Operating Systems
CS 571
Prof. Sanjeev Setia
Fall 2002

Overview

• Prerequisites
  – Computer Architecture (CS 365)
  – Data structures and programming (CS 310)
    • (C++/C/Java programming)
• Textbooks
  – Modern Operating Systems (Tannenbaum)
  – Distributed Systems – Concepts and Design
    (Coulouris, Dollimore, Kindberg)
Overview cont’d

- **Grading**
  - One midterm exam (25%), Final Exam (25%)
  - Assignments (50%)

- **Four Programming Assignments**
  - Concurrent Programming (2 weeks)
  - Interprocess Communication
    - Socket programming (2 weeks)
    - RMI/RPC (2 weeks)
  - Distributed File System (5 weeks)
    - First three to be done individually, fourth may be done in groups of two
    - Late submission penalty – 10% per day
  - Plagiarism => Severe penalties (fail the class, expelled from program)

Logistics

- **Class Web Page**
  - Slides, Handouts, Old Exams, Useful Links

- **Slides**
  - More than 90% of slides taken from slides made available by authors of textbooks
  - [http://www.cs.vu.nl/~ast/books/mos2/](http://www.cs.vu.nl/~ast/books/mos2/)
  - [http://www.cdk3.net/](http://www.cdk3.net/)
Logistics

• Office Hrs
  – Tuesday, 2 – 4 pm, Room 347, S&T II
  – setia@cs.gmu.edu
• TA
  – Arshad Ahmed (aahmed8@gmu.edu)
  – Office Hrs: Mon 2:30-4:30 pm, Tue 7:30-9:00 pm
  – Office, S&T II Room 435
• Computer Accounts (IT&E or your own machine at home/work)

Chapter 1

Introduction

1.1 What is an operating system
1.2 History of operating systems
1.3 The operating system zoo
1.4 Computer hardware review
1.5 Operating system concepts
1.6 System calls
1.7 Operating system structure
Introduction

A computer system consists of:
- hardware
- system programs
- application programs

What is an Operating System

- It is an extended machine
  - Hides the messy details of actions which must be performed to use the hardware
  - Presents user with a virtual machine, easier to use
- It is a resource manager
  - Each program gets time with the resource
  - Each program gets space on the resource
History of Operating Systems (1)

Early batch system
- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

History of Operating Systems (2)

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- Fifth generation?? 2000 – present
  - Handheld and embedded devices
History of Operating Systems (3)

• Structure of a typical FMS job – 2nd generation

History of Operating Systems (4)

• Multiprogramming system
  – three jobs in memory – 3rd generation
The Operating System Zoo

- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Real-time operating systems
- Embedded operating systems
- Smart card operating systems

Migration of Operating-System Concepts and Features
Computer Hardware Review (1)

- Components of a simple personal computer

Computer Hardware Review – CPU architecture

(a) A three-stage pipeline
(b) A superscalar CPU
Computer Hardware Review –
CPU architecture

• CPU has several general-purpose and special-purpose registers
• Special purpose registers
  – PC (program counter)
  – SP (stack pointer)
  – PSW (processor status word)
    • Reflects “status” of CPU, i.e., user/kernel mode, interrupt mask, etc.

Computer Hardware Review –
Storage Hierarchy

Typical access time                    Typical capacity

1 nsec                                <1 KB
2 nsec                                1 MB
10 nsec                               64-512 MB
10 msec                               5-50 GB
100 sec                               20-100 GB

• Typical storage hierarchy
  – numbers shown are rough approximations
Computer Hardware Review - Disk Drives

Structure of a disk drive

Computer Hardware Review - Memory Protection

One base-limit pair and two base-limit pairs
Computer Hardware Review – Interrupts

(a) Steps in starting an I/O device and getting interrupt
(b) How the CPU is interrupted

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - Usually a vectored interrupt system
- Separate segments of code (interrupt handlers) determine what action should be taken for each type of interrupt
Interrupt Time Line For a Single Process Doing Output

Computer Hardware Review - Direct Memory Access (DMA)

• Used for high-speed I/O devices able to transmit information at close to memory speeds.
• Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
• Only one interrupt is generated per block, rather than the one interrupt per byte.
Computer Hardware Review - Dual-Mode Operation

• Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
• Provide hardware support to differentiate between at least two modes of operations.
  1. **User mode** – execution done on behalf of a user.
  2. **Monitor mode** (also **kernel mode** or **system mode**) – execution done on behalf of operating system.

Dual-Mode Operation (Cont.)

• **Mode bit** added to computer hardware to indicate the current mode: monitor (0) or user (1).
• When an interrupt or fault occurs hardware switches to monitor mode.

*Privileged instructions* can be issued only in monitor mode.
Computer Hardware Review

Structure of a large Pentium system

Operating System Concepts (1)

- A process tree
  - A created two child processes, B and C
  - B created three child processes, D, E, and F
Operating System Concepts (2)

File system for a university department

Operating System Concepts (3)

- Before mounting,
  - files on floppy are inaccessible
- After mounting floppy on b,
  - files on floppy are part of file hierarchy
Operating System Concepts (4)

Two processes connected by a pipe

Operating System Functions

- Process management
- Inter-process communication
- Memory management
- File management
- I/O and Storage system management
  - Power management in newer systems
- Networking
- Security & Protection
- User Interface
  - Can be part of the OS (e.g. Windows) or outside the OS (Unix)
Operating System Interface

- **System commands**
  - `ls`, `ps`, `cat` (UNIX), `dir`, `type` (Windows)
- **System calls**
  - API for invoking operating system services
  - `read()`, `write()`, `fork()`

Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking
Steps in Making a System Call

There are 11 steps in making the system call `read(fd, buffer, nbytes)`

Some System Calls For Process Management

<table>
<thead>
<tr>
<th>Process management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pid = fork()</code></td>
<td>Create a child process identical to the parent</td>
</tr>
<tr>
<td><code>pid = waitpid(pid, &amp;status, options)</code></td>
<td>Wait for a child to terminate</td>
</tr>
<tr>
<td><code>s = execve(name, argv, environ)</code></td>
<td>Replace a process' core image</td>
</tr>
<tr>
<td><code>exit(status)</code></td>
<td>Terminate process execution and return status</td>
</tr>
</tbody>
</table>
### Some System Calls For File Management

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd = open(file, how, ...)</code></td>
<td>Open a file for reading, writing or both</td>
</tr>
<tr>
<td><code>s = close(fd)</code></td>
<td>Close an open file</td>
</tr>
<tr>
<td><code>n = read(fd, buffer, nbytes)</code></td>
<td>Read data from a file into a buffer</td>
</tr>
<tr>
<td><code>n = write(fd, buffer, nbytes)</code></td>
<td>Write data from a buffer into a file</td>
</tr>
<tr>
<td><code>position = lseek(fd, offset, whence)</code></td>
<td>Move the file pointer</td>
</tr>
<tr>
<td><code>s = stat(name, &amp;buf)</code></td>
<td>Get a file's status information</td>
</tr>
</tbody>
</table>

### Some System Calls For Directory Management

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s = mkdir(name, mode)</code></td>
<td>Create a new directory</td>
</tr>
<tr>
<td><code>s = rmdir(name)</code></td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td><code>s = link(name1, name2)</code></td>
<td>Create a new entry, name2, pointing to name1</td>
</tr>
<tr>
<td><code>s = unlink(name)</code></td>
<td>Remove a directory entry</td>
</tr>
<tr>
<td><code>s = mount(special, name, flag)</code></td>
<td>Mount a file system</td>
</tr>
<tr>
<td><code>s = umount(special)</code></td>
<td>Unmount a file system</td>
</tr>
</tbody>
</table>
Some System Calls For Miscellaneous Tasks

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s = chdir(dirname)</code></td>
<td>Change the working directory</td>
</tr>
<tr>
<td><code>s = chmod(name, mode)</code></td>
<td>Change a file’s protection bits</td>
</tr>
<tr>
<td><code>s = kill(pid, signal)</code></td>
<td>Send a signal to a process</td>
</tr>
<tr>
<td><code>seconds = time(&amp;seconds)</code></td>
<td>Get the elapsed time since Jan. 1, 1970</td>
</tr>
</tbody>
</table>

System Calls (1)

- A stripped down shell:

  ```c
  while (TRUE) {
    /* repeat forever */
    type_prompt( ); /* display prompt */
    read_command (command, parameters) /* input from terminal */
    if (fork() != 0) {
      /* Parent code */
      waitpid( -1, &status, 0); /* wait for child to exit */
    } else {
      /* Child code */
      execve (command, parameters, 0); /* execute command */
    }
  }
  ```
System Calls (2)

- Processes have three segments: text, data, stack

System Calls (3)

(a) Two directories before linking
/usr/jim/memo to ast's directory
(b) The same directories after linking
System Calls (4)

(a) File system before the mount
(b) File system after the mount

System Calls (5)

<table>
<thead>
<tr>
<th>UNIX</th>
<th>Win32</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>CreateProcess</td>
<td>Create a new process</td>
</tr>
<tr>
<td>waitpid</td>
<td>WaitForSingleObject</td>
<td>Can wait for a process to exit</td>
</tr>
<tr>
<td>execv</td>
<td>(none)</td>
<td>CreateProcess + fork + execve</td>
</tr>
<tr>
<td>exit</td>
<td>ExitProcess</td>
<td>Terminate execution</td>
</tr>
<tr>
<td>open</td>
<td>CreateFile</td>
<td>Create a file or open an existing file</td>
</tr>
<tr>
<td>close</td>
<td>CloseHandle</td>
<td>Close a file</td>
</tr>
<tr>
<td>read</td>
<td>ReadFile</td>
<td>Read data from a file</td>
</tr>
<tr>
<td>write</td>
<td>WriteFile</td>
<td>Write data to a file</td>
</tr>
<tr>
<td>lseek</td>
<td>SetFilePointer</td>
<td>Move the file pointer</td>
</tr>
<tr>
<td>stat</td>
<td>GetFileAttributesEx</td>
<td>Get various file attributes</td>
</tr>
<tr>
<td>mkdir</td>
<td>CreateDirectory</td>
<td>Create a new directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>RemoveDirectory</td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td>link</td>
<td>(none)</td>
<td>Win32 does not support links</td>
</tr>
<tr>
<td>unlink</td>
<td>DeleteFile</td>
<td>Destroy an existing file</td>
</tr>
<tr>
<td>mount</td>
<td>(none)</td>
<td>Win32 does not support mount</td>
</tr>
<tr>
<td>umount</td>
<td>(none)</td>
<td>Win32 does not support mount</td>
</tr>
<tr>
<td>chdir</td>
<td>SetCurrentDirectory</td>
<td>Change the current working directory</td>
</tr>
<tr>
<td>chmod</td>
<td>(none)</td>
<td>Win32 does not support security (although NT does)</td>
</tr>
<tr>
<td>kill</td>
<td>(none)</td>
<td>Win32 does not support signals</td>
</tr>
<tr>
<td>time</td>
<td>GetLocalTime</td>
<td>Get the current time</td>
</tr>
</tbody>
</table>

Some Win32 API calls
Operating System Structure (1)

Simple structuring model for a monolithic system

Operating System Structure (2)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The operator</td>
</tr>
<tr>
<td>4</td>
<td>User programs</td>
</tr>
<tr>
<td>3</td>
<td>Input/output management</td>
</tr>
<tr>
<td>2</td>
<td>Operator-process communication</td>
</tr>
<tr>
<td>1</td>
<td>Memory and drum management</td>
</tr>
<tr>
<td>0</td>
<td>Processor allocation and multiprogramming</td>
</tr>
</tbody>
</table>

Structure of the THE operating system
Operating System Structure (3)

Structure of VM/370 with CMS

Operating System Structure (4)

The client-server model
Operating System Structure (5)

The client-server model in a distributed system