CS3701 - Operating Systems

Chapter 1 - Introduction to Operating Systems

Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY - 2025/2026

Prepared by Dr. Mahdi Khemakhem, Reviewed by Dr. Essra Aldessouky
Updated on January 22, 2026

Outline

1. What is an Operating System?

2. Computer System
2.1 Structure, Organization and Architecture
2.2 Startup, Interrupts, and Operations
2.3 Monoprogramming and Multiprogramming Systems
2.4 Computer System Types and Environments
2.5 Key Takeaways

3. Operating System Operations
3.1 OS Startup and Interrupts
3.2 Dual-Mode Operation
3.3 Key Takeaways

4. Operating System Activities
4.1 Processes Management
4.2 Memory Management
4.3 File System Management
4.4 Mass Storage Management
4.5 I/O System Management
4.6 Protection and Security
4.7 Key Takeaways

Outline (Contd.)

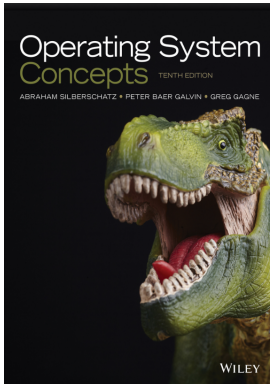
5. Operating System Services
5.1 General Overview
5.2 System Calls
5.3 System Programs
5.4 Why Applications are OS Dependent?
5.5 Free vs. not Free, Open Source vs. Closed Source OS
5.6 Key Takeaways

Textbook Reference

Required Reading

Refer to Chapters 1 and 2 of the textbook:

- **Title:** *Operating System Concepts*
- **Authors:** Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne
- **Publisher:** John Wiley & Sons
- **Edition:** 10th Edition (2018)



Outline

1. What is an Operating System?
2. Computer System
3. Operating System Operations
4. Operating System Activities
5. Operating System Services

What is an Operating System?

Definition

An **Operating System (OS)** is a **program** that acts as an *intermediary* between the **user** and the **computer hardware**.

Operating System Goals

- **Provide** a convenient environment for users to execute **programs**.
- **Manage** the **computer hardware** resources efficiently.
- **Control and coordinate** the use of *hardware* among various **application programs** and **users**.

Outline

1. What is an Operating System?

2. Computer System
2.1 Structure, Organization and Architecture
2.2 Startup, Interrupts, and Operations
2.3 Monoprogramming and Multiprogramming Systems
2.4 Computer System Types and Environments
2.5 Key Takeaways

3. Operating System Operations

4. Operating System Activities

5. Operating System Services

Computer System Structure

Computer system can be divided into four components:

- **Hardware**: Physical components (CPU, memory, I/O devices, storage).
- **Operating System**: Controls and coordinates *hardware usage* among applications and users.
- **Application Programs**: Software performing specific tasks (word processors, browsers, etc.).
- **Users**: Individuals, machines, or other computers interacting with the system.

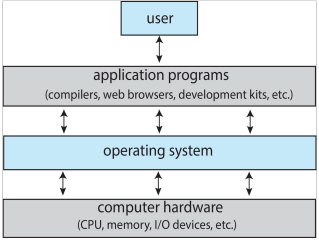
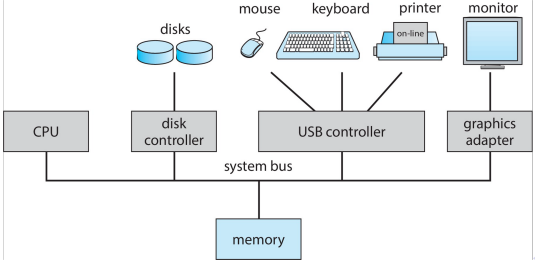


Figure 1: Computer System Structure

Computer System Organization

The computer system is physically organized as follows:

- One or more **CPUs** and **device controllers** connect through a common *bus* to access *shared memory*.
- **Concurrent execution** of CPUs and devices competing for **memory access**.
- The OS is crucial for managing *concurrent access* and ensuring **resource protection**.



Storage Devices Structure/Hierarchy

Storage devices are classified based on:

- **Speed**: *Faster* storage is more expensive and typically *smaller* in capacity.
- **Volatility**: **Volatile** (RAM) loses data when power is off; **non-volatile** (hard drives) retains data.
- **Cost**: Faster and more reliable storage devices are generally *more costly*.

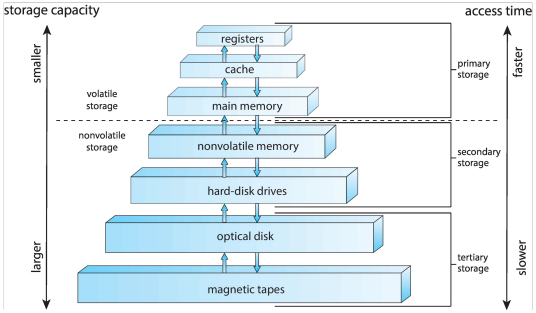


Figure 3: Storage Devices Classification

Analogy for Storage Hierarchy

Think of storage hierarchy like an office: **Registers** are **items on your desk** (fastest access), **Main Memory** is a **bookshelf in your office** (moderate access), and **Disk Storage** is a **storage room in another building** (slowest access).

Computer System Architecture

- **Single Processor Systems**: One CPU executes one instruction stream at a time.
- **Multiprocessor Systems**: Multiple CPUs share memory and I/O devices to increase throughput and reliability.
  - Also known as **parallel systems** or **tightly-coupled systems**.
  - **Advantages**: Increased throughput, economy of scale, and reliability.
  - **Types**:
    1. **Asymmetric Multiprocessing**: Each processor is assigned a specific task.
    2. **Symmetric Multiprocessing**: Each processor performs all tasks.

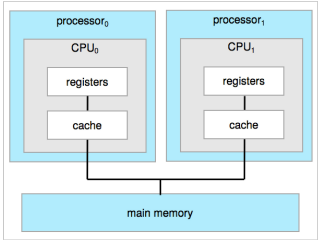


Figure 4: Multiprocessor System

Computer System Architecture (Contd.)

- **Dual-Core Systems**: A single chip with two separate processor cores that share memory and I/O devices.
- **Multicore Systems**: A single chip with multiple processor cores that share memory and I/O devices.
  - **Advantages**: Increased performance, reduced power consumption, and improved reliability.
  - **Types**:
    1. **Homogeneous Multicore**: All cores are identical.
    2. **Heterogeneous Multicore**: Cores have different architectures or capabilities.

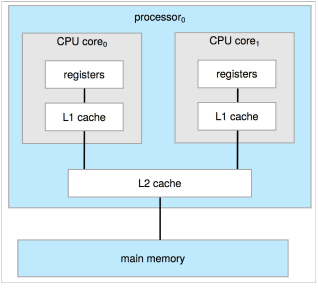


Figure 5: Dual-Core System

Computer Startup

- When powered on or rebooted, a *bootstrap program* is loaded.
- This program is stored in **ROM** or **EPROM**, known as *firmware*.
- The bootstrap program **initializes** all system aspects.
- It then loads the *OS kernel* and starts execution.

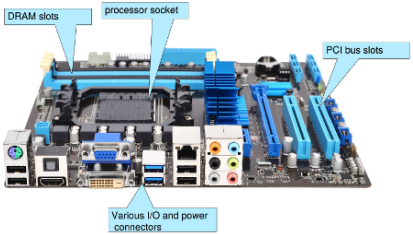


Figure 6: Computer Motherboard

Common Functions of Interrupts

- *Interrupts* are **signals** sent to CPU by **hardware** or **software** indicating an event needing **immediate attention**.
- They allow CPU to respond to **asynchronous events** (I/O operations, errors).
- When an interrupt occurs, the CPU **temporarily halts** current execution and **transfers control** to an *Interrupt Service Routine (ISR)*.
- The ISR handles the specific event that triggered the interrupt.
- After ISR completes, CPU **resumes** previous execution.

Interrupt-Driven OS

Modern computer systems use *interrupt-driven operating systems* to efficiently manage I/O operations and asynchronous events.

Computer System Operations

- The OS manages **I/O devices** through **device controllers**.
- Each device controller has a **local buffer** for data transfer.
- The CPU transfers data between **main memory** and the **local buffers**.
- I/O operations are performed between the **device** and the **local buffer**.
- Upon completion of an I/O operation, the device controller sends an **interrupt** to the CPU.
- This allows the CPU to continue executing other tasks while waiting for I/O operations to complete.

Monoprogramming Systems

- In a *monoprogramming system*, only **one program** is loaded into memory and executed at a time.
- The OS manages the execution of this **single program**.
- When the program performs I/O, it **waits** for completion before continuing.
- This leads to *inefficient CPU usage*, as CPU may be **idle** while waiting for I/O.
- **Early computer systems** used monoprogramming due to **limited memory and processing power**.

Multiprogramming Systems

- Modern computer systems use *multiprogramming*.
- In a *multiprogramming system*, **multiple programs** (processes) are loaded and executed **concurrently**.
- The OS manages execution, **switching** between programs as needed.
- When one program performs I/O, the OS **switches** to another ready program.
- This allows *efficient CPU usage*, as CPU continues executing while waiting for I/O.

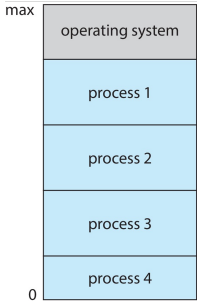


Figure 7: Memory Layout for Multiprogrammed System

Computer System Types

Computer systems can be classified into several types based on their architecture and usage:

- **Single-user systems**: Designed for one user at a time (e.g., personal computers).
- **Multi-user systems**: Allow multiple users to access the system simultaneously (e.g., mainframes, servers).
- **Time-sharing systems**: Enable multiple users to share system resources concurrently by rapidly switching between them (e.g., interactive systems).
- **Real-time systems**: Provide immediate processing and response to external events (e.g., embedded systems, industrial control systems).
- **Distributed systems**: Consist of multiple interconnected computers that work together to achieve a common goal (e.g., cloud computing, cluster computing).
- **Networked systems**: Connect multiple computers through a network to share resources and information (e.g., local area networks, wide area networks).

Computing Environments

Computing environments have evolved significantly, leading to various types of systems:

- **Standalone Systems**: Traditional personal computers that operate independently without network connectivity.
- **Client-Server Computing**: A model where clients request services from centralized servers.
- **Cloud Computing**: Provides computing resources and services over the Internet, allowing for scalability and flexibility.
- **Mobile Computing**: Involves portable devices that can connect to networks wirelessly, enabling access to information and services on the go.

Key Takeaways

Key Takeaways

- An **operating system (OS)** is an intermediary between users and computer hardware, providing a convenient environment for executing software and managing hardware resources.
- A computer system consists of four main components: **hardware**, **operating system**, **application programs**, and **users**.
- Computer systems can be organized into **single processor systems**, **multiprocessor systems**, **dual-core systems**, and **multicore systems**.
- The OS is loaded into memory during startup by a **bootstrap program** stored in ROM or EPROM.
- **Interrupts** allow the CPU to respond to asynchronous events, enabling efficient management of I/O operations.
- The majority of modern computer systems use **multiprogramming** and **interrupt-driven operating systems** to maximize CPU utilization and manage concurrent processes.

Outline

1. What is an Operating System?

2. Computer System

3. Operating System Operations
3.1 OS Startup and Interrupts
3.2 Dual-Mode Operation
3.3 Key Takeaways

4. Operating System Activities

5. Operating System Services

OS Startup and Interrupts

- The OS is **loaded into memory** during the computer startup process.
- The **OS kernel** is responsible for managing system resources and providing services to application programs.
- The **OS is interrupt-driven**, responding to both hardware and software interrupts.
- **Hardware interrupts** are generated by I/O devices to signal the completion of an operation.
- **Software interrupts**, also known as **exceptions** or **traps**, are generated by the CPU to signal errors or requests for OS services.
- **Common software interrupts include division by zero errors and system calls.**

Dual-Mode Operation

- Modern systems support *dual-mode operation* to **protect resources** and **ensure security**.
- CPU operates in two modes: *user mode* and *kernel mode*.
- In **user mode**: CPU executes applications with **limited access** to system resources.
- In **kernel mode**: CPU has **full access** to all resources and executes OS code.
- OS switches between modes during **system calls** or **interrupts**.
- This separation **prevents applications** from directly accessing *critical system resources*[1].

[1] Ressources such as memory, hardware devices, and CPU instructions that could compromise system stability or security if misused.

Dual-Mode Operation (Contd.)

- CPU uses a *mode bit* to indicate current operation mode.
- Mode bit: **0** = **user mode**, **1** = **kernel mode**.
- *Privileged instructions* can only execute in **kernel mode**.
- Attempting privileged instruction in **user mode** causes **protection violation**, generating a **software interrupt**.
- This ensures applications **cannot perform unauthorized operations** that could compromise *system stability or security*.

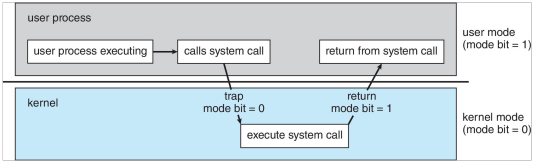


Figure 8: Transition from User to Kernel Mode

Key Takeaways

Key Takeaways

- The OS is **loaded into memory** during startup by a bootstrap program.
- The OS is **interrupt-driven**, responding to both **hardware** and **software interrupts** to manage I/O operations and system events.
- Modern computer systems support **dual-mode operation**, with **user mode** for application programs and **kernel mode** for OS code.
- This separation helps protect system resources and ensure security by preventing unauthorized access from application programs.

Outline

1. What is an Operating System?

2. Computer System

3. Operating System Operations

4. Operating System Activities
4.1 Processes Management
4.2 Memory Management
4.3 File System Management
4.4 Mass Storage Management
4.5 I/O System Management
4.6 Protection and Security
4.7 Key Takeaways

5. Operating System Services

Processes Management

- A *process* is a **program in execution**. **Program** = **passive entity**; **Process** = **active entity**.
- OS manages *processes* (instances of executing programs).
- Key functions of process management:
  - **Creation and termination**: OS creates/terminates processes as needed.
  - **Scheduling**: OS determines which process executes next (**Chapter 4**).
  - **Synchronization**: Ensures processes safely share resources (**Chapter 5**).
  - **Deadlock handling**: Detects and resolves deadlocks (**Chapter 6**).
- Systems run many processes simultaneously (**user and OS processes**). *Concurrency* achieved by **multiplexing CPU(s)**[2].

[2]Multiplexing is the process of sharing a single resource (like a CPU) among multiple processes by rapidly switching between them, giving the illusion of simultaneous execution.

Processes Management (Contd.)

- The OS maintains a **process control block (PCB)** for each process that contains important information about the process (**see Chapter 2 for details**).
- Each single-thread process has its own **program counter** and **stack**.
- The **program counter** keeps track of the next instruction to execute.
  - When a process is interrupted, the current value of the program counter is saved in the process's PCB.
  - When the process is resumed, the program counter is restored from the PCB.
- Multi-threaded processes have multiple program counters, one for each thread (**see Chapter 3 for details**).
- The **stack** contains temporary data such as function parameters, return addresses, and local variables.
- **Effective process management is crucial for maximizing CPU utilization and ensuring system responsiveness.**

Memory Management

- **To execute a program all (or part) of the instructions must be in memory.**
- The OS is responsible for managing the system's **memory resources**.
- Key functions of memory management include:
  - **Memory allocation and deallocation**: The OS allocates memory to processes when they are created and deallocates it when they terminate.
  - **Memory protection**: The OS ensures that processes cannot access memory that has not been allocated to them, preventing accidental or malicious interference (**see Chapter 7 for details**).
  - **Memory sharing**: The OS allows multiple processes to share memory regions when appropriate, improving efficiency.
  - **Virtual memory management**: The OS uses techniques such as paging and segmentation to provide each process with the illusion of a large, contiguous address space, even if physical memory is limited (**see Chapter 8 for details**).
- **Effective memory management is crucial for maximizing system performance and ensuring that processes have the resources they need to execute**

File System Management

- **A file is a collection of related information defined by its creator.**
- The OS is responsible for managing the system's **file system**.
- Key functions of file system management include:
  - **File creation and deletion**: The OS provides mechanisms for creating, deleting, and organizing files.
  - **File access and manipulation**: The OS provides APIs for reading, writing, and modifying files.
  - **File protection**: The OS enforces access control policies to ensure that only authorized users can access or modify files.
  - **File storage management**: The OS manages the allocation of disk space for files and directories, optimizing performance and minimizing fragmentation.
- **Effective file system management is crucial for ensuring data integrity, security, and efficient access to stored information.**

Mass Storage Management

- **Mass storage devices** provide long-term storage for data and programs.
- The OS is responsible for managing the system's **mass storage resources**.
- Key functions of mass storage management include:
  - **Disk scheduling**: The OS optimizes the order in which disk I/O requests are serviced to minimize seek time and improve performance.
  - **Storage allocation**: The OS manages the allocation of disk space to files and directories, ensuring efficient use of available storage.
  - **Data backup and recovery**: The OS provides mechanisms for backing up data and recovering from failures or data loss.
  - **File system organization**: The OS organizes files on disk using structures such as file allocation tables (FAT) or inodes.
- **Effective mass storage management is crucial for ensuring data availability, integrity, and efficient access to stored information.**

I/O System Management

- The OS is responsible for managing the system's **I/O devices** and **operations**.
- Key functions of I/O system management include:
  - **Device management**: The OS manages the allocation and deallocation of I/O devices to processes, ensuring fair access and preventing conflicts.
  - **I/O scheduling**: The OS optimizes the order in which I/O requests are serviced to improve overall system performance.
  - **Buffering and caching**: The OS uses techniques such as buffering and caching to improve the efficiency of I/O operations.
  - **Error handling**: The OS detects and handles errors that may occur during I/O operations, ensuring data integrity and system stability.
- **Effective I/O system management is crucial for maximizing system performance and ensuring reliable access to peripheral devices.**

Protection and Security

- The OS is responsible for ensuring the **protection** and **security** of system resources.
- Key functions of protection and security include:
  - **Access control**: The OS enforces policies that determine which users or processes can access specific resources.
  - **Authentication**: The OS verifies the identity of users before granting access to the system.
  - **Encryption**: The OS may use encryption techniques to protect sensitive data from unauthorized access.
  - **Auditing and logging**: The OS maintains logs of system activity to detect and investigate security breaches.
- **Effective protection and security measures are crucial for safeguarding system resources and ensuring the confidentiality, integrity, and availability of data.**

Key Takeaways

Key Takeaways

- The OS is responsible for managing various system resources, including **processes**, **memory**, **files**, **mass storage**, and **I/O devices**.
- Effective management of these resources is crucial for maximizing system performance, ensuring data integrity, and providing reliable access to system resources.
- The OS also plays a critical role in ensuring the **protection** and **security** of system resources, safeguarding against unauthorized access and potential threats.

Outline

1. What is an Operating System?

2. Computer System

3. Operating System Operations

4. Operating System Activities

5. Operating System Services
5.1 General Overview
5.2 System Calls
5.3 System Programs
5.4 Why Applications are OS Dependent?
5.5 Free vs. not Free, Open Source vs. Closed Source OS
5.6 Key Takeaways

Operating System Services

- The OS provides a **user interface** that allows users to interact with the system.
- This can be in the form of a **command-line interface (CLI)** or a **graphical user interface (GUI)**.
- The OS is responsible for **program execution**, loading programs into memory and managing their execution.
- The OS also manages **I/O operations**, providing mechanisms for reading from and writing to peripheral devices.
- The OS provides APIs for **file system manipulation**, allowing users and applications to create, delete, read, and write files.
- The OS also facilitates **communication** between processes through mechanisms such as **inter-process communication (IPC)**.

Operating System Services

- Additionally, the OS is responsible for **error detection**, monitoring system operations to identify and handle errors that may occur.
- The OS manages **resource allocation**, distributing system resources among competing processes.
- It also performs **accounting**, tracking resource usage for billing[3] or performance analysis.
- Finally, the OS ensures **protection** and **security** of system resources, enforcing access control policies and safeguarding against threats.

[3] Billing involves monitoring and recording resource consumption (e.g., in cloud computing environments) to charge users or departments accordingly.

A View of Operating System Services

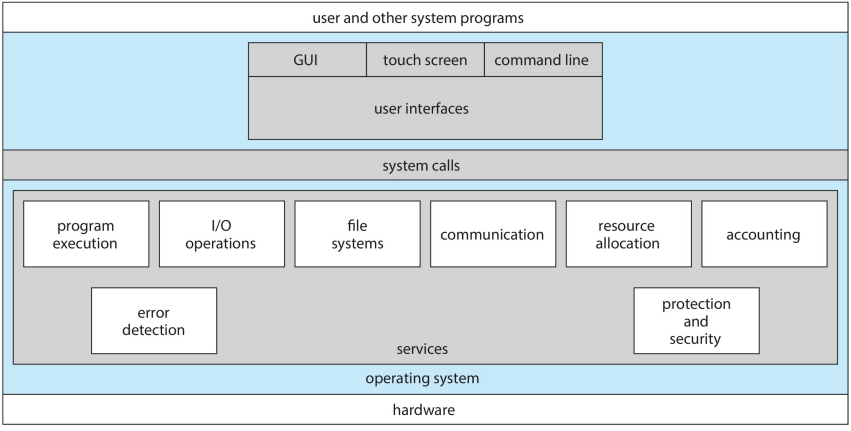


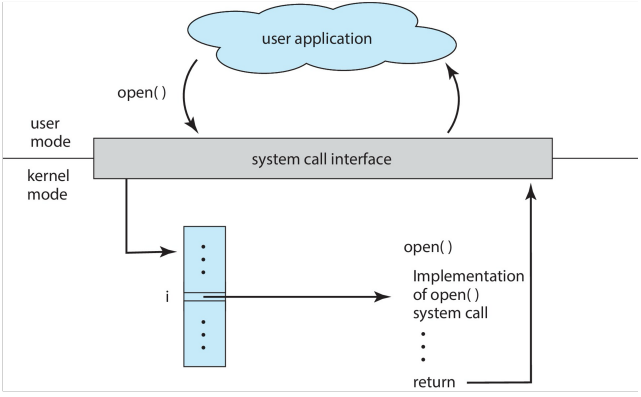
Figure 9: A View of Operating System Services

System Calls

- *System calls* provide the **interface** between a running program and the OS.
- Accessed via high-level *Application Programming Interface (API)*.
- Allow **user-level processes** to **request services** from OS kernel.
- System call categories:
  - **Process control**: Create, terminate, manage processes.
  - **File management**: Create, delete, read, write files.
  - **Device management**: Request/release I/O devices.
  - **Information maintenance**: Get/set system information.
  - **Communication**: Manage inter-process communication.
- Invoked through **software interrupt** or special instruction switching CPU to *kernel mode*.

System Call Interface

The **system call interface** is the **boundary** between a **user program** and the **operating system**.



System Call Example

A simplified example of how a system call works

1. A user-level program invokes a system call (e.g., to open a file).
2. The program uses a special instruction to switch the CPU to kernel mode.
3. The OS kernel receives the system call request and performs the requested operation (e.g., opening the file).
4. The OS kernel returns the result of the operation (e.g., a file descriptor) to the user-level program.
5. The CPU switches back to user mode, and the program continues execution with the result of the system call.
6. This process ensures that user-level programs cannot directly access or modify critical system resources, maintaining system stability and security.

Practical Examples of System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Figure 11: Examples of Windows and Unix System Calls

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:

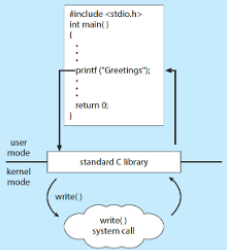


Figure 12: Standard C Library Example

System Programs

- **System programs provide a convenient environment for program development and execution.**
- They can be divided into several categories:
  - **File management programs**: Create, delete, copy, and manipulate files and directories.
  - **Status information programs**: Display system status, resource usage, and process information.
  - **File modification programs**: Edit and modify text files.
  - **Programming language support programs**: Compilers, assemblers, and debuggers.
  - **Program loading and execution programs**: Load and execute user programs.
  - **Communications programs**: Manage network connections and data transfer.
- **System programs often utilize system calls to interact with the OS kernel and perform their functions.**

Linkers and Loaders

- **Linkers and loaders are essential components of the program development and execution process.**
- A **linker** combines multiple object files generated by a compiler into a single executable file.
- The linker resolves references between object files, ensuring that all symbols are defined and correctly linked.
- A **loader** is responsible for loading the executable file into memory and preparing it for execution.
- The loader allocates memory for the program, sets up the program's stack and heap, and initializes the program's execution environment.
- **Both linkers and loaders are crucial for transforming source code into a runnable program.**

Linkers and Loaders (Contd.)

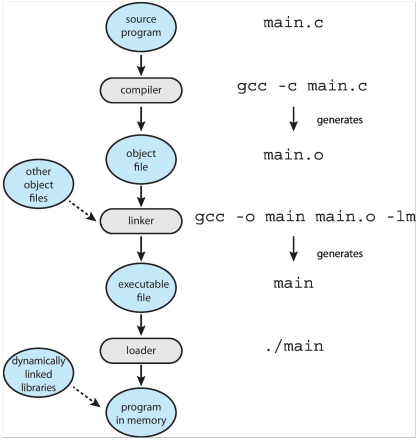


Figure 13: Linker and Loader Process

Why Applications are OS Dependent?

- Applications are often **OS dependent** due to differences in system calls, file systems, and hardware interfaces.
- Different operating systems provide different APIs and system calls, which can lead to compatibility issues.
- File systems may vary between operating systems, affecting how applications read and write data.
- Hardware interfaces and device drivers may also differ, requiring applications to be tailored for specific OS environments.
- **To achieve cross-platform compatibility, developers may use abstraction layers or frameworks that provide a consistent interface across different operating systems.**

Free vs. not Free, Open Source vs. Closed Source OS

- **Free OS**: An operating system that can be used, modified, and distributed without cost.
- **Not Free OS**: An operating system that requires payment for use, modification, or distribution.
- **Open Source OS**: An operating system whose source code is available for anyone to view, modify, and distribute.
- **Closed Source OS**: An operating system whose source code is proprietary and not available for public access or modification.
- **Examples**:
  - **Free and Open Source:** Linux, FreeBSD
  - **Not Free and Closed Source:** Windows, macOS
  - **Free and Closed Source:** Some versions of Android where the source code is not fully open[4].
  - **Not Free and Open Source:** Rare, but some specialized OS may fall into this category.

[4] It is noteworthy that while Android itself is based on the Linux kernel (which is open source), many of devices proprietary add OS layers and applications that are closed source.

Key Takeaways

Key Takeaways

- The OS provides a variety of services to users and applications, including a **user interface**, **program execution**, **I/O operations**, **file system manipulation**, **communication**, **error detection**, **resource allocation**, **accounting**, **protection**, and **security**.
- **System calls** provide the interface between user programs and the OS kernel, allowing programs to request services from the OS.
- **System programs** provide a convenient environment for program development and execution, utilizing system calls to interact with the OS.
- **Linkers** and **loaders** are essential components of the program development and execution process, transforming source code into runnable programs.
- Applications are often **OS dependent** due to differences in system calls, file systems, and hardware interfaces.
- Operating systems can be classified as **free vs. not free** and **open source vs. closed source**, with various implications for usage, modification, and distribution.

End of Chapter 1