Introduction. System Calls

- Overview of the major operating systems components
- Interrupts, Dual mode, System calls
- Self-study material: Computer Systems

What is an Operating System (OS)?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs in a well-defined environment.
  - Make the computer system convenient to use.
  - Administrate system resources.
  - Enable efficient use of the computer hardware.
- An operating system provides an environment within which other programs can do useful work; the OS does not perform any "useful" function itself.

Where are OSs found?

General purpose systems
- Unix
- Sun/Oracle Solaris
- HP-UX
- Linux
- Windows 95/98/2000, NT/XP/Vista/7
- Mac OS X

Embedded systems
- Android
- iOS
- Symbian
- Windows CE, Mobile, Phone
- Embedded Linux
- RT-Linux
- VxWorks
- OSE
- QNX

Operating System Definition

- OS is a resource allocator
  - Manages all resources of a computer system
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer
- No universally accepted definition
- "The one program running at all times on the computer" is called the kernel.
  Everything else is either a system program (ships with the operating system) or an application program.

Computer System Structure

- Computer system can be divided into 4 components:
  - Hardware provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system controls and coordinates use of hardware among various applications and users
  - Application programs define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, games
  - Users
    - People, machines, other computers
Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connected through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices, competing for memory cycles

Background: Interrupt

- Program execution (von-Neumann cycle) by a processor

CPU – I/O Device Interaction (1)

- I/O devices and the CPU can execute concurrently.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.

CPU – I/O Device Interaction (2)

- DMA = Direct Memory Access
  - allows for parallel activity of CPU and I/O data transfer

I/O Interaction using Interrupt

- Example: Read from the keyboard (KBD)
Interrupt (1)

- Interrupt transfers control to an interrupt service routine, generally through the interrupt vector (IRV), a branch table that contains the start addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- How to determine which type of interrupt has occurred?
  - polling
  - vectored interrupt system: interrupt number indexes IRV

Interrupt (2)

- A trap is a software-generated interrupt caused either by an error or a user request.
  - Examples: Division by zero; Request for OS service
- An operating system is interrupt driven.
  - The OS preserves the state of the CPU by saving registers and the program counter.

Interrupt Timeline

for a single process doing output

CPU user process executing

I/O Interrupt processing

I/O device idling

I/O request transfer done I/O request transfer done

Uniprogramming

- Process execution time:
  - CPU: 10 + 10 time units
  - I/O: 100 + 100 time units
- I.e., I/O intensive (200/220 = 90.9%), CPU utilization 9.1%

Operating System Operations

- Dual mode, system calls
- CPU management
  - Uniprogramming, Multiprogramming, Multitasking
  - Process management
  - Memory management
  - File system and mass storage management
  - Protection and security

Multiprogramming

- needed for efficiency
  - Multiprogramming organizes jobs (code and data)
  - so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling
  - When it has to wait (e.g., for I/O), OS switches to another job

Single user with single program cannot keep CPU and I/O devices busy at all times.
Multiprogramming with three programs

- A: Running, Waiting (printer), Running, Waiting
- B: Running, Waiting (disk), Running, Waiting
- C: Running, Waiting (network), Running, Waiting

Combined:
- Running, Running, Running, Waiting, Running, Running, Running, Waiting

Timesharing (Multitasking)
- Extension of multiprogramming: CPU switches jobs so frequently that users can interact with each job while it is running
  - For interactive computer systems, the response time should be short (< 1 second)
  - Each user has at least one program executing in memory
  - If several jobs ready to run at the same time
  - CPU scheduling
  - If processes don’t fit in memory, swapping moves them in and out to run
  - Virtual memory allows execution of processes not completely in memory

CPU time sharing using timer interrupt
- Timer to prevent infinite loop / process hogging resources
  - Set up to interrupt the computer after specific period
  - System decrements counter at clock ticks
  - When counter = zero, generate an interrupt
  - So, OS regains control and can reschedule or terminate a program that exceeds allotted time

Process Management
- A process is a program in execution.
  - A unit of work within the system.
  - Program is a passive entity, process is an active entity.
  - Process needs resources to accomplish its task
    - CPU, memory, I/O, files
    - Initialization data
  - Process termination requires reclaim of any reusable resources
  - Single-threaded process: has one program counter specifying location of next instruction to execute
    - Process executes instructions sequentially, one at a time, until completion
  - Multi-threaded process: has one program counter per thread
    - Typically, a system has many processes (some user, some system pr.) running concurrently on one or more CPUs
    - Concurrency by multiplexing the CPUs among the processes / threads

Memory Management
- Memory: A large array of words or bytes, each with its own address
  - Primary storage – directly accessible from CPU
    - shared by CPU and I/O devices
    - a volatile storage medium
    - All data in memory before and after processing
    - All instructions in memory in order to execute
  - Memory management determines what is in memory when.
    - Optimizing CPU utilization and memory utilization
  - OS memory management activities
    - Keeping track of which parts of memory are currently being used and by whom
    - Allocating space for processes (or parts thereof) and data to move in and out of memory
    - Allocating and deallocating memory space as needed

Process Management Activities
- The operating system is responsible for:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

  - Lecture on Processes and Threads
  - Lecture on CPU Scheduling
  - Lectures on Synchronization
  - Lecture on Deadlocks

Memory Management
- Lectures on Memory Management and Virtual Memory
Mass-Storage Management (1)
- Disks used to store data that do not fit in main memory or data that must be kept for a "long" period of time.
  - Secondary storage
- OS activities:
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Critical for system performance
- Some storage need not be fast
  - Tertiary storage
    - optical storage, magnetic tape...
    - Still must be managed

Mass-Storage Management (2)
- OS provides uniform, logical view of information storage
  - Abstracts from physical to logical storage unit: file
  - Each medium (disk, tape) has different properties:
    - access speed, data transfer rate, sequential/random access
- OS File-System management
  - Files usually organized into directories
  - Access control
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files to tertiary storage

Protection and Security
- Protection – any mechanism for controlling access of processes or users to resources defined by the OS
- Security – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs)
    - associated with all files, processes of that user
  - Group identifier (group ID)

System Calls
- System call mechanism
- System call API
- Passing parameters
- Types of system calls

Dual mode, System calls
- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode (supervisor mode, privileged mode)
    - Privileged instructions only executable in kernel mode
    - System call changes mode to kernel, on return resets it to user
    - Mode bit provided by hardware
Dual-Mode Operation (Cont.)

- When an interrupt or fault occurs, hardware switches to kernel mode.
- System calls – call OS service

![Kernel/User Mode Switch](image)

Example of System Calls

- System call sequence to copy contents of one file to another

![System Call Sequence Diagram](image)

System Call API

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, Mac OS X), and
  - Java API for the Java virtual machine (JVM)

Remark: the system-call names used here and in the book are generic.

Example of a System Call API

- ReadFile() function in Win32 API (function for reading from a file)

![Example System Call API](image)

System Call API Implementation

- System call implementation is hardware-specific, e.g. special trap instruction with a system call number passed in a register, indexing the interrupt vector (branch table)
- System call interface (usually, in C)
  - invokes the intended system call in OS kernel and returns status of the system call and any return values
- Advantage:
  - Caller does not need to know anything about how the system call is implemented
  - Most details of OS interface hidden from programmer by API

System Call API – OS Relationship

- User application
- System call interface
- Open system call
- Implementation of open() system call
- Return
Standard C Library Example

- C program invoking printf(), which calls write() system call

```c
#include <stdio.h>
int main()
{
    printf("Greetings\n");
    return 0;
}
```

**System Call Parameter Passing**

Three general methods used to pass parameters to syscalls:
- Simplest: pass the parameters in registers
  - but number of parameter registers is limited
- Parameters stored in a block in memory, and address of block passed as a parameter in a register
  - This approach taken by Linux and Solaris
- Parameters pushed onto the stack by the program and popped off the stack by the operating system

- Block and stack methods do not limit the number or length of parameters being passed

**System Call Parameter Passing via Block**

**Types of System Calls**

- Process control: load, execute, end, abort, create, terminate, wait...
- Memory allocation and deallocation
- File management: open, close, create, delete, read, write, get/set attributes...
- Device management: request / release device, read, write, ...
- Information maintenance: get / set time, date, system data, process / file attributes
- Communications: create / delete connection, send, receive, ...

**Solaris 10 dtrace**

```
dtrace: script `/all.d' matched 52377 probe cpu-fraction
0 -> _XEventQueued U
0 -> _XEventQueued U
0 -> _XITransBytesReadable U
0 -> _XITransBytesReadable U
0 -> _XITransSocketBytesReadable U
0 -> _XITransSocketBytesReadable U
0 -> _Ioctl K
0 -> __ioctl U
0 -> _getpid K
0 -> _getuid K
0 -> _getgid K
0 -> _getuid K
0 -> _getgid K
0 -> _release K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
0 -> _clear_active_fd K
```

**System Programs**

- provide a convenient environment for program development and execution.
- File management
- Status information
- File modification
- Programming language support: Compilers, assemblers, debuggers...
- Program loading and execution
- Communications: Message passing, e-mail, web browser, ...

- Some of them are simply user interfaces to system calls; others are considerably more complex
- Most users’ view of the operation system is defined by system programs, not the actual system calls
Summary

- **Operating System** = OS Kernel + System Programs
  - Mediates all accesses to system resources
  - **Interrupt-driven**
    - Error handling
    - Controlled access to system resources, e.g.
      - I/O devices, DMA
      - CPU time sharing
    - ...
  - **Dual-Mode** (user mode, kernel mode)
    - System Call API for portability

---

**Computer Systems and Environments**

- Stand-alone desktop computer
- Client-server systems
- Parallel systems
- Distributed systems
- Peer-to-Peer computing systems
- Real-time systems
- Handheld systems
- Embedded systems

---

**Client-Server Computing**

- Servers respond to requests by clients
  - Remote procedure call – also across machine boundaries via network
  - **Compute server**:
    compute an action requested by the client
  - **File server**:
    Interface to file system (read, create, update, delete)

---

**Parallel Systems (1)**

- Multiprocessor systems with more than one CPU in close communication.
  - Soon the default, even for desktop machines (multi-core / CMP)
  - **Tightly coupled system** (aka. shared-memory system, multiprocessor)
    - processors share memory and a clock;
    - communication usually takes place through the shared memory.
  - Advantages of parallel systems:
    - Increased **throughput**
    - Economical
      - Scalability of performance
      - Multiprocessor system vs multiple single-processor system (reduction of hardware such as disks, controllers etc)
    - Increased reliability
      - graceful degradation (fault tolerance, ...)
      - fail-soft systems (replication, ...)

---

**Stand-alone desktop computer**

- Blurring over time
- Office environment
  - PCs connected to a network, terminals attached to mainframe, or minicomputers providing batch and timesharing
  - Now portals allowing networked and remote systems access to same resources
- Home networks
  - Used to be single system, then modems
  - Now firewalled, networked
Parallel Systems (2)

- **Symmetric multiprocessing (SMP)**
  - Each processor runs an identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP

- **Asymmetric multiprocessing**
  - Each processor is assigned a specific task; a master processor schedules and allocates work to slave processors.
  - More common in special-purpose systems (e.g., embedded MP-SoC)

**Remark:** the notion of “processor” is relative:
A traditional PC is normally considered to only have one CPU, but it usually has a graphics processor, a communication processor etc, and this is not considered a multi-processing system.

Parallel Systems (3)

- **Speed-up of a single application by parallel processing?**
  - Requires parallelisation / restructuring of the program
  - Or explicitly parallel algorithms
  - Used in High-Performance Computing for numerically intensive applications (weather forecast, simulations, …)
  - Now ubiquitous problem due to multicore

- **Multicomputer** ("distributed memory parallel computer")
  - loosely coupled
  - can be a more economic and scalable alternative to SMP’s
  - but more cumbersome to program (message passing)
  - Example: “Beowulf clusters”

**More in TDDC78 – Programming parallel computers**

Distributed Systems (1)

- **Loosely coupled system**
  - Each processor has its own local memory
  - Processors communicate with one another through various communications lines, such as high-speed buses or telephone lines (LAN, WAN, MAN, Bluetooth, …).
  - Distribute the computation among several physical processors.
  - Advantages of distributed systems:
    - Resource sharing
    - Computation speed up
    - Adaptability: load sharing (migration of jobs)
    - Fault tolerance

Distributed Systems (2)

- **Network Operating System**
  - Provides file sharing, e.g., NFS - Network File System
  - Provides communication scheme
  - Runs independently from other computers on the network

- **Distributed Operating System**
  - Less autonomy between computers
  - Gives the impression there is a single operating system controlling the network.

**More about this in TDDD25 Distributed Systems**

Real-Time Systems

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.

- Well-defined tasks with fixed time constraints.

- **Hard real-time systems.**
  - Conflicts with time-sharing systems, not supported by general-purpose OSs.

- **Soft real-time systems**
  - Limited utility in industrial control or robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced OS features.

**More in TDDD07 Real-time Systems; see also Chapter 19 in the book**

Storage Hierarchy

- **Storage systems organized in hierarchy.**
  - Speed
  - Cost
  - Volatility

- **Caching** – keeping copies of some data temporarily in faster but smaller storage, closer to CPU
**Background: Storage Hierarchy**

Performance of various levels of storage:

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>disk storage</td>
</tr>
<tr>
<td>Typical size (GB)</td>
<td>&lt; 1</td>
<td>&gt; 16</td>
<td>16</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>Technology</td>
<td>semiconductor</td>
<td>semiconductor</td>
<td>semiconductor</td>
<td>semiconductor</td>
</tr>
<tr>
<td>Access time (ms)</td>
<td>0.25 – 0.5</td>
<td>0.5 – 5</td>
<td>5 – 25</td>
<td>80 – 250</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 – 100,000</td>
<td>5000 – 10,000</td>
<td>1000 – 5000</td>
<td>20 – 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>CD or tape</td>
</tr>
</tbody>
</table>

**Background: Caching**

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

**Memory Consistency Problems**

- Multitasking environments must be careful to use the most recent value, no matter where it is stored in the storage hierarchy
- Example: Migration of integer A from disk to register creates 3 copies
- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
  - More on this in TDDC78 – Programming parallel computers and TDDD56 – Multicore and GPU programming
- Distributed environment situation even more complex
  - Similar consistency problem: Several copies of a datum can exist
  - Various solutions exist, covered in [SGG7] Ch. 17 / [SGG8] Ch. 16-18, and in TDDD25 – Distributed Systems