
fprintf(fileID, "%d", datum);
```

Disk Abstractions

(a) Direct Control

(b) write() abstraction

(c) fprintf() abstraction

Application Programmer

OS Programmer
Abstract Resources

- User Interface
- Application
- Abstract Resources (API)
- Middleware
- OS Resources (OS Interface)
- OS
- Hardware Resources

Abstract Machines

Idea → Program → Abstract Machines → Physical Machine → Result
Resource Sharing

• Space- vs time-multiplexed sharing
• To control sharing, must be able to isolate resources
• OS usually provides mechanism to isolate, then selectively allows sharing
  – How to isolate resources
  – How to be sure that sharing is acceptable
• Concurrency

The OS as a Conductor

The OS *coordinates the sharing and use* of all the components in the computer
Multiprogramming

- Technique for *sharing* the CPU among *runnable* processes
  - Process may be *blocked* on I/O
  - Process may be *blocked* waiting for other resource, including the CPU
- While one process is blocked, another might be able to run
- Multiprogramming OS accomplishes CPU sharing “automatically” – *scheduling*
- Reduces time to run all processes
How Multiprogramming Works

Process 1
Process 2
Process 3
Process 4

Time-multiplexed CPU
Space-multiplexed Memory

Speeding Up the Car Wash

(a) The Sequential Car Wash
(b) The Parallel Car Wash
Multiprogramming Performance

Time

0 \( t_i \)

\( t_i \)

(a) \( P_i \)'s Use of Machine Resources

\( P_1 \)

\( P_2 \)

\( \ldots \)

\( P_i \)

\( \ldots \)

\( P_N \)

Time

(a) All Processes’ Use of Machine Resources

- Using the processor
- I/O operation

OS Strategies

- Batch processing
- Timesharing
- Personal computer & workstations
- Process control & real-time
- Network
- Distributed
- Small computers
Batch Processing

Job 19

Input Spooler

Input Spool

Job 3

Output Spooler

Output Spool

Batch Processing (2)

- Uses multiprogramming
- *Job* (file of OS commands) prepared offline
- Batch of jobs given to OS at one time
- OS processes jobs one-after-the-other
- No human-computer interaction
- OS optimizes resource utilization
- Batch processing (as an option) still used today
A Shell Script Batch File

cc -g -c menu.c
cc -g -o driver driver.c menu.o
driver < test_data > test_out
lpr -PthePrinter test_out
tar cvf driver_test.tar menu.c driver.c test_data test_out
uuencode driver_test.tar driver_test.tar >driver_test.encode

Timesharing Systems
Timesharing Systems(2)

- Uses multiprogramming
- Support interactive computing model (Illusion of multiple consoles)
- Different scheduling & memory allocation strategies than batch
- Tends to propagate processes
- Considerable attention to resource isolation (security & protection)
- Tend to optimize response time

Personal Computers

- CPU sharing among one person’s processes
- Power of computing for personal tasks
  - Graphics
  - Multimedia
- Trend toward very small OS
- OS focus on resource abstraction
- Rapidly evolved to “personal multitasking” systems
Process Control & Real-Time

- Computer is dedicated to a single purpose
- Classic embedded system
- Must respond to external stimuli in fixed time
- Continuous media popularizing real-time techniques
- An area of growing interest

Networks

- LAN (Local Area Network) evolution
- High speed communication means new way to do computing
  - Shared files
  - Shared memory
  - Shared procedures/objects
  - ???
Distributed OS

• Wave of the future

Small Computers

• PDAs, STBs, embedded systems became commercially significant
• Have an OS, but
  – Not general purpose
  – Limited hardware resources
  – Different kinds of devices
    • Touch screen, no keyboard
    • Graffiti
  – Evolving & leading to new class of Oses
• PalmOS, Pocket PC (WinCE), VxWorks, …
Evolution of Modern OS

Batch
- Memory Mgmt
- Protection
- Scheduling
- Files
- Devices

Timesharing

PC & Wkstation
- System software
- Human-Computer Interface

Network OS
- Client-Server Model
- Protocols
- Scheduling

Real-Time

Small Computer
- Network storage, Resource management

Modern OS

Examples of Modern OS

- UNIX variants (e.g. Linux) -- have evolved since 1970
- Windows NT/2K -- has evolved since 1989 (much more modern than UNIX
  - Win2K = WinNT, V5
- Research OSes – still evolving …
- Small computer OSes – still evolving …
The Microsoft OS Family

- Win32 API
- Win32 API Subset
- Windows CE (Pocket PC)
- Windows 95/98/Me
- Windows NT/2000/XP

Summary

An Operating System must be able to:
- provide functionality to apps
- provide abstraction of hardware to users and apps
- provide the sharing of resources to processes
- provide security and protection
- be as transparent as possible
- be as light as possible