Chapter # 1 Objective

- Overview of major components of a contemporary computer system.
- Understanding data structures used in operating systems.
- Exploring computing environments and open-source operating systems.

Section 1.1

Key Points:

1. **Definition of Operating System:**
   - An operating system is software that manages a computer's hardware, serves as an intermediary between users and hardware, and provides a basis for application programs.
   - Operating systems are found in various computing environments, including cars, home appliances, smartphones, personal computers, enterprise computers, and cloud computing.

2. **Role of Operating System:**
   - The operating system allocates resources (CPU, memory, I/O devices, storage) to programs.
   - It is created piece by piece, with well-delineated components, inputs, outputs, and functions.

3. **User View and System View:**
   - **User View:** Focus on ease of use for applications like word processors and web browsers.
   - **System View:** Operating system as a resource allocator, managing resources like CPU time, memory, storage, and I/O devices.

4. **Defining Operating Systems:**
   - No universally accepted definition; it evolves due to the rapid evolution of computers.
   - Operating systems aim to create a usable computing system, facilitating program execution and user problem-solving.
   - Components include the kernel (always-running program), system programs, and application programs.
   - Historical context: The U.S. Department of Justice sued Microsoft in 1998 for including too much functionality in its operating systems.

5. **Why Study Operating Systems:**
   - Understanding operating systems is crucial for proper, efficient, effective, and secure programming.
   - Almost all code runs on top of an operating system, making knowledge of their fundamentals essential for programmers and users alike.

Section 1.2

Key Points:

1. **Computer-System Organization:**
   - A modern general-purpose computer system consists of one or more CPUs, device controllers, and shared memory connected through a common bus.
   - Device controllers are responsible for specific devices (e.g., disk drives, audio devices).
   - Operating systems typically have a device driver for each device controller, providing a uniform interface to the operating system.

2. **Interrupts:**
   - Interrupts are used to alert the CPU to events that require attention.
   - Device controllers use interrupts to inform the device driver that an operation has finished.
   - Interrupts are crucial for asynchronous event handling and hardware interaction.

3. **Storage Structure:**
   - Computer storage is organized in a hierarchy, including registers, cache, main memory (volatile), and secondary storage (nonvolatile).
   - Secondary storage devices include hard-disk drives (HDDs) and nonvolatile memory (NVM) devices like flash memory.
   - Memory is volatile, while secondary storage retains data when power is lost.

4. **I/O Structure:**
   - Operating system code dedicates a significant portion to managing Input/Output (I/O) due to its importance in system reliability and performance.
   - Direct Memory Access (DMA) is used for efficient bulk data movement between devices and main memory without CPU intervention.
   - High-end systems may use switch architecture for concurrent communication between components.

5. **Storage Definitions and Notation:**
   - The basic unit of computer storage is a bit, and a byte consists of 8 bits.
   - Storage capacity is measured in bytes (e.g., kilobyte, megabyte, gigabyte).

- Volatile storage is referred to as memory, while nonvolatile storage is classified as NVS (Secondary storage - HDD, Optical disks, etc., or NVM - Flash memory, SSD, etc.).

6. **I/O Overview:**
- A general-purpose computer system involves multiple devices exchanging data via a common bus.
- Interrupt-driven I/O is suitable for small data movements, but Direct Memory Access (DMA) is used for bulk data movement, reducing CPU overhead.
- Some systems use switch architecture for concurrent communication between components.

Section 1.3
Key Points:

1. **Single-Processor Systems:**
- Historically, computers used a single processor with one CPU and a single processing core.
- Special-purpose processors (e.g., disk, keyboard, graphics controllers) may assist but don't run processes.
- Some special-purpose processors are managed by the operating system, while others operate autonomously.
- PCs often have a microprocessor in the keyboard to convert keystrokes into codes for the CPU.
- The term "single-processor system" is rare today due to the prevalence of multiprocessor systems.

2. **Multiprocessor Systems:**
- Modern computers commonly use multiprocessor systems with two or more processors, each having a single-core CPU.
- Symmetric multiprocessing (SMP) is common, where each CPU performs all tasks, sharing physical memory over the system bus.
- Multiprocessor systems can be more efficient, but overhead and contention for resources may limit the speedup ratio.

3. **Multicore Systems:**
- The definition of multiprocessor now includes multicore systems with multiple computing cores on a single chip.
- Multicore systems can be more power-efficient and faster due to on-chip communication.

4. **NUMA Systems:**
- Non-uniform memory access (NUMA) systems involve providing each CPU with its own local memory connected by a shared system interconnect.
- NUMA systems can scale effectively with added processors but may face increased latency for remote memory access.

5. **Clustered Systems:**
- Clustered systems gather multiple CPUs or nodes, often with each node being a multicore system.
- Clusters can be loosely or tightly coupled, providing high availability and increased reliability.
- Clusters can also be used for high-performance computing by running applications concurrently on all computers in the cluster.
- Different types of clusters include parallel clusters and those operating over a wide-area network (WAN).
- Cluster technology is evolving rapidly, supporting thousands of systems and taking advantage of storage-area networks (SANs).

Overall, the text provides an overview of the evolution and characteristics of computer-system architectures, emphasizing the shift from single-processor to multiprocessor and clustered systems.

Section 1.4
Key Points:

Section 1.4: Operating-System Operations

1. Bootstrap Program and Operating System Initialization
- A bootstrap program initializes the system, loading the operating-system kernel into memory.
- The kernel, once loaded, provides services to the system and users.
- System programs become system daemons, running alongside the kernel.

1.4.1: Multiprogramming and Multitasking
- Multiprogramming allows multiple processes to be in memory simultaneously, preventing CPU idle time.
- Processes switch during waiting periods, optimizing CPU utilization.
- Multitasking is an extension of multiprogramming, providing fast response times.
- Memory management, CPU scheduling, and limiting process interference are essential for multitasking.

1.4.2: Dual-Mode and Multimode Operation
- Dual-mode operation (kernel mode and user mode) protects the operating system from errors and malicious programs.
- Mode bit distinguishes between user and kernel mode.
- Privileged instructions execute only in kernel mode.
- Extended modes exist (e.g., protection rings in Intel processors, multiple modes in ARMv8 systems).

1.4.2: System Calls
- System calls allow user programs to request operating-system tasks.
- Transition from user mode to kernel mode occurs during system calls.
- System calls are handled by the operating system's service routine.

1.4.3: Timer
- Timers prevent user programs from monopolizing the CPU.
- Timers can be fixed or variable, interrupting the system after a specified period.
- Linux uses timers with a specified frequency (HZ) to manage interrupts.

Section 1.9
Key points:

1. **Lists, Stacks, and Queues:**
   - Lists, essential in computer science, can be implemented using linked lists.
   - Singly linked lists, doubly linked lists, and circularly linked lists offer different advantages.
   - Stacks follow the Last In, First Out (LIFO) principle, while queues follow the First In, First Out (FIFO) principle.
   - Stacks are used for function call operations in operating systems, while queues are common for tasks waiting to be processed.

2. **Trees:**
   - Trees are hierarchical data structures with parent–child relationships.
   - Binary trees have at most two children for each parent, and binary search trees impose ordering.
   - Balanced binary search trees, like red-black trees, are used for efficient searching in algorithms, as seen in Linux's CPU-scheduling algorithm.

3. **Hash Functions and Maps:**
   - Hash functions convert data into numeric values, serving as indices for quick data retrieval from tables (arrays).
   - Hash maps associate key-value pairs using a hash function, enhancing data retrieval efficiency.
   - Collisions, where different inputs yield the same output, are managed by linked lists at table locations.

4. **Bitmaps:**
   - Bitmaps are strings of binary digits used to represent the status of items.
   - Efficient for representing the availability of numerous resources, as they are space-efficient compared to using full Boolean values.
   - Widely used in scenarios such as disk drives to indicate the availability of individual units.

5. **Linux Kernel Data Structures:**
   - Linux kernel uses linked lists, queues (kfifo), and balanced binary search trees (red-black trees) for various functionalities.
   - Header files like <linux/list.h> and <linux/rbtree.h> provide details on these data structures in the Linux kernel source code.

In summary, the text highlights the importance of these data structures in operating systems and provides insights into their applications and implementations.

Section 1.10
Key points:

1. **Traditional Computing:**
   - Evolution of the "typical office environment" with PCs, servers, and network connections.
   - Web technologies and increased WAN bandwidth expanding traditional computing boundaries.
   - Growth in home network capabilities, including firewall usage.

2. **Mobile Computing:**
   - Refers to handheld devices like smartphones and tablets.
   - Evolution of mobile devices beyond email and web browsing to include music, video, books, and augmented-reality applications.
   - Utilization of unique features like GPS, accelerometers, and gyroscopes.
   - Dominance of Apple iOS and Google Android in mobile operating systems.

3. **Client–Server Computing:**
   - Architecture where server systems fulfill client-generated requests.
   - Distinction between compute servers and file servers.
   - Example scenarios of compute servers running databases and file servers delivering web content.

4. **Peer-to-Peer Computing:**
   - Peer-to-peer systems treat all nodes as peers, allowing them to act as clients or servers.
   - Advantages over client–server systems, avoiding server bottlenecks.

- Examples like Napster, Gnutella, and Skype showcasing peer-to-peer functionalities.

5. **Cloud Computing:**
- Delivery of computing, storage, and applications as services across networks.
- Various types: public cloud, private cloud, hybrid cloud, Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS).
- Cloud management tools and virtualization technologies play a crucial role.

6. **Real-Time Embedded Systems:**
- Embedded systems are prevalent, found in various devices with specific tasks.
- Real-time operating systems crucial for systems with rigid time requirements.
- Applications include scientific experiments, medical imaging, industrial control, and more.
- Mention of the growth and potential applications of embedded systems in home automation.

The text concludes by referencing future chapters that delve into specific topics related to operating systems, such as real-time components in Linux.

Section 1.11 (Home reading)
key points:

1. **Free and Open-Source Operating Systems:**
- Free software and open-source software are available in source-code format.
- Free software allows no-cost use, redistribution, and modification, while open-source software may not necessarily offer these freedoms.
- Examples include GNU/Linux, Microsoft Windows (closed source), and Apple's macOS (hybrid approach).

2. **Benefits of Starting with Source Code:**
- Studying operating systems by examining the source code allows programmers to produce executable binary code.
- Benefits of learning from source code include modification, compilation, and running code for practical understanding.
- The text includes projects involving modifying operating-system source code and provides examples of open-source code for deeper study.

3. **Advantages of Open-Source Operating Systems:**
- Open-source systems foster a community of unpaid programmers contributing to code development, debugging, analysis, support, and suggestions.
- Security is argued to be higher in open-source code due to more eyes scrutinizing it.
- Companies like Red Hat demonstrate that open-sourcing code can be beneficial for revenue generation through support contracts and hardware sales.

4. **History:**
- In the early days, software generally came with source code, but companies later sought to limit software use and distribution.
- Richard Stallman initiated the free-software movement in 1984, developing GNU, a free UNIX-compatible operating system.
- The GNU General Public License (GPL) implements copyleft, allowing users essential freedoms with the condition of preserving those freedoms during redistribution.

5. **GNU/Linux:**
- Linux, a UNIX-like kernel released by Linus Torvalds in 1991, combined with GNU tools to create the GNU/Linux operating system.
- GNU/Linux has various distributions (distros) like Red Hat, SUSE, Debian, and Ubuntu, each with specific features and purposes.
- The text provides a virtual machine image of GNU/Linux for educational purposes.

6. **BSD UNIX:**
- BSD UNIX, with versions like FreeBSD, NetBSD, and OpenBSD, has a longer history, starting in 1978 as a derivative of AT&T's UNIX.
- BSD UNIX's development history involves a lawsuit by AT&T, but eventually, open-source versions were released.
- Darwin, the core kernel of macOS, is based on BSD UNIX and is open-sourced.

7. **Solaris and Open-Source Learning Tools:**
- Sun Microsystems' Solaris, a commercial UNIX-based OS, was partially open-sourced through the OpenSolaris project.
- Open-source operating systems serve as valuable learning tools, allowing students to explore, modify, and contribute to mature projects.
- The diversity of open-source projects, licenses, and goals contributes to a rich learning environment.

The text emphasizes that the availability of source code has made it easier to study operating systems, promoting a more engaging and accessible learning experience.