Design and Implementation of OS not “solvable”, but some approaches have proven successful

Internal structure of different Operating Systems can vary widely

Start the design by defining goals and specifications

Affected by choice of hardware, type of system

**User** goals and **System** goals

- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
Important principle to separate

**Policy**: *What* will be done?

**Mechanism**: *How* to do it?

Mechanisms determine how to do something, policies decide what will be done

The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example – timer)

Specifying and designing an OS is highly creative task of software engineering
Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware
Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel - Mach
Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
Non Simple Structure -- UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- Systems programs
- The kernel
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level
Beyond simple but not fully layered

<table>
<thead>
<tr>
<th>(the users)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shells and commands</td>
</tr>
<tr>
<td>compilers and interpreters</td>
</tr>
<tr>
<td>system libraries</td>
</tr>
</tbody>
</table>

**system-call interface to the kernel**

- signals terminal handling
- character I/O system drivers
- file system swapping
  - block I/O system
  - disk and tape drivers
- CPU scheduling
  - page replacement
  - demand paging
  - virtual memory

**kernel interface to the hardware**

- terminal controllers
  - terminals
- device controllers
  - disks and tapes
- memory controllers
  - physical memory
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of microkernel
  - Mac OS X kernel (*Darwin*) partly based on Mach
- Communication takes place between user modules using message passing

**Benefits:**
- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

**Detriments:**
- Performance overhead of user space to kernel space communication
Microkernel System Structure

- Application Program
- File System
- Device Driver

- Interprocess Communication
- memory management
- CPU scheduling

- messages
- messages

- microkernel

- hardware

- user mode
- kernel mode
Many modern operating systems implement loadable kernel modules

- Uses object-oriented approach
- Each core component is separate
- Each talks to the others over known interfaces
- Each is loadable as needed within the kernel

Overall, similar to layers but with more flexible

- Linux, Solaris, etc
Solaris Modular Approach

- device and bus drivers
- scheduling classes
- file systems
- loadable system calls
- executable formats
- STREAMS modules
- miscellaneous modules

core Solaris kernel
Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem personalities
- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)
### Mac OS X Structure

<table>
<thead>
<tr>
<th>Graphical User Interface</th>
<th>Aqua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Environments and Services</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>Cocoa</td>
</tr>
<tr>
<td>Kernel Environment</td>
<td></td>
</tr>
<tr>
<td>Mach</td>
<td>BSD</td>
</tr>
<tr>
<td>I/O Kit</td>
<td>Kernel Extensions</td>
</tr>
</tbody>
</table>
iOS

Apple mobile OS for **iPhone, iPad**
- Structured on Mac OS X, added functionality
- Does not run OS X applications natively
  - Also runs on different CPU architecture (ARM vs. Intel)
- **Cocoa Touch** Objective-C API for developing apps
- **Media services** layer for graphics, audio, video
- **Core services** provides cloud computing, databases
- Core operating system, based on Mac OS X kernel
Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc
Android Architecture

Application Framework

Libraries

- SQLite
- openGL
- surface manager
- media framework
- webkit
- libc

Android runtime

- Core Libraries
- Dalvik virtual machine
Operating-System Debugging

- **Debugging** is finding and fixing errors, or bugs
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using **trace listings** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan’s Law: “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”
Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager
DTrace

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems

- **Probes** fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes

- Example of following XEventsQueued system call move from libc library to kernel and back

```bash
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
 0 -> XEventsQueued  U
 0 -> _XEventsQueued  U
 0 -> _X11TransBytesReadable  U
 0 <- _X11TransBytesReadable  U
 0 -> _X11TransSocketBytesReadable  U
 0 <- _X11TransSocketBytesReadable  U
 0 -> ioctl  U
 0 -> ioctl  K
 0 -> getf  K
 0 -> set_active_fd  K
 0 <- set_active_fd  K
 0 <- getf  K
 0 -> get_udatamodel  K
 0 <- get_udatamodel  K
...
 0 -> releases  K
 0 -> clear_active_fd  K
 0 <- clear_active_fd  K
 0 -> cv_broadcast  K
 0 <- cv_broadcast  K
 0 <- releases  K
 0 <- ioctl  K
 0 <- ioctl  U
 0 <- _XEventsQueued  U
 0 <- XEventsQueued  U
```
DTrace (Cont.)

- DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
sched::on-cpu
uid == 101
{ 
    self->ts = timestamp;
}
sched::off-cpu
self->ts
{
    @time[execname] = sum(timestamp - self->ts);
    self->ts = 0;
}
```

```
# dtrace -s sched.d
 dtrace: script 'sched.d' matched 6 probes

  %C
  gnome-settings-d  142354
  gnome-vfs-daemon  158243
  dsdm               189804
  wnck-applet       200030
  gnome-panel       277864
  clock-applet      374916
  mapping-daemon    385475
  xscreensaver      514177
  metacity          539281
  Xorg              2579646
  gnome-terminal    5007269
  mixer.applet2     7388447
  java              10769137

Figure 2.21  Output of the D code.
```
Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.

- **SYSGEN** program obtains information concerning the specific configuration of the hardware system:
  - Used to build system-specific compiled kernel or system-tuned.
  - Can generate more efficient code than one general kernel.
System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – bootstrap loader, stored in ROM or EEPROM locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where boot block at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, GRUB, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then running
End of Chapter 2