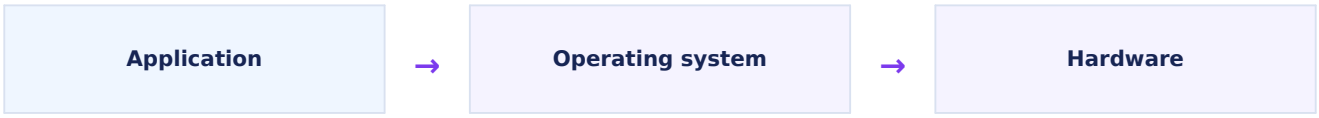
LEARNHUB / TECHNICAL COURSE LIBRARY

Operating Systems

Course introduction

Study the operating system as the manager between applications and hardware, coordinating processes, CPU time, memory, files, and shared resources.



Level: Beginner · Category: Computer Science

Original, beginner-friendly study content designed to accompany the course syllabus and its linked video lesson.

Contents

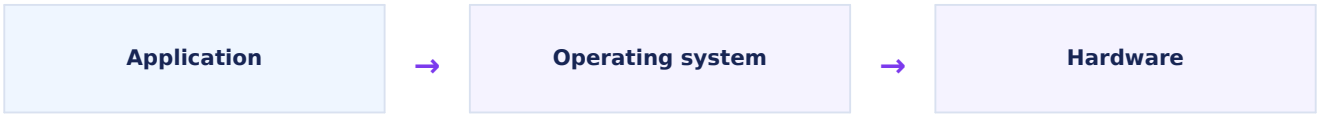
- How to use this resource
- Learning map
- Chapter 01 · Processes and threads
- Chapter 02 · CPU scheduling
- Chapter 03 · Synchronization and deadlocks
- Chapter 04 · Memory management and virtual memory
- Chapter 05 · File systems and durable storage
- Summary and next steps

Course purpose: Study the operating system as the manager between applications and hardware, coordinating processes, CPU time, memory, files, and shared resources.

How to use this resource

Read the chapter titles first, then study the explanations and examples. Pause at each practice prompt before reading its suggested answer. Write down questions that remain and revisit them during revision.

Learning map



Chapter 01 · Processes and threads

The central idea

A process is a running program with its own address space and resources. A thread is an execution path within a process that shares some process state.

How to think about it

Operating systems create, schedule, and terminate processes. Threads can share data efficiently but must coordinate access. Context switching saves enough execution state to resume another task, and excessive switching has overhead.

First example

A browser may use separate processes for isolation and threads inside a process to perform work concurrently.

Try this

Give one benefit and one coordination risk of moving two tasks into threads within one process.

A way to check your answer

> Threads share memory and can communicate quickly. They can race while updating shared state, so synchronization or ownership design is needed.

Chapter 02 · CPU scheduling

The central idea

CPU scheduling selects which ready task runs next. A scheduling policy balances throughput, waiting time, response time, fairness, and priority.

How to think about it

First-come-first-served is simple but can cause long jobs to delay short ones. Round-robin gives each task a time slice. Priority policies can favor urgent work but may starve low-priority tasks without safeguards.

First example

An interactive system uses small time slices so a typing application stays responsive while background work continues.

Try this

Choose a scheduling policy for an interactive terminal and explain one trade-off.

A way to check your answer

> Round-robin can improve responsiveness by sharing the CPU, but a very small time slice adds context-switch overhead. Choose a quantum appropriate to the workload.

Chapter 03 · Synchronization and deadlocks

The central idea

A race condition occurs when behavior depends on unpredictable timing of concurrent operations. A deadlock is a set of tasks waiting indefinitely for resources held by one another.

How to think about it

Protect shared updates with appropriate locks or atomic operations. Deadlock requires mutual exclusion, hold-and-wait, no preemption, and circular wait. Prevention breaks a condition; avoidance and detection use different information and trade-offs.

First example

Two bank-transfer tasks lock accounts in opposite orders; consistent lock ordering can prevent a circular wait.

Try this

Name two necessary deadlock conditions and suggest a simple policy that prevents one of them.

A way to check your answer

> Mutual exclusion and circular wait are two conditions. Enforcing a global resource ordering can prevent circular wait by making lock acquisition follow one direction.

Chapter 04 · Memory management and virtual memory

The central idea

Virtual memory gives each process an address space that the operating system maps to physical memory. Paging divides memory into fixed-size pages and frames.

How to think about it

Address translation protects processes and lets the system use memory flexibly. A page fault occurs when a needed page is not resident and must be loaded. Excessive paging can make a system slow because storage access is much slower than RAM.

First example

A process accesses a virtual page; the memory-management unit consults translation information and the operating system loads the page if needed.

Try this

Explain why virtual memory helps isolation and name one performance cost when RAM pressure is high.

A way to check your answer

> Each process uses its own address-space view and permissions, which helps isolate memory. Heavy page faults can cause storage traffic and a large performance drop.

Chapter 05 · File systems and durable storage

The central idea

A file system organizes named data and metadata on storage devices. It manages directories, permissions, allocation, and recovery behavior.

How to think about it

Applications rely on file operations such as open, read, write, and close. Permissions control access, while caches and write buffers affect when data reaches durable media. Backups remain necessary because file-system correctness cannot prevent every loss.

First example

A course application reads a PDF from a path and depends on correct permissions, available storage, and a valid file format.

Try this

A saved document appears in the file list but contains old data after a sudden shutdown. Suggest two relevant concepts to investigate.

A way to check your answer

Review buffered writes and file-system recovery guarantees, then check whether the application flushed or closed the file correctly. A recent backup may be needed to restore expected content.

Summary and next steps

Operating Systems brings together 5 connected topics. Review each definition, explain one example without looking, and then try the practice prompt again.

Key ideas to carry forward

Processes and threads: A process is a running program with its own address space and resources. A thread is an execution path within a process that shares some process state.

CPU scheduling: CPU scheduling selects which ready task runs next. A scheduling policy balances throughput, waiting time, response time, fairness, and priority.

Synchronization and deadlocks: A race condition occurs when behavior depends on unpredictable timing of concurrent operations. A deadlock is a set of tasks waiting indefinitely for resources held by one another.

Memory management and virtual memory: Virtual memory gives each process an address space that the operating system maps to physical memory. Paging divides memory into fixed-size pages and frames.

File systems and durable storage: A file system organizes named data and metadata on storage devices. It manages directories, permissions, allocation, and recovery behavior.

A short review routine

1. Recall the main idea from each chapter before rereading. 2. Work one example by hand. 3. Explain a common mistake and how you would catch it. 4. Revisit a difficult topic after a short break.

Course outcomes

- Distinguish processes from threads
- Trace common CPU scheduling strategies
- Explain synchronization and deadlock conditions
- Describe virtual memory and file-system responsibilities

This course introduction is an original LearnHub learning aid. Use the referenced ideas for study, then consult the course video and the rest of the library for additional practice.