# CS370: Operating Systems

Computer Science Department, Colorado State University

CS370 is a core undergraduate course taken by junior undergraduate students. We will explore the broad range of issues that underlie modern Operating Systems. In particular, we focus on key concepts and algorithms that are used in both commercial and open-source operating systems. This course will cover the following broad areas:

1. Basic operating system terminology.
2. Processes and thread management.
3. Scheduling algorithms
4. Memory management concepts
5. Interprocess communication
6. Synchronization
7. Deadlocks and resource management
8. File System and storage architecture
9. Symmetric multiprocessing
10. Virtualization

We will be covering virtualization and containers as well. Virtual machines are used heavily in data centers and provide excellent properties such as isolation, consolidation, and support for legacy applications. We will look at Type-1 and Type-2 hypervisors.

**Course Objectives:**

Upon successful completion of this course students will be able to:

1. Explain basic operating system terminology
2. Explain processes and thread management.
3. Distill core concepts in scheduling algorithms and develop tools to assess their performance.
4. Synthesize diverse concepts in memory management.
5. Contrast mechanisms for interprocess communications
6. Distill and build upon core concepts in process and task synchronization
7. Design resource management schemes that mitigate deadlocks
8. Explain file systems and storage architecture
9. Contrast virtualization and containers alongside identifying when one approach outperforms the othe

**Modules in CS370**

| **Introduction** | | |
|---|---|---|
| This module provides an overview of the course, grading criteria, and a brief introduction to high level operating systems concepts. We will explore the differences between kernel mode and user-mode and why they exist. | | |
| | Objectives:<br><br>1. Summarize basic operating systems concepts<br>2. Highlight key developments in the history of operating systems | |
| | | |

**Processes**

Processes are a foundational construct in organizing computations with a program. This module will contrast differences between programs and processes. A key idea covered in this module is the notion of multiprogramming which can used to give the illusion that multiple processes are executing concurrently. We will explore the layout of processes in memory and the various metadata elements regarding a process that are organized within a Process Control Block (PCB). The PCB plays a foundational role in how the OS context-swictches between different processes.

Objectives:

1. Contrast programs and processes
2. Explain the memory layout of processes
3. Describe Process Control Blocks
4. Explain the notion of Interrupts and Context Switches
5. Describe process groups

**Inter-Process Communications**

One of the operating system's main jobs is to keep processes isolated, making sure they run independently without interfering with each other. But sometimes, isolation isn't enough: processes *need* to communicate. In this module, we'll explore how the OS makes that possible. We'll break down the three key mechanisms that allow processes to exchange information while still maintaining order and control.

Objectives:

1. Explain inter-process communications based on Shared Memory
2. Explain inter-process communications based on Pipes
3. Explain inter-process communications based on message passing
   Contrast inter-process communications based on shared memory, pipes, and message passing
4. Design programs that implement inter-process communications

**Threads**

Think of a thread as a lightweight process -- small, fast, and designed to coexist within a larger process. But why would we want a process within a process? The main reason comes down to efficiency: simplified data sharing and rapid context switching. When multiple threads run inside the same process, they can share data easily, without the overhead that comes with communication between separate processes. Achieving this kind of seamless sharing across independent processes would be far more complex and much slower. Ultimately, threads give us a way to keep things both parallel and tightly connected, striking a balance between speed and coordination.

Objectives:

1. Explain differences between processes and threads
2. Compare multithreading models

3. Contrast differences between user and kernel threads
4. Relate dominant threading libraries: POSIX, Win32, and Java
5. Design threaded programs that can synchronize their actions

**Process Synchronization**

When multiple processes run at the same time and need to work together, they have to coordinate their actions carefully. The challenge isn't just making them run concurrently: it's making sure they do so *correctly* and *safely*. A flawed approach can lead to chaos, with processes stepping on each other's toes in unpredictable ways. To get a deeper understanding of these challenges, we'll explore some classic problems in synchronization. These time-tested scenarios reveal the fundamental issues that arise when processes share resources and need to stay in sync.

Objectives:

1. Formulate the critical section problem.
2. Analyze a software solution to the critical section problem (case study: Peterson's solution)
3. Explain Synchronization hardware and Instruction Set Architecture support for concurrency primitives.
4. Evaluate classic problems in synchronization: bounded buffers, readers-writers, dining philosophers.

**Atomic Transcations**

This module will cover issues relating to preserving atomicty of transcactions. We will explore issues that arise when a multiplicty of transcactions need to execute concurrently while preserving safety properties.

Objectives:

1. Explain serializability of transactions
2. Assess locking protocols
3. Explain checkpointing and rollback recovery in transactional systems

**CPU Scheduling algoirithms**

The kernel's job isn't just to run processes -- it has to *juggle* them, making sure they all make progress without stepping on each other's toes. The trick is in the **scheduling**. A good scheduling algorithm has to balance several competing goals: latency, throughput, priority, and fairness. No single approach is perfect, so we'll explore a range of scheduling algorithms, each with its own strengths and trade-offs.

Objectives:

1. Evaluate scheduling criteria including fairness and time quanta.
2. Explain and compare different approaches to scheduling: preemptive and non-preemptive
3. Explain and analyze scheduling algorithms: FCFS, shortest jobs, priority, round-robin, multilevel feedback queues, and the Linux completely fair scheduler.
4. Understand how CPU scheduling algorithms function on multiprocessors.

**Deadlocks**

A large number of processes compete for limited resources on the machine. Incorrect synchronization between these competing processes leads to deadlocks. In this module, we will look at how to characterize deadlocks and the various mechanisms we can use to prevent them by negating structural requirments necessary for deadlocks.

Objectives:

1. Explain deadlock characterization
2. Contrast and explain schemes for deadlock prevention
3. Evaluate approaches to deadlock avoidance
4. Understand recovery from deadlocks

**Memory Management**

Memory is a shared resource that must be effectively managed across different processes that are executing concurrently, Given that Instruction Set Architectures (ISA) operate on on data in registers and memory, how memory is managed and shared across competing processes has implications for performance including completion times and throughput.

Objectives:

1. Understand address binding and address spaces
2. Explain contiguous memory allocations: including their advantages and disadvantages.
3. Analyze the key constructs underpinning paging systems including hardware support, shared pages, and structure of page tables.
4. Explain memory protection in paging environments
5. Explain segmentation based approaches to memory management alongside settings in which they are particularly applicable.

**Virtual Memory**

A pure paging based memory allocation schemes require processes to be entirely memory-resident. This is often infeasible and wasteful. In this module we will explore algorithms that facilitate effective allocation of memory while minimizing wasteful allocations. We consider aspects

of program behavior (such as the working set model) which reduces the total number of pages that need to be allocated to a process.

| | | |
|---|---|---|
| | Objectives:<br><br>1. Explain demand paging and page faults<br>2. Contrast page replacement algorithms and explain Belady's anomaly<br>3. Justify the rationale for stack algorithms<br>4. Explain frame allocations<br>5. Synthesize the concepts of thrashing and working sets | |
| | | |

**Virtualization**

Virtualization creates the illusion of multiple (virtual) machines on the same physical hardware. Virtualization allows a single computer to host multiple virtual machines; each virtual machine potentially running a different OS.As part of this module we will look at Type-1 and Type-2 hypervisors and techniques for effective virtualization.

| | | |
|---|---|---|
| | Objectives:<br><br>1. Explain Virtual Machine Monitors (VMMs)<br>2. Justify the Popek and Goldberg requirements for virtualization<br>3. Explain how Virtualization works in the x86 architecture<br>4. Compare Type-1 and Type-2 Hypervisors | |
| | | |

**Containers**

Containers are a lightweight, performant alternative to virtual machines. Unlike, virtual machines every container does not require its own full-blown OS. In fact, all containers on a single host share a single OS. In this module we will see how a container is ultimately just a group of processes ; as such,, a container can do anything that processes can do albeit with restrictions enforced by the kernel.

| | | |
|---|---|---|
| | Objectives:<br><br>1. Explain containers and contrast how they differ from virtualization<br>2. Synthesize enabling concepts in containerization including cgroups, namespaces, and capabilities.<br>3. Identify key elements that comprise container images | |
| | | |

**File Systems**

Data managed on a hard disk must be amenable to updates, discovery, and retrievals. The underlying storage system only deals with disk blocks. In this module we explore a foundational construct in file systems -- the file control block. We will explore how the design of the file control block informs efficiency in retrievals of content. We will round out our discussion of file systems with a look at the unix file system, file allocation table, and the NT File system.

| | |
|---|---|
| | Objectives:<br><br>1. Summarize file system structure<br>2. Contrast contiguous allocation vs indexed allocations<br>3. Explain the Unix File System<br>4. Explain and contrast Windows File Systems: the File Allocation table and NTFS |

| **Mass Storage** |
|---|
| In this module, we will explore the technologies behind the two popular data storage frameworks: hard disk drives and solid state drives. We will explore the key enablers of these systems. In the case of solid-state drives we will explore issues such as write-amplifications, wear leveling, and read-disturb errors. |

| | |
|---|---|
| | Objectives:<br><br>1. Explain data storage in hard disk drives<br>2. Explain data storage in solid state drives |

| **Disk Scheduling Algorithms** |
|---|
| In this module we will explore the rationale and need for disk scheduling algorithms. We will review several metrics that are used to assess the peformance of disk scheduling algorithms. We will explore and analyze several different disk scheduling algorthms. These algorithms are used to inform disk head movements as data are retrieved. We will look at key disk scheduling algorithms. |

| | |
|---|---|
| | Objectives:<br><br>1. Explain the rationale and need for disk scheduling<br>2. Profile and interpret the performance of disk scheduling algorithms<br>3. Contrast the performance characteriscs of diverse disk scheduling algorithms: FCFS, SSTF, SCAN, C-SCAN, and LOOK |

**Course Textbook:**
*Operating Systems Concepts* book by Silberschatz, Galvin, and Gagne 10th edition. John Wiley & Sons, Inc. ISBN-13: 978-1119800361

**Additional Useful References**

1. Andrew S Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/978-0133591620. [AT]
2. Thomas Anderson and Michael Dahlin. *Operating Systems: Principles and Practice*, 2nd Edition. Recursive Books. ISBN: 0985673524/978-0985673529. [AD]

3. Kay Robbins & Steve Robbins. *Unix Systems Programming*, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2. [RR]
4. *C Programming Language* (2nd Edition). Brian W. Kernighan and Dennis M. Ritchie. Prentice Hall. ISBN: 0131103628/978-0131103627
5. *Concurrent Programming in Java(TM): Design Principles and Pattern* (2nd Edition). Doug Lea. Prentice Hall. ISBN: 0201310090/978-0201310092.

**Grading**

| Course Element | Weight |
|---|---|
| Assignments (6) | 45% [5, 5, 5, 10, 10, 10] |
| Quizzes | 10% |
| Midterm Exam | 20% |
| Comprehensive Final Exam | 25% |

Letter grades will be based on the following standard breakpoints: >= 90 is an A, >= 88 is an A-, >=86 is a B+, >=80 is a B, >=78 is a B-, >=76 is a C+, >=70 is a C, >=60 is a D, and <60 is an F.

**About the Professor**

Shrideep Pallickara is a Professor of Computer Science at Colorado State University where he also directs the Center for eXascale Spatial Data Analytics and Computing (XSD). His research explores the design of large-scale computing systems for big data, distributed analytics, and GeoAI helping to shape the ways in which we can illuminate the dynamics of complex systems. He has published over 175 peer-reviewed articles. His research has been supported by major funding agencies in both the United States and the United Kingdom, including the National Science Foundation's flagship programs in scientific innovation, computing systems, and artificial intelligence; the Department of Homeland Security; the National Institute of Food and Agriculture; the U.K. EPSRC's e-Science program; and the National Endowment for the Humanities. His research and assorted systems software have been deployed in domains such as epidemiology, precision agriculture, urban resilience, earthquake science, environmental and ecological monitoring, and high-energy physics. He is a recipient of the *Board of Governor*'s award, the NSF CAREER award, and a Monfort Professorship.