Chapter 2
Operating System Overview

Eighth Edition
By William Stallings
Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware

Main objectives of an OS:
- convenience
- efficiency
- ability to evolve
Figure 2.1 Computer Hardware and Software Structure
Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting
Key Interfaces

- Instruction set architecture (ISA)
- Application binary interface (ABI)
- Application programming interface (API)
The Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data
- The OS is responsible for managing these resources
Operating System as Software

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must depend on the processor to allow it to regain control
Figure 2.2   The Operating System as Resource Manager
A major OS will evolve over time for a number of reasons:

- hardware upgrades
- new types of hardware
- new services
- Fixes
Evolution of Operating Systems

- Stages include:
  - Serial Processing
  - Simple Batch Systems
  - Multiprogrammed Batch Systems
  - Time Sharing Systems
Serial Processing

Earliest Computers:

- No operating system
  - programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in “series”

Problems:

- Scheduling:
  - most installations used a hardcopy sign-up sheet to reserve computer time
    - time allocations could run short or long, resulting in wasted computer time
- Setup time
  - a considerable amount of time was spent just on setting up the program to run
Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization

- Monitor
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them together and places them on an input device
  - program branches back to the monitor when finished
Monitor Point of View

- Monitor controls the sequence of events
- **Resident Monitor** is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

Figure 2.3 Memory Layout for a Resident Monitor
Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "control is passed to a job" means processor is fetching and executing instructions in a user program
- "control is returned to the monitor" means that the processor is fetching and executing instructions from the monitor program
Job Control Language (JCL)

Special type of programming language used to provide instructions to the monitor

- what compiler to use
- what data to use
### Desirable Hardware Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory protection for monitor</td>
<td>while the user program is executing, it must not alter the memory area containing the monitor</td>
</tr>
<tr>
<td>Timer</td>
<td>prevents a job from monopolizing the system</td>
</tr>
<tr>
<td>Privileged instructions</td>
<td>can only be executed by the monitor</td>
</tr>
<tr>
<td>Interrupts</td>
<td>gives OS more flexibility in controlling user programs</td>
</tr>
</tbody>
</table>
Modes of Operation

User Mode
- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

Kernel Mode
- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed
Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor

- Sacrifices:
  - some main memory is now given over to the monitor
  - some processor time is consumed by the monitor

- Despite overhead, the simple batch system improves utilization of the computer
Multiprogrammed Batch Systems

- Processor is often idle
- even with automatic job sequencing
- I/O devices are slow compared to processor

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read one record from file</td>
<td>15</td>
</tr>
<tr>
<td>Execute 100 instructions</td>
<td>1</td>
</tr>
<tr>
<td>Write one record to file</td>
<td>15</td>
</tr>
<tr>
<td>TOTAL</td>
<td>31</td>
</tr>
</tbody>
</table>

Percent CPU Utilization = \( \frac{1}{31} \times 100 = 3.2\% \)

Figure 2.4 System Utilization Example
The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding.
Multiprogramming

- There must be enough memory to hold the OS (resident monitor) and one user program.
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O.

<table>
<thead>
<tr>
<th>Program A</th>
<th>Run</th>
<th>Wait</th>
<th>Run</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program B</td>
<td>Wait</td>
<td>Run</td>
<td>Wait</td>
<td>Run</td>
</tr>
<tr>
<td>Combined</td>
<td>Run A</td>
<td>Run B</td>
<td>Wait</td>
<td>Run A</td>
</tr>
</tbody>
</table>

(b) Multiprogramming with two programs

<table>
<thead>
<tr>
<th>Time</th>
<th>(b) Multiprogramming with two programs</th>
</tr>
</thead>
</table>

Figure 2.5   Multiprogramming Example
Multiprogramming

- also known as multitasking
- memory is expanded to hold three, four, or more programs and switch among all of them

(c) Multiprogramming with three programs
# Multiprogramming Example

<table>
<thead>
<tr>
<th>Type of job</th>
<th>JOB1</th>
<th>JOB2</th>
<th>JOB3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>5 min</td>
<td>15 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Memory required</td>
<td>50 M</td>
<td>100 M</td>
<td>75 M</td>
</tr>
<tr>
<td>Need disk?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Need terminal?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Need printer?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.1 Sample Program Execution Attributes
## Effects on Resource Utilization

<table>
<thead>
<tr>
<th></th>
<th>Uniprogramming</th>
<th>Multiprogramming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor use</td>
<td>20%</td>
<td>40%</td>
</tr>
<tr>
<td>Memory use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Disk use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Printer use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>30 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Throughput</td>
<td>6 jobs/hr</td>
<td>12 jobs/hr</td>
</tr>
<tr>
<td>Mean response time</td>
<td>18 min</td>
<td>10 min</td>
</tr>
</tbody>
</table>

Table 2.2 Effects of Multiprogramming on Resource Utilization
Figure 2.6 Utilization Histograms
Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation
**Batch Multiprogramming vs. Time Sharing**

<table>
<thead>
<tr>
<th></th>
<th>Batch Multiprogramming</th>
<th>Time Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal objective</td>
<td>Maximize processor use</td>
<td>Minimize response time</td>
</tr>
<tr>
<td>Source of directives to</td>
<td>Job control language commands provided with the job</td>
<td>Commands entered at the terminal</td>
</tr>
<tr>
<td>operating system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3  Batch Multiprogramming versus Time Sharing
Compatible Time-Sharing Systems

**CTSS**
- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

**Time Slicing**
- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn
Figure 2.7  CTSS Operation
Major Achievements

- Operating Systems are among the most complex pieces of software ever developed

Major advances in development include:

- processes
- memory management
- information protection and security
- scheduling and resource management
- system structure
Fundamental to the structure of operating systems

A *process* can be defined as:

- a program in execution
- an instance of a running program
- the entity that can be assigned to, and executed on, a processor
- a unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources
Development of the Process

Three major lines of computer system development created problems in timing and synchronization that contributed to the development:

- **multiprogramming batch operation**
  - processor is switched among the various programs residing in main memory

- **time sharing**
  - be responsive to the individual user but be able to support many users simultaneously

- **real-time transaction systems**
  - a number of users are entering queries or updates against a database
Causes of Errors

- **Improper synchronization**
  - a program must wait until the data are available in a buffer
  - improper design of the signaling mechanism can result in loss or duplication

- **Failed mutual exclusion**
  - more than one user or program attempts to make use of a shared resource at the same time
  - only one routine at a time allowed to perform an update against the file

- **Nondeterminate program operation**
  - program execution is interleaved by the processor when memory is shared
  - the order in which programs are scheduled may affect their outcome

- **Deadlocks**
  - it is possible for two or more programs to be hung up waiting for each other
  - may depend on the chance timing of resource allocation and release
Components of a Process

A process contains three components:
- an executable program
- the associated data needed by the program (variables, work space, buffers, etc.)
- the execution context (or “process state”) of the program

The execution context is essential:
- it is the internal data by which the OS is able to supervise and control the process
- includes the contents of the various process registers
- includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event
Process Management

- The entire state of the process at any instant is contained in its context.
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature.

Figure 2.8 Typical Process Implementation
The OS has five principal storage management responsibilities:

- Process isolation
- Automatic allocation and management
- Support of modular programming
- Protection and access control
- Long-term storage
Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently
Paging

- Allows processes to be comprised of a number of fixed-size blocks, called pages

- Program references a word by means of a virtual address
  - consists of a page number and an offset within the page
  - each page may be located anywhere in main memory

- Provides for a dynamic mapping between the virtual address used in the program and a real (or physical) address in main memory
Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.
Figure 2.10  Virtual Memory Addressing
The nature of the threat that concerns an organization will vary greatly depending on the circumstances.

The problem involves controlling access to computer systems and the information stored in them.
Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:
  - Fairness
  - Differential responsiveness
  - Efficiency
Figure 2.11 Key Elements of an Operating System for Multiprogramming
Different Architectural Approaches

Demands on operating systems require new ways of organizing the OS

Different approaches and design elements have been tried:

- microkernel architecture
- multithreading
- symmetric multiprocessing
- distributed operating systems
- object-oriented design
Microkernel Architecture

- Assigns only a few essential functions to the kernel:
  - address spaces
  - interprocess communication (IPC)
  - basic scheduling

- The approach:
  - simplifies implementation
  - provides flexibility
  - is well suited to a distributed environment
Multithreading

- Technique in which a process, executing an application, is divided into threads that can run concurrently

**Thread**
- dispatchable unit of work
- includes a processor context and its own data area to enable subroutine branching
- executes sequentially and is interruptible

**Process**
- a collection of one or more threads and associated system resources
- programmer has greater control over the modularity of the application and the timing of application related events
Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- Several processes can run in parallel
- Multiple processors are transparent to the user
  - these processors share same main memory and I/O facilities
  - all processors can perform the same functions
- The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors
SMP Advantages

- **Performance**: more than one process can be running simultaneously, each on a different processor.
- **Availability**: failure of a single process does not halt the system.
- **Incremental Growth**: performance of a system can be enhanced by adding an additional processor.
- **Scaling**: vendors can offer a range of products based on the number of processors configured in the system.
(a) Interleaving (multiprogramming, one processor)

(b) Interleaving and overlapping (multiprocessing; two processors)

Figure 2.12 Multiprogramming and Multiprocessing
## Distributed Operating System

- Provides the illusion of
  - a single main memory space
  - single secondary memory space
  - unified access facilities
- State of the art for distributed operating systems lags that of uniprocessor and SMP operating systems

## Object-Oriented Design

- Used for adding modular extensions to a small kernel
- Enables programmers to customize an operating system without disrupting system integrity
- Eases the development of distributed tools and full-blown distributed operating systems
Fault Tolerance

- Refers to the ability of a system or component to continue normal operation despite the presence of hardware or software faults.
- Typically involves some degree of redundancy.
- Intended to increase the reliability of a system.
  - Typically comes with a cost in financial terms or performance.
- The extent adoption of fault tolerance measures must be determined by how critical the resource is.
Fundamental Concepts

- The basic measures are:
  - Reliability
    - $R(t)$
    - defined as the probability of its correct operation up to time $t$ given that the system was operating correctly at time $t=0$
  - Mean time to failure (MTTF)
    - mean time to repair (MTTR) is the average time it takes to repair or replace a faulty element
  - Availability
    - defined as the fraction of time the system is available to service users’ requests
MTTF = \frac{B1 + B2 + B3}{3} \quad MTTR = \frac{A1 + A2 + A3}{3}

Figure 2.13  System Operational States
## Availability Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Availability</th>
<th>Annual Downtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Fault Tolerant</td>
<td>0.99999</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Fault Resilient</td>
<td>0.9999</td>
<td>53 minutes</td>
</tr>
<tr>
<td>High Availability</td>
<td>0.999</td>
<td>8.3 hours</td>
</tr>
<tr>
<td>Normal Availability</td>
<td>0.99 - 0.995</td>
<td>44-87 hours</td>
</tr>
</tbody>
</table>

Table 2.4 Availability Classes
Fault Categories

- **Permanent**
  - a fault that, after it occurs, is always present
  - the fault persists until the faulty component is replaced or repaired

- **Temporary**
  - a fault that is not present all the time for all operating conditions
  - can be classified as
    - Transient – a fault that occurs only once
    - Intermittent – a fault that occurs at multiple, unpredictable times

Spatial (physical) redundancy

involves the use of multiple components that either perform the same function simultaneously or are configured so that one component is available as a backup in case of the failure of another component

Temporal redundancy

involves repeating a function or operation when an error is detected effective with temporary faults but not useful for permanent faults

Information redundancy

provides fault tolerance by replicating or coding data in such a way that bit errors can be both detected and corrected
A number of techniques can be incorporated into OS software to support fault tolerance:

- process isolation
- concurrency
- virtual machines
- checkpoints and rollbacks
Symmetric Multiprocessor OS Considerations

- A multiprocessor OS must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors.

- **Key design issues:**
  
<table>
<thead>
<tr>
<th>Simultaneous concurrent processes or threads</th>
<th>Scheduling</th>
<th>Synchronization</th>
<th>Memory management</th>
<th>Reliability and fault tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously.</td>
<td>Any processor may perform scheduling, which complicates the task of enforcing a scheduling policy.</td>
<td>With multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronization.</td>
<td>The reuse of physical pages is the biggest problem of concern.</td>
<td>The OS should provide graceful degradation in the face of processor failure.</td>
</tr>
</tbody>
</table>
The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantial on-chip resources efficiently.

Potential for parallelism exists at three levels:

- Hardware parallelism within each core processor, known as instruction level parallelism.
- Potential for multiprogramming and multithreaded execution within each processor.
- Potential for a single application to execute in concurrent processes or threads across multiple cores.
Grand Central Dispatch

- Developer must decide what pieces can or should be executed simultaneously or in parallel

Grand Central Dispatch (GCD)

- implemented in Mac Os X 10.6
- helps a developer once something has been identified that can be split off into a separate task
- thread pool mechanism
- allows anonymous functions as a way of specifying tasks
Virtual Machine Approach

- Allows one or more cores to be dedicated to a particular process and then leave the processor alone to devote its efforts to that process.

- Multicore OS could then act as a hypervisor that makes a high-level decision to allocate cores to applications but does little in the way of resource allocation beyond that.
Figure 2.14  Windows Architecture

- Lsass = local security authentication server
- POSIX = portable operating system interface
- GDI = graphics device interface
- DLL = dynamic link libraries

Colored area indicates Executive

User mode
Kernel mode

System support processes
- Service control manager
- Lsass
- Winlogon

Session manager

Service processes
- SVHost.exe
- Winmgmt.exe
- Spooler

System service dispatcher

Application
- Task manager
- Windows Explorer
- User
- Subsystem DLLs

Environment subsystems
- POSIX

User application

System threads

Ntdll.dll

(Kernel-mode callable interfaces)

I/O manager
- File system cache
- Object manager
- Plug and play
- Power manager
- Security reference monitor
- Virtual memory
- Processes and threads
- Configuration manager (registry)

Win32 USER, GDI

Local procedure call

Win32

Graphics drivers

Device and file system drivers

Hardware abstraction layer (HAL)

Hardware abstraction layer (HAL)
Kernel-Mode Components of Windows

- Executive
  - contains the core OS services

- Kernel
  - controls execution of the processors

- Hardware Abstraction Layer (HAL)
  - maps between generic hardware commands and responses and those unique to a specific platform

- Device Drivers
  - dynamic libraries that extend the functionality of the Executive

- Windowing and Graphics System
  - implements the GUI functions
Four basic types are supported by Windows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special System Processes</td>
<td>• user-mode services needed to manage the system</td>
</tr>
<tr>
<td>Service Processes</td>
<td>• the printer spooler, event logger, and user-mode components that cooperate with device drivers, and various network services</td>
</tr>
<tr>
<td>Environment Subsystems</td>
<td>• provide different OS personalities (environments)</td>
</tr>
<tr>
<td>User Applications</td>
<td>• executables (EXEs) and DLLs that provide the functionality users run to make use of the system</td>
</tr>
</tbody>
</table>
Client/Server Model

- Windows OS services, environmental subsystems, and applications are all structured using the client/server model
- Common in distributed systems, but can be used internal to a single system
- Processes communicate via RPC

Advantages:
- it simplifies the Executive
- it improves reliability
- it provides a uniform means for applications to communicate with services via RPCs without restricting flexibility
- it provides a suitable base for distributed computing
Threads and SMP

Two important characteristics of Windows are its support for threads and for symmetric multiprocessing (SMP):

- OS routines can run on any available processor, and different routines can execute simultaneously on different processors.
- Windows supports the use of multiple threads of execution within a single process. Multiple threads within the same process may execute on different processors simultaneously.
- Server processes may use multiple threads to process requests from more than one client simultaneously.
- Windows provides mechanisms for sharing data and resources between processes and flexible interprocess communication capabilities.
Windows Objects

- Windows draws heavily on the concepts of object-oriented design
- Key object-oriented concepts used by Windows are:

  - Encapsulation
  - Object class and instance
  - Inheritance
  - Polymorphism
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Procedure Call</td>
<td>Used to break into the execution of a specified thread and to cause a procedure to be called in a specified processor mode.</td>
</tr>
<tr>
<td>Deferred Procedure Call</td>
<td>Used to postpone interrupt processing to avoid delaying hardware interrupts. Also used to implement timers and inter-processor communication</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Used to connect an interrupt source to an interrupt service routine by means of an entry in an Interrupt Dispatch Table (IDT). Each processor has an IDT that is used to dispatch interrupts that occur on that processor.</td>
</tr>
<tr>
<td>Process</td>
<td>Represents the virtual address space and control information necessary for the execution of a set of thread objects. A process contains a pointer to an address map, a list of ready threads containing thread objects, a list of threads belonging to the process, the total accumulated time for all threads executing within the process, and a base priority.</td>
</tr>
<tr>
<td>Thread</td>
<td>Represents thread objects, including scheduling priority and quantum, and which processors the thread may run on.</td>
</tr>
<tr>
<td>Profile</td>
<td>Used to measure the distribution of run time within a block of code. Both user and system code can be profiled.</td>
</tr>
</tbody>
</table>

**Table 2.5  Windows Kernel Control Objects**
Traditional UNIX Systems

- Were developed at Bell Labs and became operational on a PDP-7 in 1970
- Incorporated many ideas from Multics
- PDP-11 was a milestone because it first showed that UNIX would be an OS for all computers
- Next milestone was rewriting UNIX in the programming language C
  - demonstrated the advantages of using a high-level language for system code
- Was described in a technical journal for the first time in 1974
- First widely available version outside Bell Labs was Version 6 in 1976
- Version 7, released in 1978 is the ancestor of most modern UNIX systems
- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution)
Figure 2.15 General UNIX Architecture
Figure 2.16 Traditional UNIX Kernel
Common Facilities

- virtual memory framework
- exec switch
- file mappings
- anonymous mappings
- block device switch
- disk driver
- tape driver
- network driver
- tty driver
- vnode/vfs interface
- STREAMS
- scheduler framework
- NFS
- FFS
- s5fs
- RFS
- time-sharing processes
- system processes
- system processes

Figure 2.17 Modern UNIX Kernel [VAHA96]
System V Release 4 (SVR4)

- Developed jointly by AT&T and Sun Microsystems
- Combines features from SVR3, 4.3BSD, Microsoft Xenix System V, and SunOS
- New features in the release include:
  - real-time processing support
  - process scheduling classes
  - dynamically allocated data structures
  - virtual memory management
  - virtual file system
  - preemptive kernel
- Berkeley Software Distribution

- 4.xBSD is widely used in academic installations and has served as the basis of a number of commercial UNIX products

- 4.4BSD was the final version of BSD to be released by Berkeley
  - major upgrade to 4.3BSD
  - includes
    - a new virtual memory system
    - changes in the kernel structure
    - several other feature enhancements

- FreeBSD
  - one of the most widely used and best documented versions
  - popular for Internet-based servers and firewalls
  - used in a number of embedded systems
  - Mac OS X is based on FreeBSD 5.0 and the Mach 3.0 microkernel
Solaris 10

- Sun’s SVR4-based UNIX release
- Provides all of the features of SVR4 plus a number of more advanced features such as:
  - a fully preemptable, multithreaded kernel
  - full support for SMP
  - an object-oriented interface to file systems
- Most widely used and most successful commercial UNIX implementation
LINUX Overview

- Started out as a UNIX variant for the IBM PC
- Linus Torvalds, a Finnish student of computer science, wrote the initial version
- Linux was first posted on the Internet in 1991
- Today it is a full-featured UNIX system that runs on several platforms
- Is free and the source code is available
- Key to success has been the availability of free software packages
- Highly modular and easily configured
Modular Monolithic Kernel

- Includes virtually all of the OS functionality in one large block of code that runs as a single process with a single address space
- All the functional components of the kernel have access to all of its internal data structures and routines
- Linux is structured as a collection of modules

Loadable Modules

- Relatively independent blocks
- A module is an object file whose code can be linked to and unlinked from the kernel at runtime
- A module is executed in kernel mode on behalf of the current process
- Have two important characteristics:
  - dynamic linking
  - stackable modules
Figure 2.18 Example List of Linux Kernel Modules
signals system calls
processes & scheduler
virtual memory
system calls
processes & scheduler
virtual memory
processes & scheduler
virtual memory

traps & faults
physical memory
interrupts
char device drivers
block device drivers
file systems
network protocols
network device drivers

CPU
system memory
terminal
disk
network interface controller

Figure 2.19 Linux Kernel Components
# Linux Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>Terminal hangup</td>
<td>SIGCONT</td>
<td>Continue</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Keyboard quit</td>
<td>SIGTSTP</td>
<td>Keyboard stop</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>Trace trap</td>
<td>SIGTTOU</td>
<td>Terminal write</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>Bus error</td>
<td>SIGXCPU</td>
<td>CPU limit exceeded</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>Kill signal</td>
<td>SIGVTALRM</td>
<td>Virtual alarm clock</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>Segmentation violation</td>
<td>SIGWINCH</td>
<td>Window size unchanged</td>
</tr>
<tr>
<td>SIGPIPT</td>
<td>Broken pipe</td>
<td>SIGPWR</td>
<td>Power failure</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Termination</td>
<td>SIGRTMIN</td>
<td>First real-time signal</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>Child status unchanged</td>
<td>SIGRTMAX</td>
<td>Last real-time signal</td>
</tr>
</tbody>
</table>

**Table 2.6 Some Linux Signals**
### Filesystem related

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>Close a file descriptor.</td>
</tr>
<tr>
<td>link</td>
<td>Make a new name for a file.</td>
</tr>
<tr>
<td>open</td>
<td>Open and possibly create a file or device.</td>
</tr>
<tr>
<td>read</td>
<td>Read from file descriptor.</td>
</tr>
<tr>
<td>write</td>
<td>Write to file descriptor</td>
</tr>
</tbody>
</table>

### Process related

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execve</td>
<td>Execute program.</td>
</tr>
<tr>
<td>exit</td>
<td>Terminate the calling process.</td>
</tr>
<tr>
<td>getpid</td>
<td>Get process identification.</td>
</tr>
<tr>
<td>setuid</td>
<td>Set user identity of the current process.</td>
</tr>
<tr>
<td>ptrace</td>
<td>Provides a means by which a parent process may observe and control the</td>
</tr>
<tr>
<td></td>
<td>execution of another process, and examine and change its core image and</td>
</tr>
<tr>
<td></td>
<td>registers.</td>
</tr>
</tbody>
</table>

### Scheduling related

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_getparam</td>
<td>Sets the scheduling parameters associated with the scheduling policy for</td>
</tr>
<tr>
<td></td>
<td>the process identified by <code>pid</code>.</td>
</tr>
<tr>
<td>sched_get_priority_max</td>
<td>Returns the maximum priority value that can be used with the scheduling</td>
</tr>
<tr>
<td></td>
<td>algorithm identified by <code>policy</code>.</td>
</tr>
<tr>
<td>sched_setscheduler</td>
<td>Sets both the scheduling policy (e.g., FIFO) and the associated</td>
</tr>
<tr>
<td></td>
<td>parameters for the process <code>pid</code>.</td>
</tr>
<tr>
<td>sched_rr_get_interval</td>
<td>Writes into the timespec structure pointed to by the parameter <code>tp</code> the</td>
</tr>
<tr>
<td></td>
<td>round robin time quantum for the process <code>pid</code>.</td>
</tr>
<tr>
<td>sched_yield</td>
<td>A process can relinquish the processor voluntarily without blocking</td>
</tr>
<tr>
<td></td>
<td>via this system call. The process will then be moved to the end of the</td>
</tr>
<tr>
<td></td>
<td>queue for its static priority and a new process gets to run.</td>
</tr>
</tbody>
</table>

**Table 2.7  Some Linux System Calls (page 1 of 2)**
### Interprocess Communication (IPC) related

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgrcv</td>
<td>A message buffer structure is allocated to receive a message. The system call then reads a message from the message queue specified by msqid into the newly created message buffer.</td>
</tr>
<tr>
<td>semctl</td>
<td>Performs the control operation specified by cmd on the semaphore set semid.</td>
</tr>
<tr>
<td>semop</td>
<td>Performs operations on selected members of the semaphore set semid.</td>
</tr>
<tr>
<td>shmat</td>
<td>Attaches the shared memory segment identified by shmid to the data segment of the calling process.</td>
</tr>
<tr>
<td>shmctl</td>
<td>Allows the user to receive information on a shared memory segment, set the owner, group, and permissions of a shared memory segment, or destroy a segment.</td>
</tr>
</tbody>
</table>

### Socket (networking) related

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>Assigns the local IP address and port for a socket. Returns 0 for success and -1 for error.</td>
</tr>
<tr>
<td>connect</td>
<td>Establishes a connection between the given socket and the remote socket associated with sockaddr.</td>
</tr>
<tr>
<td>gethostname</td>
<td>Returns local host name.</td>
</tr>
<tr>
<td>send</td>
<td>Send the bytes contained in buffer pointed to by *msg over the given socket.</td>
</tr>
<tr>
<td>setsockopt</td>
<td>Sets the options on a socket</td>
</tr>
</tbody>
</table>

### Miscellaneous

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsync</td>
<td>Copies all in-core parts of a file to disk, and waits until the device reports that all parts are on stable storage.</td>
</tr>
<tr>
<td>time</td>
<td>Returns the time in seconds since January 1, 1970.</td>
</tr>
<tr>
<td>vhangup</td>
<td>Simulates a hangup on the current terminal. This call arranges for other users to have a &quot;clean&quot; tty at login time.</td>
</tr>
</tbody>
</table>

Table 2.7 Some Linux System Calls (page 2 of 2)
Android Operating System

- A Linux-based system originally designed for touchscreen mobile devices such as smartphones and tablet computers
- The most popular mobile OS
- Development was done by Android Inc., which was bought by Google in 2005
- 1st commercial version (Android 1.0) was released in 2008
- Most recent version is Android 4.3 (Jelly Bean)
- The Open Handset Alliance (OHA) was responsible for the Android OS releases as an open platform
- The open-source nature of Android has been the key to its success
## Figure 2.20 Android Software Architecture

<table>
<thead>
<tr>
<th>Applications</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Dialer</td>
<td>SMS/MMS</td>
<td>IM</td>
</tr>
<tr>
<td>Contacts</td>
<td>Voice Dial</td>
<td>Email</td>
<td>Calendar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Media Player</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Albums</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Camera</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alarm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Calculator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Framework</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Package Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telephony Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content Providers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View System</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notification Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMPP Service</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Libraries</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL/ES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeType</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media Framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LibWebCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Libc</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Android Runtime</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Libraries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dalvik Virtual Machine</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linux Kernel</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bluetooth Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash Memory Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binder (IPC) Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keypad Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFi Driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Drivers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Management</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Implementation:**
- Applications, Application Framework: Java
- System Libraries, Android Runtime: C and C++
- Linux Kernel: C
Application Framework

- Provides high-level building blocks accessible through standardized API’s that programmers use to create new apps
  - architecture is designed to simplify the reuse of components

- Key components:

  **Activity Manager**
  - Manages lifecycle of applications
  - Responsible for starting, stopping, and resuming the various applications

  **Window Manager**
  - Java abstraction of the underlying Surface Manager
  - Allows applications to declare their client area and use features like the status bar

  **Package Manager**
  - Installs and removes applications

  **Telephony Manager**
  - Allows interaction with phone, SMS, and MMS services
Application Framework
(cont.)

- Key components: (cont.)
  - Content Providers
    - these functions encapsulate application data that need to be shared between applications such as contacts
  - Resource Manager
    - manages application resources, such as localized strings and bitmaps
  - View System
    - provides the user interface (UI) primitives as well as UI Events
  - Location Manager
    - allows developers to tap into location-based services, whether by GPS, cell tower IDs, or local Wi-Fi databases
  - Notification Manager
    - manages events, such as arriving messages and appointments
  - XMPP
    - provides standardized messaging functions between applications
System Libraries

- Collection of useful system functions written in C or C++ and used by various components of the Android system
- Called from the application framework and applications through a Java interface
- Exposed to developers through the Android application framework
- Some of the key system libraries include:
  - Surface Manager
  - OpenGL
  - Media Framework
  - SQL Database
  - Browser Engine
  - Bionic LibC
Every Android application runs in its own process with its own instance of the Dalvik virtual machine (DVM)

DVM executes files in the Dalvik Executable (.dex) format

Component includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language

To execute an operation the DVM calls on the corresponding C/C++ library using the Java Native Interface (JNI)
Figure 2.21 Android System Architecture
Activities

- An activity is a single visual user interface component, including things such as menu selections, icons, and checkboxes.

- Every screen in an application is an extension of the Activity class.

- Use Views to form graphical user interfaces that display information and respond to user actions.
Power Management

Alarms

- Implemented in the Linux kernel and is visible to the app developer through the AlarmManager in the RunTime core libraries
- Is implemented in the kernel so that an alarm can trigger even if the system is in sleep mode
  - this allows the system to go into sleep mode, saving power, even though there is a process that requires a wake up

Wakelocks

- Prevents an Android system from entering into sleep mode
- These locks are requested through the API whenever an application requires one of the managed peripherals to remain powered on
- An application can hold one of the following wakelocks:
  - Full_Wake_Lock
  - Partial_Wake_Lock
  - Screen_Dim_Wake_Lock
  - Screen_Bright_Wake_Lock
Summary

- Operating system objectives and functions
  - User/computer interface
  - Resource manager
- Evolution of operating systems
  - Serial processing
  - Simple/multiprogrammed/time-sharing batch systems
- Major achievements
- Developments leading to modern operating systems
- Fault tolerance
  - Fundamental concepts
  - Faults
  - OS mechanisms
- OS design considerations for multiprocessor and multicore
- Microsoft Windows overview
- Traditional Unix systems
  - History/description
- Modern Unix systems
  - System V Release 4 (SVR4)
  - BSD
  - Solaris 10
- Linux
  - History
  - Modular structure
  - Kernel components
- Android
  - Software/system architecture
  - Activities
  - Power management