Introduction to OSs

• What is an Operating System?

• Architectural Support for Operating Systems

• System Calls

• Basic Organization of an Operating System

• Reading: Silberschatz (8th ed), Chapters 1, 2
What is an operating system?

- What an operating system is not:
  - An o.s. is not a language or a compiler
  - An o.s. is not a command interpreter / window system
  - An o.s. is not a library of commands
  - An o.s. is not a set of utilities

A Short Historical Tour

- **First Generation** Computer Systems (1949-1956):
  - Single user: writes program, operates computer through console or card reader / printer
  - Absolute machine language
  - I/O devices
  - Development of libraries; device drivers
  - Compilers, linkers, loaders
  - Relocatable code
Programming Early Machines

Wiring the ENIAC with a new program
(U.S. Army photo, from archives of the ARL Technical Library)


- Problems: scheduling, setup time
- Automation of Load/Translate/Load/Execute
  - Batch systems
  - Monitor programs
  - Job Control Language
  - Advent of operators: computers as input/output box
- Problem: Resource management and I/O still under control of programmer
  - Memory protection
  - Timers
  - Privileged instructions

Device drivers
Job sequencer / loader
Control card interpreter
Monitor
User program area
Example: IBM Punch Card System

Card Punch

Card Verifier

Card Sorter

(Computer Museum of America)

Batching Program Execution

$FTN
$JOB
$END
$LOAD
$RUN
... Program ...
... Data ...
$JOB
$FTN
Overlapping CPU and I/O Operations

Traditional Batch Operation:

- card reader → CPU → line printer

Off-Line Processing:

- card readers → CPU → line printers

Spooling: I/O Channels:

- disk → card reader → CPU → line printer

Off-Line vs. Pure Batch

- batch

- off-line (single set of card reader/printer)
Off-Line vs. Pure Batch (II)

- **batch**

  - card reader
  - CPU
  - printer

- **off-line (multiple of card readers/printers)**

  - card reader
  - tape reader
  - CPU
  - tape reader
  - printer


- Problem with batching: one-job-at-a-time
  - sequential:
    - CPU
    - I/O
  - better:
    - CPU
    - I/O

- Solution: **Multiprogramming**
  - Job pools: have several programs ready to execute
  - Keep several programs in memory

- New issues:
  - Job scheduling
  - Memory management
  - Protection
Time Sharing (mid 1960s on)

- OS interleaves execution of multiple user programs with time quantum
  - CTSS (1961): time quantum 0.2 sec

- User returns to own the machine

- New aspects and issues:
  - On-line file systems
  - Resource protection
  - Virtual memory
  - Sophisticated process scheduling

- Advent of systematic techniques for designing and analyzing OSs.

The Recent Past

- Personal computers and Computing as Utility
  - History repeats itself

- Parallel systems
  - Resource management
  - Fault tolerance

- Real-Time Systems

- Distributed Systems
  - Communication
  - Resource sharing
  - Network operating systems
  - Distributed operating systems

- Secure Systems
The Future?

- The “Invisible Computer”
- Computing-in-the-ultra-small
- Speed vs. Power vs. Heat
- Breaking up the layered design

What, then, is an Operating System?

- Controls and coordinates the use of system resources.

- **Primary goal**: Provide a convenient environment for a user to access the available resources (CPU, memory, I/O)
  - Provide appropriate abstractions (files, processes, ...)
  - “virtual machine”

- **Secondary goal**: Efficient operation of the computer system.

- **Resource Management**
  - **Transforming**: Create virtual substitutes that are easier to use.
  - **Multiplexing**: Create the illusion of multiple resources from a single resource
  - **Scheduling**: “Who gets the resource when?”
**Resources**

- Disks and other Devices
- Timers / Clocks
- Locks
- CPU
- Memory
- Power / Heat
- I/O Controllers

**The OS as Servant to Two Masters**

- Devices
- Clocks&Timers
- Locks
- Memory
- Heat&Power
- I/O Controllers
- CPUs

- Performance
- Plug&Play
- Security
- Predictability
- Convenience
- Fault-Tolerance
- Power-Effectiveness
- ...
Introduction to OSs

- What is an Operating System?
- Architectural Support for Operating Systems
- System Calls
- Basic Organization of an Operating System

Architectural Support for OS’s

- Dealing with Asynchronous Events: Exceptions, Interrupts
  - Modern OS’s are interrupt-driven (some still are not!).
  - Simple interrupt handling vs. exception handling MIPS-style.

- Hardware Protection
  - Privilege Levels (e.g. user/kernel/supervisor, etc.)
  - Privileged instructions: typically CPU control instructions
  - I/O Protection
  - Memory Protection

- Support for Address Spaces

- Timers
Modern OS’s are Interrupt-Driven

Interrupts / Exceptions

- When an interrupt occurs, CPU stops, saves state, typically changes into supervisor mode, and immediately jumps to predefined location.
- Appropriate interrupt service routine is found through the interrupt vector.
- Return-from-interrupt automatically restores state.

- Interrupts/Exceptions can be invoked by asynchronous events (I/O devices, timers, various errors) or can be software-generated (system calls).
Exceptions, MIPS-Style

- MIPS CPU deals with exceptions.
  - Interrupts are just a special case of exceptions.

- The MIPS Architecture has no interrupt-vector table!
  - All exceptions trigger a jump to the same location, and de-multiplexing happens in the exception handler, after looking up the reason for the exception in the `CAUSE` register.

MIPS Exception Handler (low-level)

- set up exception frame on stack
- save enough registers to get by
- save rest of registers
- call C exception handler
- restore registers
- return from exception
Hardware Protection

- Originally: User owned the machine, no monitor. No protection necessary.
- Resident monitor, resource sharing: One program can adversely affect the execution of others.
- Examples
  - `halt` instruction
  - modify data or code in other programs or monitor itself
  - access/modify data on storage devices
  - refuse to relinquish processor
- Benign (bug) vs. malicious (virus)

Hardware Protection (2)

- Dual-mode operation
  - `user mode` vs. `supervisor mode`
  - e.g. `halt` instruction is privileged.
- I/O Protection
  - define all I/O operations to be `privileged`
- Memory Protection
  - protect interrupt vector, interrupt service routines
  - determine legal address ranges

CPU >= base

CPU < base + limit

memory

trap to operating system!
Timers

- Timers can be set, and a trap occurs when the timer expires. (And OS acquires control over the CPU.)

- Other uses of timers:
  - time sharing
  - time-of-day

Introduction to OSs

- What is an Operating System?
- Architectural Support for Operating Systems
- System Calls
- Basic Organization of an Operating System
External Structure of an OS

The outsider’s view of the OS.

System Calls

Provide the interface between a process and the OS.

Example: vanilla copy:

```c
int copy(char *fname1, *fname2) {
    FILE *f, *g;
    char c;
    f = fopen(fname1, "r");
    g = fopen(fname2, "w");
    while (read(f, &c, 1) > 0)
        write(g, c, 1);
    fclose(f);
    fclose(g);
}
```
System Call Implementation: Linux on x86

- **Example:** `_syscalls(int, setuid, uid_t, uid)`
- **expands to:**
  ```
  _setuid:
  subl $4,%esp
  pushl %ebx
  movl 12(%esp),%eax
  movl %eax,4(%esp)
  movl $23,%eax  <<--- System Call number (setuid = 23)
  mov 4(%esp),%ebx
  int $0x80
  movl %eax,%edx
  text1 leds,leds
  jge L2
  negl %edx
  movl %edx,_errno
  movl $-1,%eax
  popl %ebx
  addl $4,%esp
  retL2:
  movl %edx,leds
  popl %ebx
  addl $4,%esp
  ret
  ```

Why Interrupts?

**Reason 1:** Can load user program into memory without knowing exact address of system procedures

**Reason 2:** Separation of address space, including stacks: 
*user stack* and *kernel stack*.

**Reason 3:** Automatic change to *supervisor mode*.

**Reason 4:** Can control *access* to kernel by masking interrupts.
Reason 2: Buffer Overrun Attacks (Silberschatz et al)

```c
#include <stdio.h>
#define BUFFER_SIZE 256
int main(int argc, char *argv[])
{
    char buffer[BUFFER_SIZE];
    if (argc < 2)
        return -1;
    else {
        strcpy(buffer, argv[1]);
        return 0;
    }
}
```

Stack Separation sufficient?

- Buffer overruns in kernel code?
- Device drivers?
Reason 4: Mutual Exclusion in Kernel

1. User process 1
2. System call
3. Trap
4. User process 2

Process 1 executing in kernel
interrupts are masked

Process 2 can not enter
kernel because
of masked interrupts

Unmask interrupts
and return

System Call Implementation: Linux on x86

- Example: _syscall(int, setuid, uid_t, uid)
- Expands to:

```assembly
_setuid:
  subl $4, temp
  pushl %ebx
  moval 12(temp), %eax
  movl %eax,4(temp)
  movl $23, %eax  <------ System Call number (setuid = 23)
  mov 4(temp), %eax
  int $0x80  <------ Call transfer to kernel entry point _system_call()
  mov %eax, %edx
  testl %edx, %edx
  jgp l2
  negl %edx
  movl %edx, _errno
  movl 5-1, %eax
  popl %ebx
  addl $4, temp
rtoi2:
  movl %edx, %eax
  pop %ebx
  addl $4, temp
  ret
```
Why Interrupts?

Reason 1: Can load user program into memory without knowing exact address of system procedures.

Reason 2: Separation of address space, including stacks: user stack and kernel stack.

Reason 3: Automatic change to supervisor mode.

Reason 4: Can control access to kernel by masking interrupts.

---

Reason 2: Buffer Overrun Attacks (Silberschatz et al)

Stack Separation sufficient?

- Buffer overruns in kernel code?
- Device drivers?

Reason 4: Mutual Exclusion in Kernel

1. User process 1
2. System call
3. Trap
4. User process 2
5. Unmask interrupts and return

Process 1 executing in kernel interrupts are masked
Process 2 can not enter kernel because of masked interrupts
Introduction to OSs

- What is an Operating System?
- Architectural Support for Operating Systems
- System Calls
- Basic Organization of an Operating System

External Structure of an OS

The outsider’s view of the OS.

- Applications programs/processes
- System call interface
- Kernel
- Device drivers
- Hardware
Internal Structure: Layered Services

The insider’s view of the OS.
Example: XINU [Comer 1984]

- user programs
- file system
- intermachine network communication
- device manager and device drivers
- real-time clock manager
- interprocess communication
- process coordinator
- process manager
- memory manager
- hardware

Internal Structure: micro-Kernels

- Layered Kernels vs. Microkernels

Hierarchical decomposition.
Interaction only between adjacent layers.

Kernel has only core operating system functions (memory management, IPC, I/O, interrupts)
Other functions run in server processes in user space.
Operations in a m-Kernel

- Non-kernel components of the OS are implemented as server processes.
- Communication between user and servers using messages through kernel.
- “client-server architecture within a single computer”
- Examples: Mach, Windows NT, Chorus, L4, ...

Windows 2000/XP System Structure

Figure 2.— System architecture
Benefits of micro-Kernels

- Extensibility:
  - New services can be added by adding server processes.
- Flexibility:
  - Services can be customized.
- Portability:
  - Kernel small, with well-defined interface.
- Distributed System Support:
  - Interface between users and services is message-based.

micro-Kernels: Performance is Problem

- Request traverses user/kernel boundary twice, same for reply.
- Solutions:
  - Move critical services back into the kernel ("make kernel bigger")
  - Make kernel “smaller”
Why are OSs so Slow? (Why Aren’t Operating Systems Getting Faster As Fast As Hardware? John Ousterhout, 1989)

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Abbreviation</th>
<th>RISC/CISC</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS M2000</td>
<td>M2000</td>
<td>RISC</td>
<td>20</td>
</tr>
<tr>
<td>DECstation 3100</td>
<td>DS3100</td>
<td>RISC</td>
<td>13</td>
</tr>
<tr>
<td>Sun-4/280</td>
<td>Sun4</td>
<td>RISC</td>
<td>9</td>
</tr>
<tr>
<td>VAX 8800</td>
<td>8800</td>
<td>CISC</td>
<td>6</td>
</tr>
<tr>
<td>Sun-3/75</td>
<td>Sun3</td>
<td>CISC</td>
<td>1.8</td>
</tr>
<tr>
<td>Microvax II</td>
<td>MVAX2</td>
<td>CISC</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 1: Hardware Platforms

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (microseconds)</th>
<th>MIPS-Relative Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2000 RISC/4.0</td>
<td>18</td>
<td>0.54</td>
</tr>
<tr>
<td>DS3100 Sprite</td>
<td>26</td>
<td>0.49</td>
</tr>
<tr>
<td>DS3100 Ultra 3.1</td>
<td>25</td>
<td>0.60</td>
</tr>
<tr>
<td>8800 Ultra 3.0</td>
<td>28</td>
<td>1.15</td>
</tr>
<tr>
<td>Sun4 SunOS 4.0</td>
<td>32</td>
<td>0.68</td>
</tr>
<tr>
<td>Sun3 Sprite</td>
<td>32</td>
<td>0.58</td>
</tr>
<tr>
<td>Sun3 SunOS 3.5</td>
<td>92</td>
<td>1.0</td>
</tr>
<tr>
<td>MVAX2 Ultra 3.0</td>
<td>207</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 2: Getzps kernel call time

Why are OSs so Slow? (2)

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Abbreviation</th>
<th>RISC/CISC</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS M2000</td>
<td>M2000</td>
<td>RISC</td>
<td>20</td>
</tr>
<tr>
<td>DECstation 3100</td>
<td>DS3100</td>
<td>RISC</td>
<td>13</td>
</tr>
<tr>
<td>Sun-4/280</td>
<td>Sun4</td>
<td>RISC</td>
<td>9</td>
</tr>
<tr>
<td>VAX 8800</td>
<td>8800</td>
<td>CISC</td>
<td>6</td>
</tr>
<tr>
<td>Sun-3/75</td>
<td>Sun3</td>
<td>CISC</td>
<td>1.8</td>
</tr>
<tr>
<td>Microvax II</td>
<td>MVAX2</td>
<td>CISC</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 1: Hardware Platforms

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (ms)</th>
<th>MIPS-Relative Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2000 RISC/4.0</td>
<td>0.30</td>
<td>0.71</td>
</tr>
<tr>
<td>DS3100 Ultra 3.1</td>
<td>0.54</td>
<td>0.96</td>
</tr>
<tr>
<td>DS3100 Sprite</td>
<td>0.51</td>
<td>0.65</td>
</tr>
<tr>
<td>8800 Ultra 3.0</td>
<td>0.70</td>
<td>1.0</td>
</tr>
<tr>
<td>Sun4 SunOS 4.0</td>
<td>1.02</td>
<td>0.47</td>
</tr>
<tr>
<td>Sun4 Sprite</td>
<td>1.17</td>
<td>0.41</td>
</tr>
<tr>
<td>Sun3 SunOS 3.5</td>
<td>2.36</td>
<td>1.0</td>
</tr>
<tr>
<td>Sun3 Sprite</td>
<td>2.41</td>
<td>1.0</td>
</tr>
<tr>
<td>MVAX2 Ultra 3.0</td>
<td>3.66</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 3: Cowsibc: echo one byte between processes using pipes.