

Operating System: Interview Short Notes



Operating systems are the foundation of modern digital devices. These systems determine how you see and interact with your machines. Operating systems determine how we program an application, what software we can use, and how hardware should be designed. Understanding operating systems is essential for developers to ensure you're designing programs that work to the strengths of a particular operating system.

Today, we'll break down the basics of operating system components and functions, starting with the most basic kernel and ending with the more advanced topic of virtualization. By the end of this article, you'll have a solid foundation of operating systems and be ready to jump into intermediate practice.

Here's what we'll cover:

- What is an operating system?
 - Popular operating systems for developers
- What is a kernel?
 - Resource Management
- What is a process?
- What is a service?
 - Common Services
 - Program Execution
 - Input/Output (I/O) operations
 - File System manipulation
 - Communication
 - Error Detection
 - Resource Allocation
 - Protection
- Types of operating system configurations
 - Network Operating System
 - Batch Operating System
 - Time-sharing Operating System
 - Distributed Operating System
 - Real-time Operating System
- Advanced OS concepts
 - Process Scheduling
 - Shortest Remaining Time (SRT)
 - Round Robin Scheduling
 - Multiple-Level Queues Scheduling
 - First In, First Out (FIFO)

- Shortest Job Next (SJN)
- Priority-based Scheduling
- Threads and Multithreading
 - Processes vs. Threads
- Virtualization

What is an operating system?

An operating system (OS) is the core software that contains a collection of services essential for the machine to run programs. Think of this as the **manager of a computer's memory and processes**. An OS is how we can communicate with a computer without understanding the same language. Without an OS, a user would need to know dozens of command line statements to manually control each part of a computer.

The most important function of an OS is to standardize and mediate programs and the machine's internal hardware. The OS is how a computer schedules sequential tasks, completes tasks concurrently, and delegates resources. In other words, the OS is to provide both the platform and resources needed for programs to run.

Before operating systems were commonplace, developers had to design their programs to work with individual pieces of hardware. However, it was impossible to know what hardware your customers would have, so widely distributed software was nearly impossible. Operating systems provide a standard medium that developers can program. If the system can run the OS, it can run your program.

Popular operating systems for developers

There are many operating systems out there, many that you probably haven't even heard of. They generally all provide the same base but have different

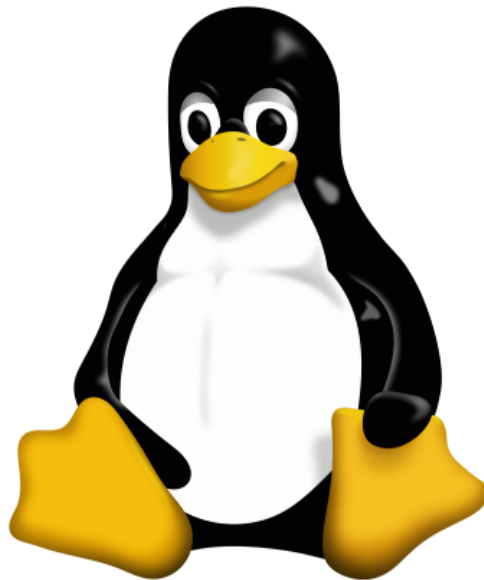
functionalities such as support for hardware, security, intended user-base, or regular updates. Today, the three most popular operating systems used by developers are:

Linux:

This operating system is an open-source operating system available for all computers. Linux's open-source properties allow it to be highly customizable and well suited for installing and running other open-source programs. This combination of open-source technologies has allowed Linux systems to develop far faster than Windows or macOS.

For this reason, Linux is the most popular OS for developers across the board, particularly in cybersecurity fields.

However, Linux has a steep learning curve and can be difficult to pick up.



macOS:

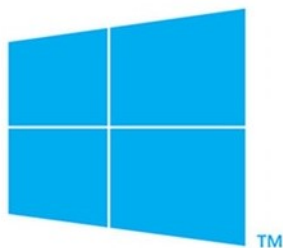
This operating system is developed by Apple and is available on Mac machines. macOS is well-liked by developers for its Unix-based functions like built-in bash-shell utilities and a simple Unix file system. However, many find that macOS doesn't have enough flexibility and instead opt for Linux.



Mac[™]OS

Windows:

This operating system is developed by Microsoft and is available on PC machines. Windows is known for its ease of use and consistent functionality across versions. Since Windows is simple, many developers use other operating systems that have more built-in functions, like macOS, or that are more changeable, like Linux.



Windows[®]

What is a kernel?

The kernel is the central core of an operating system that allows programs to access hardware resources. A kernel is how a system starts up, translates input, and outputs requests to the central processing unit (CPU).

Each OS must have a kernel to function. As a result, the kernel is stored in restricted memory space so it's not accidentally overwritten. Developers working on kernels aim to make it as small as possible without altering functionality so that more memory is open to the user.

Kernels rarely change from version to version. In fact, the modern OS uses the same versions from over 20 years ago with only minor updates or modernizations.

- **Linux** uses the Linux kernel
- **macOS** uses the XNU kernel
- **Windows** uses the Windows NT kernel

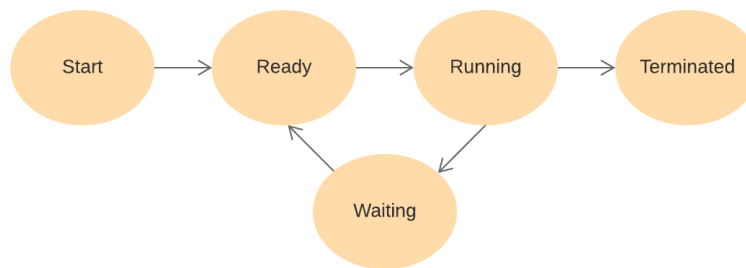
Resource management

Resource management is one of the most important services of a kernel. The kernel delegates computer resources like memory and CPU to each process being executed. By regulating processes, the kernel prevents greedy programs from starving the system and slowing down other concurrent programs. This service ensures that all processes have enough resources to function and that each process makes use of all available resources.

What is a process?

A process is the basic unit of work in an executing program. The developer doesn't write processes. Instead, they write programs that become processes when executed by the machine.

Each process moves through 5 steps:



1. **Start:** The process is first created from a program.
2. **Ready:** The process is waiting to be assigned to a processor by the operating system scheduler.
3. **Running:** The process' instructions are executed by the processor.
4. **Waiting:** The process is set to waiting if it requires additional resources like user input or file access.
5. **Terminated:** After code execution, the process is moved to a terminated state where it will eventually be removed from main memory.

How a process is executed depends on the type of operating system it is on. Below we'll see how each uniquely handles processes.

What is a service?

The kernel is our platform to run programs while services provide tools to the program to do its job. A service is a built-in set of functions that programs can use to implement common behaviors, like saving to files or sharing data. Without services, developers would have to write the code for these behaviors in each program that needs them.

Computer scientists working on OS systems quickly realized that many programs would benefit from these behaviors and included them in the OS package. This makes programming for an OS more convenient, as the developer can avoid writing the most frequently needed behaviors from scratch each time.

For example, a cloud storage application doesn't need to include instructions on how to write data to memory. Instead, the application simply calls the save behavior included in the file system manipulation service.

Some services, like error detection, run at all times regardless of program calls.

Common services

The services included in an operating system vary based from OS to OS, but some have become standard:

- **Program execution:** This service allows the computer to load, execute, and produce the output of a program.
- **Input/Output (I/O) operations:** This service provides the ability to communicate with standard devices like printers and keyboards. The device can use this to make requests of the kernel or visa versa. Software to handle specific devices is called a driver, that instructs the I/O operations service how to interface with the new device.
- **File System manipulation:** This service includes a standardized method of navigating memory so we can create and place files within directories. Programs can then navigate these directories, alter properties, or access/create files. It also provides the interface for users to interact with files through requests like `create/delete` and `move`.
- **Communication:** The communication service enables links between individual processes to share data or results. It also includes the framework for distributed systems of multiple machines working together.
- **Error Detection:** Error detection allows computers to scan all processes for errors by monitoring progress and expected outcomes. This service can also correct minor errors without user interference.
- **Resource Allocation:** This kernel service allows the computer to grant and free up resources dynamically as processes are added or completed. It also allows the computer to create process sequences using CPU schedulers.
- **Protection:** This service provides basic cybersecurity protection through controlling system access, limiting I/O access to unrelated resources, and password authentication.

Keep learning about OS.

You can try Educative's extensive **operating systems** course walks you through the three core pillars of advanced OS use: Virtualization, Concurrency & Persistence. Learn to implement each with hands-on practice.

Operating Systems: Virtualization, Concurrency & Persistence

Types of operating system configurations

Operating systems are used on every digital system and come in different types. Linux, macOS and Windows are all just one type of OS, called a network operating system. While these work great for computers, other systems like email servers and air-traffic control systems have different needs.

Additional types of OS were invented to meet these needs. Each type has unique strengths and processing algorithms to optimize the system for certain tasks.

Companies need developers to make programs tailored to the adopted type of OS. It's essential for you to know how each type of OS works and the unique challenges each poses to a developer.

Below, we'll explore the 6 types of OS, what tasks they're used for, and how you can optimize your programs for each.

Network operating system

Network Operating Systems run on a server and provide the server the ability to manage networking functions like data, users, groups, security, and software programs. These systems are designed to grant file sharing and printer access among multiple computers in a network, typically a local area network (LAN), a private network, or other networks like the internet.

This is the most commonly used type of OS, found on home computers, and on network management devices like routers and switches. Windows, macOS, and Linux are all network operating systems as they must frequently use the Internet.

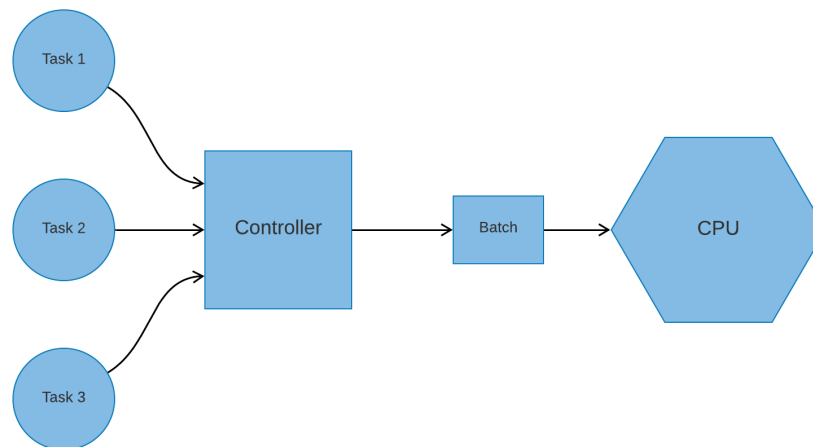
Advantages:

- Server stability and security are handled for the user
- Easy to upgrade hardware and software
- Server access can be achieved in many locations or devices

Disadvantages:

- Expensive to buy and set up a local server
- Reliant on the central server to function
- Requires regular updates and maintenance

Batch operating system



Batch operating systems are designed to speed up processing by “batching” groups of tasks to be run later.

Users do not interact with the computer directly in a batched operating system. Instead, tasks are drafted on separate offline devices and handed off to a computer controller.

The controller runs the batch once they receive it at a specific time, like the end of the workday, or when they get enough tasks to warrant execution. This speeds up processing as similar processes can be completed at the same time.

Batch operating systems are often used in financial fields. Payroll and bank statement processing systems are the two most common systems to implement

batch operating systems. These systems are of an older style, with many companies moving to convert them to time-sharing systems in recent years.

When designing programs for a batch system, minimize processing time as any long process will slow down the work pipeline. However, your program can be very resource-intensive as the device only runs one process at a time.

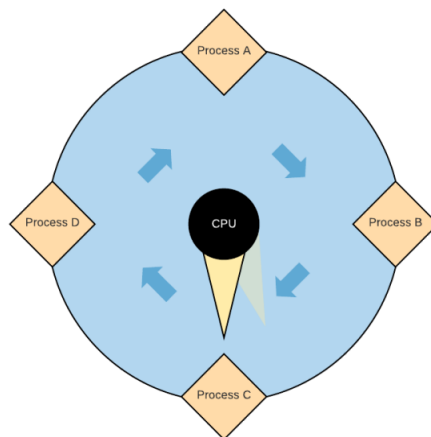
Advantages:

- Always know when the process will be completed
- Shareable by multiple workers
- Can handle consistent large workloads

Disadvantages:

- Efficiency depends on the efficiency of the operator
- Difficult to debug
- Expensive to set up
- Failed jobs cause a backlog

Time-sharing operating system



Time-sharing operating systems allot a period of time, called a quantum, to work on each user's tasks and rapidly switch between each process until all are

complete. While CPUs can only work on one process at a time, the rapid switching simulates concurrent processing.

Time-sharing systems prioritize response time as processes are completed at the same pace regardless of when they were issued. This avoids stacking response time where the first process in a sequence is completed swiftly while the final process is untouched until all others are complete.

Time-sharing systems are not widely used but are used to convert old batch OS devices to be more responsive. Time-sharing is also used by Multics, an operating system that defined modern multitasking systems and Unix, the basis for macOS. This style of system is the backbone of multithreading techniques.

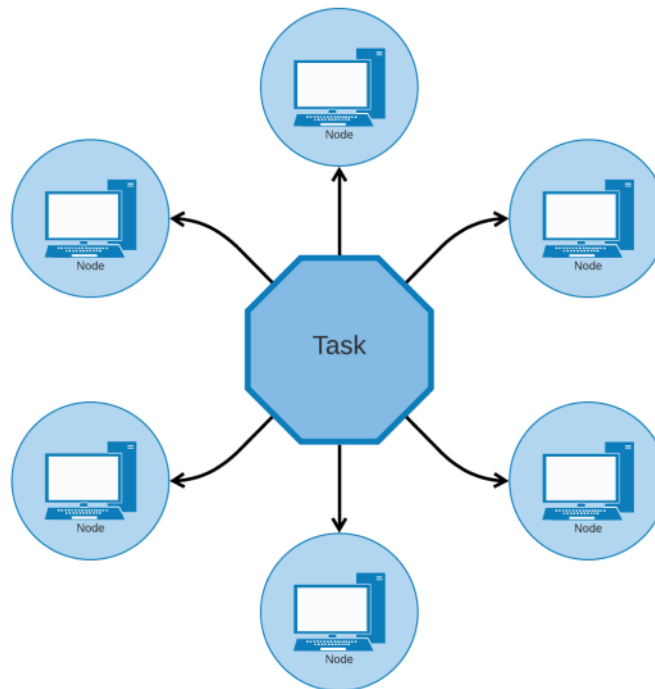
Advantages:

- Ensures a consistently quick response time
- Minimal CPU downtime

Disadvantages:

- Prone to errors
- Minimal security, malware processes will be completed like regular processes
- Concurrent processes cannot easily share data

Distributed operating system



Distributed operating systems use multiple connected computers, called nodes, to handle user tasks. Each process is distributed to available devices for individual completion and all pieces are combined once all nodes are finished. The primary advantage of a distributed system is that it allows individual users to concurrently use the processing power of many devices, resulting in faster processing.

When designing programs for distributed operating systems, split the program into smaller modular tasks to ensure the system can efficiently use its divide and conquer structure.

Common examples of distributed operating systems can be found in large scale, response-focused systems like email servers, cell phone networks, and electronic banking systems.

Advantages:

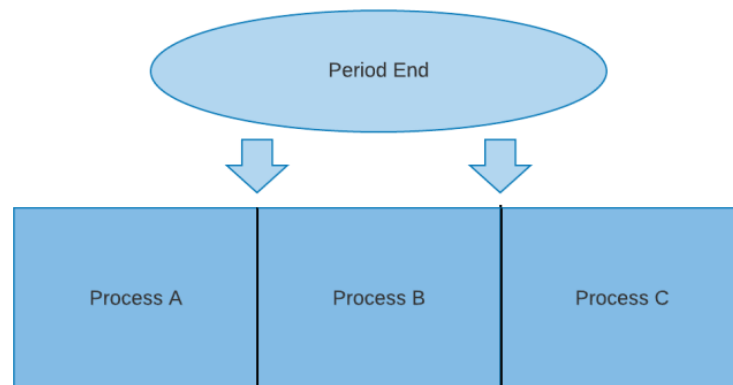
- Durable, the failure of one node will not interrupt other nodes
- Data easily shareable across nodes
- Fast computation due to resources of multiple nodes
- Reduced load on each node

- Scalable, new nodes can be seamlessly added

Disadvantages:

- Only works when connected to the node network
- Expensive to set up

Real-time operating system



This operating system gives each program a given period, with any program that would exceed that processing time being rejected. This maintains a strict schedule of processes to ensure certain high importance tasks can always be performed at a moment's notice.

Real-time systems differ from time-sharing systems in how many processes can be completed at once. Real-time operating systems complete only one process at a time within a certain period. The process is interrupted if the processing time exceeds its dedicated period. Time-sharing systems, on the other hand, switch focus between multiple processes and are allowed to take multiple quanta to complete.

These systems are widely used when responsiveness and strict schedules are essential, such as in equipment for scientific experiments, air traffic control systems, or defense systems.

When writing programs for a real-time system, you must know how long each period is in your system and ensure your program is small enough to complete

within that period.

Advantages:

- Optimal resource use for each process
- Very fast response time, usually just 3 microseconds
- Small program size allows them to be embedded within other systems
- Few errors

Disadvantages:

- Only runs one process at a time
- Achieving maximum efficiency is resource-intensive
- Difficult to program for, each program must have interruptor conditions throughout the program that stop the process if the period ends

Advanced OS concepts

Process scheduling

The CPU scheduler component in the resource allocation service can use one of 6 algorithms to decide on which process completes first. Each of these algorithms is either preemptive or non-preemptive.

Preemptive schedulers use a priority system to influence processing. High-priority processes are scheduled as soon as they reach the `ready` stage, even if another process is currently running. If this happens, the high-priority process overrules the lower priority process.

Below are the 3 common preemptive scheduling algorithms:

- **Shortest remaining time (SRT):** The processor prioritizes processes closest to completion. If a new process is added that has a lower completion time than the current process, the new process is executed first. This is often used with batch OS systems.

- **Round-robin scheduling:** The processor rotates each process, allowing each a quantum of time processing. The process is interrupted once the quantum is over and is then preempted by the next process. This algorithm is the foundation of time-sharing operating systems.
- **Multiple-Level Queues Scheduling:** Processes are broken into separate queues, each with different scheduling algorithms. The processor alternates between each queue and assigns processes based on the algorithm of their home queue. This is often used when one system handles vastly different processes that each call for a unique algorithm.

Non-preemptive schedulers don't allow interruptions. Instead, a process cannot be preempted once it has started. They may still use a priority system but high-priority processes must still wait until the current process is complete before beginning.

- **First in, first out (FIFO):** The processor completes processes in the order they came in. This is the simplest algorithm to implement but also results in the slowest average processing time.
- **Shortest job next (SJN):** The processor always queues the next shortest job. This algorithm produces a very low processing time but cannot be implemented unless both the duration of CPU availability and the processing time of each process are known. SJN is the non-preemptive equivalent of the shortest remaining time and is also used in batch OS systems.
- **Priority-based scheduling:** Each process is assigned a priority based on a preset resource-based factor such as memory usage or processing time. Processes are then completed by the CPU in order of their priority. If multiple processes have the same priority, the CPU executes them using the FIFO algorithm. This is one of the most widely used scheduling algorithms.




Threads and multithreading

A thread is the smallest sequence of programmed instructions and is the main component of any process. Each thread completes a particular command. When all threads are complete, the process has been fully executed.

Multiple threads can exist in one process and each share data and memory space between all threads in the process. Threads can be completed one at a time or

concurrently using either context switching or multiple CPU cores.

Processes vs. Threads

 S.No.	 Processes	 Threads
<u>1.</u>	Processes are independent	Threads are a subset of processes
<u>2.</u>	Processes carry more unique information than threads	Multiple threads within a process share resources
<u>3.</u>	Processes have separate memory addresses	Threads share their address space
<u>4.</u>	Processes context switch relatively slowly	Threads context switch quickly

Virtualization

Virtualization is the technique of making virtual objects that behave as hardware, such as the switching ability of time-sharing operating systems. The time-sharing OS can complete multiple processes simultaneously by switching between them rapidly.

Normally, a computer would need two individual CPUs to complete processes concurrently. As a result, a time-sharing OS achieves virtualization by using rapid process switching to emulate an additional CPU.

Virtualization can simulate entire OS instances with individually assigned sections of the host resources, data, and devices for each container. These containers are isolated from the host machine and all other hosted virtual machines (VMs). This larger scale of virtualization is called operating-system-virtualization or, more commonly, containerization.

Programs run on these containers as though they are a true operating system. They access resources allocated to the container and execute processes using a containerized scheduler.

Virtualization, therefore, grants the ability to **run multiple concurrent programs in separate containers** while all being hosted on the same physical machine.

This gives developers a high level of control over OS container environments. By controlling container environments, developers can test programs on different kinds of operating systems or simulate hardware environments all on the same machine. Also, programs are more efficient when using concurrent programs across multiple VMs.

Advantages:

- Access to multiple operating systems on the same physical machine
- Isolated failure, the host system will continue to function even if one or more of the hosted VMs crash
- Virtual machines are secure, malware is quarantined to individual VMs and cannot damage other OS containers
- Lower maintenance cost than an equivalent number of physical machines

Disadvantages:

- VMs are less efficient than equivalent physical machines as VMs must indirectly request hardware use through the host OS
- Resources are divided among different VMs meaning most machines can only efficiently run a few at a time

Reference:

[Educative.io](https://www.educative.io)

If you don't know Educative is another online learning platform that is gaining a lot of traction for its **text-based, interactive learning courses**. Reading is generally faster than watching and If you prefer reading text than watching videos then this is the platform to checkout.

If you are preparing/appearing for interviews, then I would highly recommend [Grokking the Coding Interview: Patterns for Coding Questions](#).