Operating Systems (OS)

An operating system defines an abstraction of hardware and manages resource sharing among the computer's users. The topics in this area explain the most basic knowledge of operating systems in the sense of interfacing an operating system to networks, teaching the difference between the kernel and user modes, and developing key approaches to operating system design and implementation. This knowledge area is structured to be complementary to the Systems Fundamentals (SF), Networking and Communication (NC), Information Assurance and Security (IAS), and the Parallel and Distributed Computing (PD) knowledge areas. The Systems Fundamentals and Information Assurance and Security knowledge areas are the new ones to include contemporary issues. For example, Systems Fundamentals includes topics such as performance, virtualization and isolation, and resource allocation and scheduling; Parallel and Distributed Systems includes parallelism fundamentals; and and Information Assurance and Security includes forensics and security issues in depth. Many courses in Operating Systems will draw material from across these knowledge areas.

OS. Operating Systems (4 Core-Tier1 hours; 11 Core Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
OS/Overview of Operating Systems	**2**		**N**
OS/Operating System Principles	**2**		**N**
OS/Concurrency		**3**	**N**
OS/Scheduling and Dispatch		**3**	**N**
OS/Memory Management		**3**	**N**
OS/Security and Protection		**2**	**N**
OS/Virtual Machines			**Y**
OS/Device Management			**Y**
OS/File Systems			**Y**
OS/Real Time and Embedded Systems			**Y**
OS/Fault Tolerance			**Y**
OS/System Performance Evaluation			**Y**

OS/Overview of Operating Systems

[2 Core-Tier1 hours]

Topics:

- Role and purpose of the operating system
- Functionality of a typical operating system
- Mechanisms to support client-server models, hand-held devices
- Design issues (efficiency, robustness, flexibility, portability, security, compatibility)
- Influences of security, networking, multimedia, windowing systems

Learning Outcomes:

1. Explain the objectives and functions of modern operating systems. [Familiarity]
2. Analyze the tradeoffs inherent in operating system design. [Usage]
3. Describe the functions of a contemporary operating system with respect to convenience, efficiency, and the ability to evolve. [Familiarity]
4. Discuss networked, client-server, distributed operating systems and how they differ from single user operating systems. [Familiarity]
5. Identify potential threats to operating systems and the security features design to guard against them. [Familiarity]

OS/Operating System Principles

[2 Core-Tier1 hours]

Topics:

- Structuring methods (monolithic, layered, modular, micro-kernel models)
- Abstractions, processes, and resources
- Concepts of application program interfaces (APIs)
- The evolution of hardware/software techniques and application needs
- Device organization
- Interrupts: methods and implementations
- Concept of user/system state and protection, transition to kernel mode

Learning Outcomes:

1. Explain the concept of a logical layer. [Familiarity]
2. Explain the benefits of building abstract layers in hierarchical fashion. [Familiarity]
3. Describe the value of APIs and middleware. [Assessment]
4. Describe how computing resources are used by application software and managed by system software. [Familiarity]
5. Contrast kernel and user mode in an operating system. [Usage]
6. Discuss the advantages and disadvantages of using interrupt processing. [Familiarity]
7. Explain the use of a device list and driver I/O queue. [Familiarity]

OS/Concurrency

[3 Core-Tier2 hours]

Topics:

- States and state diagrams (cross-reference SF/State and State Machines)
- Structures (ready list, process control blocks, and so forth)
- Dispatching and context switching
- The role of interrupts
- Managing atomic access to OS objects
- Implementing synchronization primitives
- Multiprocessor issues (spin-locks, reentrancy) (cross-reference SF/Parallelism)

Learning Outcomes:

1. Describe the need for concurrency within the framework of an operating system. [Familiarity]
2. Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks. [Usage]
3. Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and describe the benefits of each. [Familiarity]
4. Explain the different states that a task may pass through and the data structures needed to support the management of many tasks. [Familiarity]
5. Summarize techniques for achieving synchronization in an operating system (e.g., describe how to implement a semaphore using OS primitives). [Familiarity]
6. Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system. [Familiarity]
7. Create state and transition diagrams for simple problem domains. [Usage]

OS/Scheduling and Dispatch

[3 Core-Tier2 hours]

Topics:

- Preemptive and non-preemptive scheduling (cross-reference SF/Resource Allocation and Scheduling, PD/Parallel Performance)
- Schedulers and policies (cross-reference SF/Resource Allocation and Scheduling, PD/Parallel Performance)
- Processes and threads (cross-reference SF/Computational paradigms)
- Deadlines and real-time issues

Learning Outcomes:

1. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes. [Usage]
2. Describe relationships between scheduling algorithms and application domains. [Familiarity]
3. Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O. [Familiarity]
4. Describe the difference between processes and threads. [Usage]
5. Compare and contrast static and dynamic approaches to real-time scheduling. [Usage]
6. Discuss the need for preemption and deadline scheduling. [Familiarity]
7. Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and problems beyond computing. [Usage]

OS/Memory Management

[3 Core-Tier2 hours]

Topics:

- Review of physical memory and memory management hardware
- Working sets and thrashing
- Caching (cross-reference AR/Memory System Organization and Architecture)

Learning Outcomes:

1. Explain memory hierarchy and cost-performance trade-offs. [Familiarity]
2. Summarize the principles of virtual memory as applied to caching and paging. [Familiarity]
3. Evaluate the trade-offs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed. [Assessment]
4. Defend the different ways of allocating memory to tasks, citing the relative merits of each. [Assessment]
5. Describe the reason for and use of cache memory (performance and proximity, different dimension of how caches complicate isolation and VM abstraction). [Familiarity]
6. Discuss the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem. [Familiarity]

OS/Security and Protection

[2 Core-Tier2 hours]

Topics:

- Overview of system security
- Policy/mechanism separation
- Security methods and devices
- Protection, access control, and authentication
- Backups

Learning Outcomes:

1. Articulate the need for protection and security in an OS (cross-reference IAS/Security Architecture and Systems Administration/Investigating Operating Systems Security for various systems). [Assessment]
2. Summarize the features and limitations of an operating system used to provide protection and security (cross-reference IAS/Security Architecture and Systems Administration). [Familiarity]
3. Explain the mechanisms available in an OS to control access to resources (cross-reference IAS/Security Architecture and Systems Administration/Access Control/Configuring systems to operate securely as an IT system). [Familiarity]
4. Carry out simple system administration tasks according to a security policy, for example creating accounts, setting permissions, applying patches, and arranging for regular backups (cross-reference IAS/Security Architecture and Systems Administration). [Usage]

OS/Virtual Machines

[Elective]

Topics:

- Types of virtualization (including Hardware/Software, OS, Server, Service, Network)
- Paging and virtual memory
- Virtual file systems
- Hypervisors
- Portable virtualization; emulation vs. isolation
- Cost of virtualization

Learning Outcomes:

1. Explain the concept of virtual memory and how it is realized in hardware and software. [Familiarity]
5. Differentiate emulation and isolation. [Familiarity]
6. Evaluate virtualization trade-offs. [Assessment]
2. Discuss hypervisors and the need for them in conjunction with different types of hypervisors. [Usage]

OS/Device Management

[Elective]

Topics:

- Characteristics of serial and parallel devices
- Abstracting device differences
- Buffering strategies
- Direct memory access
- Recovery from failures

Learning Outcomes:

1. Explain the key difference between serial and parallel devices and identify the conditions in which each is appropriate. [Familiarity]
2. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system. [Usage]
3. Explain buffering and describe strategies for implementing it. [Familiarity]
4. Differentiate the mechanisms used in interfacing a range of devices (including hand-held devices, networks, multimedia) to a computer and explain the implications of these for the design of an operating system. [Usage]
5. Describe the advantages and disadvantages of direct memory access and discuss the circumstances in which its use is warranted. [Usage]
6. Identify the requirements for failure recovery. [Familiarity]
7. Implement a simple device driver for a range of possible devices. [Usage]

OS/File Systems

[Elective]

Topics:

- Files: data, metadata, operations, organization, buffering, sequential, nonsequential
- Directories: contents and structure
- File systems: partitioning, mount/unmount, virtual file systems
- Standard implementation techniques
- Memory-mapped files
- Special-purpose file systems
- Naming, searching, access, backups
- Journaling and log-structured file systems

Learning Outcomes:

1. Describe the choices to be made in designing file systems. [Familiarity]
2. Compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each. [Usage]
3. Summarize how hardware developments have led to changes in the priorities for the design and the management of file systems. [Familiarity]
4. Summarize the use of journaling and how log-structured file systems enhance fault tolerance. [Familiarity]

OS/Real Time and Embedded Systems

[Elective]

Topics:

- Process and task scheduling
- Memory/disk management requirements in a real-time environment
- Failures, risks, and recovery
- Special concerns in real-time systems

Learning Outcomes:

1. Describe what makes a system a real-time system. [Familiarity]
2. Explain the presence of and describe the characteristics of latency in real-time systems. [Familiarity]
3. Summarize special concerns that real-time systems present, including risk, and how these concerns are addressed. [Familiarity]

OS/Fault Tolerance

[Elective]

Topics:

- Fundamental concepts: reliable and available systems (cross-reference SF/Reliability through Redundancy)
- Spatial and temporal redundancy (cross-reference SF/Reliability through Redundancy)
- Methods used to implement fault tolerance
- Examples of OS mechanisms for detection, recovery, restart to implement fault tolerance, use of these techniques for the OS's own services

Learning Outcomes:

1. Explain the relevance of the terms fault tolerance, reliability, and availability. [Familiarity]
2. Outline the range of methods for implementing fault tolerance in an operating system. [Familiarity]
3. Explain how an operating system can continue functioning after a fault occurs. [Familiarity]

OS/System Performance Evaluation

[Elective]

Topics:

- Why system performance needs to be evaluated (cross-reference SF/Performance/Figures of performance merit)
- What is to be evaluated (cross-reference SF/Performance/Figures of performance merit)
- Systems performance policies, e.g., caching, paging, scheduling, memory management, and security
- Evaluation models: deterministic, analytic, simulation, or implementation-specific
- How to collect evaluation data (profiling and tracing mechanisms)

Learning Outcomes:

1. Describe the performance measurements used to determine how a system performs. [Familiarity]
2. Explain the main evaluation models used to evaluate a system. [Familiarity]

Systems Fundamentals (SF)

The underlying hardware and software infrastructure upon which applications are constructed is collectively described by the term "computer systems." Computer systems broadly span the sub-disciplines of operating systems, parallel and distributed systems, communications networks, and computer architecture. Traditionally, these areas are taught in a non-integrated way through independent courses. However these sub-disciplines increasingly share important common fundamental concepts within their respective cores. These concepts include computational paradigms, parallelism, cross-layer communications, state and state transition, resource allocation and scheduling, and so on. The Systems Fundamentals Knowledge Area is designed to present an integrative view of these fundamental concepts in a unified albeit simplified fashion, providing a common foundation for the different specialized mechanisms and policies appropriate to the particular domain area.

SF. Systems Fundamentals. [18 Core-Tier1 hours, 9 Core-Tier2 hours]

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
SF/Computational Paradigms	**3**		**N**
SF/Cross-Layer Communications	**3**		**N**
SF/State and State Machines	**6**		**N**
SF/Parallelism	**3**		**N**
SF/Evaluation	**3**		**N**
SF/Resource Allocation and Scheduling		**2**	**N**
SF/Proximity		**3**	**N**
SF/Virtualization and Isolation		**2**	**N**
SF/Reliability through Redundancy		**2**	**N**
SF/Quantitative Evaluation			**Y**

SF/Computational Paradigms

[3 Core-Tier1 hours]

The view presented here is the multiple representations of a system across layers, from hardware building blocks to application components, and the parallelism available in each representation. Cross-reference PD/Parallelism Fundamentals.

***Topics**:*

- Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; Datapath + Control + Memory)
- Hardware as a computational paradigm: Fundamental logic building blocks; Logic expressions, minimization, sum of product forms
- Application-level sequential processing: single thread
- Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers
- Basic concept of pipelining, overlapped processing stages
- Basic concept of scaling: going faster vs. handling larger problems

Learning Outcomes:

1. List commonly encountered patterns of how computations are organized. [Familiarity]
2. Describe the basic building blocks of computers and their role in the historical development of computer architecture. [Familiarity]
3. Articulate the differences between single thread vs. multiple thread, single server vs. multiple server models, motivated by real world examples (e.g., cooking recipes, lines for multiple teller machines and couples shopping for food). [Familiarity]
4. Articulate the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem. This can be motivated by the simple, real-world examples. [Familiarity]
5. Design a simple logic circuit using the fundamental building blocks of logic design. [Usage]
6. Use tools for capture, synthesis, and simulation to evaluate a logic design. [Usage]
7. Write a simple sequential problem and a simple parallel version of the same program. [Usage]
8. Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved. [Assessment]

SF/Cross-Layer Communications

Cross-reference NC/Introduction, OS/Operating Systems Principles

[3 Core-Tier1 hours]

***Topics**:*

- Programming abstractions, interfaces, use of libraries
- Distinction between Application and OS services, Remote Procedure Call
- Application-Virtual Machine Interaction
- Reliability

***Learning Outcomes**:*

1. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers. [Familiarity]

2. Describe how hardware, VM, OS, and applications are additional layers of interpretation/processing. [Familiarity]
3. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers. [Familiarity]
4. Construct a simple program using methods of layering, error detection and recovery, and reflection of error status across layers. [Usage]
5. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging. [Usage]

SF/State and State Machines

[6 Core-Tier1 hours]

Cross-reference AL/Basic Computability and Complexity, OS/State and State Diagrams, NC/Protocols

Topics:

- Digital vs. Analog/Discrete vs. Continuous Systems
- Simple logic gates, logical expressions, Boolean logic simplification
- Clocks, State, Sequencing
- Combinational Logic, Sequential Logic, Registers, Memories
- Computers and Network Protocols as examples of state machines

Learning Outcomes:

1. Describe computations as a system characyterized by a known set of configurations with transitions from one unique configuration (state) to another (state). [Familiarity]
2. Describe the distinction between systems whose output is only a function of their input (Combinational) and those with memory/history (Sequential). [Familiarity]
3. Describe a computer as a state machine that interprets machine instructions. [Familiarity]
4. Explain how a program or network protocol can also be expressed as a state machine, and that alternative representations for the same computation can exist. [Familiarity]
5. Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers). [Usage]
6. Derive time-series behavior of a state machine from its state machine representation. [Assessment]

SF/Parallelism

[3 Core-Tier1 hours]

Cross-reference PD/Parallelism Fundamentals.

Topics*:*

- Sequential vs. parallel processing
- Parallel programming vs. concurrent programming
- Request parallelism vs. Task parallelism
- Client-Server/Web Services, Thread (Fork-Join), Pipelining
- Multicore architectures and hardware support for synchronization

***Learning Outcomes**:*

1. For a given program, distinguish between its sequential and parallel execution, and the performance implications thereof. [Familiarity]
2. Demonstrate on an execution time line that parallelism events and operations can take place simultaneously (i.e., at the same time). Explain how work can be performed in less elapsed time if this can be exploited. [Familiarity]
3. Explain other uses of parallelism, such as for reliability/redundancy of execution. [Familiarity]
4. Define the differences between the concepts of Instruction Parallelism, Data Parallelism, Thread Parallelism/Multitasking, Task/Request Parallelism. [Familiarity]
5. Write more than one parallel program (e.g., one simple parallel program in more than one parallel programming paradigm; a simple parallel program that manages shared resources through synchronization primitives; a simple parallel program that performs simultaneous operation on partitioned data through task parallel (e.g., parallel search terms; a simple parallel program that performs step-by-step pipeline processing through message passing). [Usage]
6. Use performance tools to measure speed-up achieved by parallel programs in terms of both problem size and number of resources. [Assessment]

SF/Evaluation

[3 Core-Tier1 hours]

Cross-reference PD/Parallel Performance.

***Topics**:*

- Performance figures of merit
- Workloads and representative benchmarks, and methods of collecting and analyzing performance figures of merit
- CPI (Cycles per Instruction) equation as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
- Amdahl's Law: the part of the computation that cannot be sped up limits the effect of the parts that can

***Learning Outcomes**:*

1. Explain how the components of system architecture contribute to improving its performance. [Familiarity]
2. Describe Amdahl's law and discuss its limitations. [Familiarity]
3. Design and conduct a performance-oriented experiment. [Usage]
4. Use software tools to profile and measure program performance. [Assessment]

SF/Resource Allocation and Scheduling

[2 Core-Tier2 hours]

***Topics**:*

- Kinds of resources (e.g., processor share, memory, disk, net bandwidth)
- Kinds of scheduling (e.g., first-come, priority)
- Advantages of fair scheduling, preemptive scheduling

***Learning Outcomes**:*

1. Define how finite computer resources (e.g., processor share, memory, storage and network bandwidth) are managed by their careful allocation to existing entities. [Familiarity]

2. Describe the scheduling algorithms by which resources are allocated to competing entities, and the figures of merit by which these algorithms are evaluated, such as fairness. [Familiarity]
3. Implement simple schedule algorithms. [Usage]
4. Use figures of merit of alternative scheduler implementations. [Assessment]

SF/Proximity

[3 Core-Tier2 hours]

Cross-reference AR/Memory Management, OS/Virtual Memory.

***Topics**:*

- Speed of light and computers (one foot per nanosecond vs. one GHz clocks)
- Latencies in computer systems: memory vs. disk latencies vs. across the network memory
- Caches and the effects of spatial and temporal locality on performance in processors and systems
- Caches and cache coherency in databases, operating systems, distributed systems, and computer architecture
- Introduction into the processor memory hierarchy and the formula for average memory access time

***Learning Outcomes**:*

1. Explain the importance of locality in determining performance. [Familiarity]
2. Describe why things that are close in space take less time to access. [Familiarity]
3. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time. [Assessment]

SF/Virtualization and Isolation

[2 Core-Tier2 hours]

***Topics**:*

- Rationale for protection and predictable performance
- Levels of indirection, illustrated by virtual memory for managing physical memory resources
- Methods for implementing virtual memory and virtual machines

***Learning Outcomes**:*

1. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources. [Familiarity]
2. Describe how the concept of indirection can create the illusion of a dedicated machine and its resources even when physically shared among multiple programs and environments. [Familiarity]
3. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation. [Assessment]

SF/Reliability through Redundancy

[2 Core-Tier2 hours]

***Topics**:*

- Distinction between bugs and faults
- Redundancy through check and retry

- Redundancy through redundant encoding (error correcting codes, CRC, FEC)
- Duplication/mirroring/replicas
- Other approaches to fault tolerance and availability

***Learning Outcomes**:*

1. Explain the distinction between program errors, system errors, and hardware faults (e.g., bad memory) and exceptions (e.g., attempt to divide by zero). [Familiarity]
2. Articulate the distinction between detecting, handling, and recovering from faults, and the methods for their implementation. [Familiarity]
3. Describe the role of error correcting codes in providing error checking and correction techniques in memories, storage, and networks. [Familiarity]
4. Apply simple algorithms for exploiting redundant information for the purposes of data correction. [Usage]
5. Compare different error detection and correction methods for their data overhead, implementation complexity, and relative execution time for encoding, detecting, and correcting errors. [Assessment]

SF/Quantitative Evaluation

[Elective]

***Topics**:*

- Analytical tools to guide quantitative evaluation
- Order of magnitude analysis (Big-Oh notation)
- Analysis of slow and fast paths of a system
- Events on their effect on performance (e.g., instruction stalls, cache misses, page faults)
- Understanding layered systems, workloads, and platforms, their implications for performance, and the challenges they represent for evaluation
- Microbenchmarking pitfalls

***Learning Outcomes**:*

1. Explain the circumstances in which a given figure of system performance metric is useful. [Familiarity]
2. Explain the inadequacies of benchmarks as a measure of system performance. [Familiarity]
3. Use limit studies or simple calculations to produce order-of-magnitude estimates for a given performance metric in a given context. [Usage]
4. Conduct a performance experiment on a layered system to determine the effect of a system parameter on figure of system performance. [Assessment]