Systems Core Themes and Course Descriptions

All systems core courses, including the advanced systems courses, will emphasize the following pervasive themes about computer systems:

Information = Bits + Context

- Everything in a digital computer is represented as a sequence of bits. Both humans and the computer need context to interpret those bits and make use of their information content.

The Semiotics of Systems

- Values can have names or addresses (called signs or, if they involve a particular place in time and space, tokens), addresses can be absolute or relative, and aliases and proxies are sometimes used. In addition, all of these different kinds of information are sometimes grouped into sets/classes (called types).

System Design Involves Tradeoffs

- Complex systems often have multiple objectives and these objectives are often in conflict. Hence, system design often involves tradeoffs (e.g., space-time, security-convenience, setup-processing).
- The metrics we use (both theoretical and empirical) and how we measure both matter.

Systems Programs are a Foundation

- Systems programs almost never have value in isolation. Hence, they must be thought of as a foundation (e.g., to satisfy the needs of application programs and/or users).
- Systems programming often involves reactive programming.

Digital/Discrete Models of an Analog/Continuous World

- In many situations, the world is and should be treated as continuous/analog. Computer systems are discrete/digital representations of the world and, hence, are not just simplifications but also approximations.
- A software system can be in one of a finite number of states at any point in time. Understanding those states and how it transitions between them is critical.
- Systems at different scales (i.e., from individual chips to distributed systems) can have many things in common.

Computer as Other

- Students must learn how to communicate with (and otherwise interact with) the computer. This includes learning how to be precise and formal when providing instructions to the computer (e.g., algorithms, programming languages) and how to be careful and systematic while also being flexible when receiving information from the computer (e.g., compiler warnings and errors, symptomatic behavior).

Appearances can be Deceiving

- It is easy to misunderstand complex systems (e.g., a symptom may not arise near the fault, it is easy to confuse proximal and root causes, systems that appear to be deterministic may be stochastic).

From Abstract Specifications to Complete Implementations

- System programmers must be able to think abstractly but also create concrete implementations (e.g. from protocol to service, from abstract data type to concrete data structure and algorithms, from header/interface to implementation, from application programming interface to a specific language binding).
- System programmers must be able to understand how macromotives can lead to microbehaviors (e.g., how a desire for a reliable stream-oriented protocol leads to handshaking and statefullness).
- System programmers must be able to understand how abstract solutions can be applied to particular situations (e.g., design patterns, architectural styles).

Communication has Multiple Dimensions

- There are many ways for two components of a system to communicate, and they can be characterized along a variety of different dimensions (e.g., the channel, synchronous/asynchronous, directionality). This means that communications that appear similar in some ways may be very different in others, and communications that appear very different may, in fact, be very similar.

Resources must be Shared

- In complex systems the same resource often needs to be used by multiple components. They can be shared in a variety of different ways but different resources may need to be shared in similar ways and similar resources may need to be shared in very different ways. Hence, it is critical to understand methods of sharing (e.g., multiplexers, caches, centralized/decentralized control systems, scheduling).

Reliability can be Elusive

- Most systems can fail in many ways. We tend to model such systems probabilistically which leads us to conclude that no system can be completely reliable. It also leads us towards particular ways of increasing reliability (e.g., redundancy).

Use the Right Tool for the Job

- Systems programmers have many tools at their disposal. It is important to gain experience with all of them and to learn which one to apply in any given situation.

CS 432: Compilers

Course objectives:

- Describe the common phases of a modern compiler, including lexical analysis, parsing, semantic analysis, instruction selection, optimization, and code generation
- Summarize the theoretical and practical limitations of static analysis techniques
- In informal terms, evaluate the information lost during compilation
- Illustrate the relationship between blocks of high-level language and machine code
- Evaluate the trade-offs involved in compiler-based optimization techniques
- Analyze source code and predict the machine language behavior
- Explain how the compiler acts as a bridge between programmer and machine
- Describe how the compiler acts as a subsystem of an overall computing environment
- Summarize the difficulty of writing meaningful compiler error messages
- Discuss why it is important to be able to turn optimizations off when compiling
- Develop a sense of how to use a compiler effectively when developing software

Systems core themes:

- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- From Abstract Specifications to Complete Implementations
- Use the Right Tool for the Job

Modules:

- Lexical analysis (6)
- Parsing (6)
- Semantic analysis (3)
- Activation records (3)
- Intermediate representation (6)
- Basic blocks (3)
- Instruction selection (3)
- Analysis and optimizations (3-6)
- Register allocation (3)

Assessment strategies:

- Build a components of a basic compiler:
  - Use a parser generator to automate lexical analysis and parsing
  - Generate semantic actions that can automate type checking and build intermediate representation of a high-level language
  - Generate executable or interpretable code from intermediate representation
- Apply optimization techniques to dataflow and control flow graphs
- Evaluate source code for compliance with a context free grammar
- Create and apply regular expressions for a given finite automaton

CS 450: Operating Systems

Course objectives:
- Explain the services and design structures of modern OS
- Summarize the relationship between hardware and software support for virtual memory
- Describe the evolution from resident monitors to kernels to hypervisors
- Discuss the relative merits of policies for scheduling and resource allocation
- Compare and contrast techniques for file storage and I/O support
- Implement reactive code that provides services on demand
- Explain the potential and the limits of the role the OS plays in security
- Describe the cooperative nature of OS and hardware design
- Use common industry tools to build a large project collaboratively
- Develop a sense of appreciation for how OS design evolved to solve real problems
- Describe the importance of interfaces and documentation for system integration
- Leverage modular design and interfaces to build complex systems
- Consult technical documentation rather than relying on direct instruction

Systems core themes:
- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Use the Right Tool for the Job

Modules:
- Thread and process management (4.5)
- OS interface and IPC (4.5)
- Synchronization implementation (6)
- Memory management (6)
- Virtualization (6)
- Scheduling (3)
- I/O and file systems (6)
- Security and protection (6)

Assessment strategies:
- Implement key features of a functioning OS
- Simulate and evaluate scheduling and allocation algorithms
- Implement either trap-and-emulate or binary translation virtualization techniques

CS 456: Architecture

Course objectives:
- Summarize the construction of a pipelined processor from low-level building blocks
- Describe hardware techniques for parallel implementation at the instruction, data, and thread levels
- Summarize storage and I/O interfacing techniques
- Apply address decoding and memory hierarchy strategies
- Evaluate the performance impact of cache designs
- Implement custom hardware designs in an FPGA
- Compare and contrast the actual execution of code with software designs
- Justify the use of hardware-based optimization that fails occasionally
- Analyze how a person's logical flow of thinking (sequential) differs from the processor implementation
- Describe how hardware implementations can improve overall system performance
- Develop a sense for the challenges of hardware debugging

Systems core themes:
- Information = Bits + Context
- System Design Involves Tradeoffs
- Digital/Discrete Models of an Analog/Continuous World
- Computer as Other
- From Abstract Specifications to Complete Implementations
- Use the Right Tool for the Job

Modules:
- Assembly language and instruction encoding (6)
- Building a datapath (6)
- Hardware descriptor languages (3)
- Pipelined datapath and hazards (6)
- Memory hierarchy and cache design (6)
- Storage and I/O interfacing (4.5)
- Instruction-level parallelism (4.5)
- Data-level parallel architectures (3)
- Thread-level parallel techniques (3)

Assessment strategies:
- Evaluate quantitative measures of performance
- Write several small assembly language programs
- Implement a basic RISC processor using an FPGA (Verilog/VHDL)

CS 466: Networking Internals

Course objectives:
- Implement network routing and packet delivery algorithms
- Evaluate and critique the overhead in protocol data units at various layers of the TCP/IP stack
- Compare and contrast algorithms that provide end-to-end reliability over an unreliable medium
- Analyze models of major application layer protocols
- Compare and contrast application-layer paradigms for distributed file storage
- Illustrate techniques to mitigate security vulnerabilities of network protocols
- Summarize the changes resulting from the IPv4 to IPv6 transition
- Explain the potential for extending the traditional network stack to support new applications
- Justify protocol design trade-offs from the perspective of multiple stakeholders
- Evaluate formal and informal protocol specifications for correctness

Systems core themes:
- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Systems Programs are a Foundation
- Use the Right Tool for the Job

Modules:
- Network address mapping (7.5)
- Network routing (4.5)
- Transport layer algorithms (3)
- TCP (6)
- Distributed file systems (4.5)
- Host configuration and DNS (4.5)
- File transfer applications (3)
- Network management applications (6)
- IPv6 transition (3)

Assessment strategies:
- Write a program to analyze raw network traffic captured by TCPdump
- Write a program to simulate the encapsulation of upper-layer protocol payloads
- Implement and evaluate the RIP routing protocol on an intranet
- Demonstrate the routing of a query in the Chord P2P network
- Write a simulator for a simplified TCP finite state machine
- Write a simulator for the TCP sliding windows flow and congestion control technique

CS 470: Parallel and Distributed Systems

Course objectives:

- Describe hardware architectures relevant to parallel and distributed systems
- Compare and contrast local, shared, parallel, and distributed file systems
- Compare and apply parallel decomposition strategies including task and data parallelism
- Build software for shared-memory multiprocessing, and for distributed computation using message-passing
- Justify the role of middleware for distributed and parallel computation
- Describe tradeoffs between wide-scale parallel paradigms, including cluster, grid, and peer-to-peer computing
- Explain performance metrics and how they relate to our understanding of a program's behavior
- Describe relevant societal and scientific problems that can be addressed using parallel and distributed computing

Systems core themes:

- System Design Involves Tradeoffs
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Use the Right Tool for the Job
- Communication has Multiple Dimensions
- Resources must be Shared

Modules:

- Parallel/distributed concepts (3)
- Parallel patterns (6)
- Parallel systems (9)
- Distributed systems (9)
- Grid, P2P, and cloud computing systems (6)
- Distributed file systems (6)
- Parallel performance (3)

Assessment strategies:

- Implement three projects illustrating parallel concepts and challenges
  - OpenMP for parallel patterns and data reorganization (*e.g.*, sorting, linear algebra)
  - MPI for large-scale computation and distributed communication (*e.g.*, simulation)
  - Distributed file system simulation using RPC for data replication and consistency issues
- Non-programming design assignments converting sequential algorithms to parallel algorithms
  - Nearest-neighbor
  - Producer/consumer
  - Map/reduce

CS 261: Systems I

Module: C Basics

Instructional objectives:
- Summarize the structure of a typical C program
- Demonstrate basic C programming techniques, including stdio and string processing
- Summarize the fundamental segments of the C memory model
- Justify the use of C99 stdint data types
- Distinguish an address from the data stored at that address
- Compile and execute a C program on the command line
- Demonstrate dereferencing a pointer to a local or global variable
- Explain the importance of pointer type declarations
- Describe the relationship between pointers and arrays
- Perform basic pointer arithmetic
- Explain the results of accessing memory beyond the end of a buffer
- Demonstrate dynamic memory allocation and de-allocation
- Compare and contrast Java object handling with C pointer/structure handling
- Analyze the output of a C program that employs pointers
- Consult language documentation for standard C library functions

Prerequisite material:
- Programming
- Command shell basics
- Unit testing

Themes addressed:
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- Reliability can be Elusive
- Use the Right Tool for the Job

- **Skills and assessments:**
- Write a C program that utilize standard I/O and string processing functionalities
- Write a C program that requires dynamically allocating a data structure and freeing memory
- Write a C program that demonstrates pointer arithmetic and array processing

Module: Compiler and Debugger Use

Instructional objectives:

- Describe the process of executing of an interpreted program (e.g., a shell script)
- Describe the process of compiling and executing a partially compiled/interpreted program (e.g., in Java)
- Explain the stages of the compilation, assembly, linking and loading process
- Explain the difference between compile-time, link-time and run-time messages
- Explain the difference between warning and error messages
- Paraphrase warning and error messages
- Explain why the line numbers in messages do not always correctly identify the location of the fault
- Explain the difference between static analysis and dynamic analysis
- Describe how test cases can be used to find symptoms (that indicate a failure of some kind)
- Explain how programs can be instrumented (e.g., output statements, packet sniffers, debuggers) to help isolate faults
- Explain the role of breakpoints
- Explain the difference between "step into" and "step over"
- Describe the information that can be obtained from a debugged (e.g., watches, stack traces)
- Discuss how instrumentation can change a program's behavior
- Discuss the challenges of debugging multi-threaded/multi-processed programs

Prerequisite material:

- Programming
- Command shell basics
- Unit testing

Themes addressed:

- The Semiotics of Systems
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:

- Complete an assignment in which they write and execute the same program in an interpreted language, a partially compiled/interpreted language, and a compiled language
- Complete an assignment in which they are given code that contains problems of various kinds which they must correct

Module: CPU and Memory Organization

Instructional objectives:

- Identify the main types of storage systems (CPU registers, main memory, online disks, flash memory, backup systems, network-attached storage) and contrast their capabilities
- Describe the organization of the main memory subsystem and its interface with the system bus
- Summarize the principles of location/temporal locality
- Explain the effect of memory latency on running time
- Contrast the operation of different cache organizations (direct mapping, associative mapping, multi-way associative mapping)
- Assess the effect of the alternative cache organizations replacement policies on the effective memory latency

Prerequisite material:

- Information representation and bitwise operations in binary / hexadecimal
- Basic experience in programming using C/C++ or similar languages.

Themes addressed:

- Information = Bits + Context
- System Design Involves Tradeoffs
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Resources must be Shared

Skills and assessments:

- Identify the individual memory access sequence of operations resulting from the execution of a small program that utilizes an array.
- Compute the hit ratio and the effective memory latency given a specific cache organization for a known memory access sequence.
- Predict the content of a given cache organization with a given replacement policy when a specific memory access sequence is executed.

Module: Binary Representation of Information

Instructional objectives:
- Describe positional number systems
- Convert numbers from one positional system to another
- Describe the ASCII encoding scheme for characters
- Describe RGB raster images as a matrix of binary data
- Describe the IEEE 754 floating point standard
- Apply techniques to convert between integer representations
- Apply high-level language bit-wise operations
- Analyze the output of a software base conversion program for correctness
- Justify the trade-offs made in designing the IEEE 754 standard

Prerequisite material:
- Basic structure of C/C++ programming
- Analyze compiler warnings and errors, as well as correcting source code
- Defining test cases and expected outputs
- Standard I/O operations
- Very generic sense of data structures (arrays, contiguous memory, structs/classes)

Themes addressed:
- Information = Bits + Context
- System Design Involves Tradeoffs
- Digital/Discrete Models of an Analog/Continuous World
- Computer as Other
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:
- Perform manual conversion between binary and other representations (e.g., alphabetic characters, hexadecimal, octal, decimal)
- Write software to read a stream of binary data and generate human-readable output
- Write software to generate a simple image format (e.g., TGA)

Module: von Neumann Instruction Cycle

Instructional objectives:

- Apply the five steps of the von Neumann execution cycle to a machine language program to emulate the execution
- Describe the role of the PC, SP, instruction register and other relevant registers in the execution cycle
- Specify how the data and instructions flow between memory, registers, and input/output devices
- Decode machine-language instructions given a simple ISA
- Describe memory organization in terms of application program, stack, and heap
- Explain the notion of a pipeline and how it relates to the von Neumann cycle
- Relate the CPU clock speed with the von Neumann cycle

Prerequisite material:

- Memory addressing
- Mapping between machine language code and assembly language code
- Architecture at a high level: CPU, registers, bus, main memory
- Opcodes/operands and instruction format of machine language instructions

Themes addressed:

- Information = Bits + Context
- System Design Involves Tradeoffs
- Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations

Skills and assessments:

- Complete a programming assignment in a high level language that emulates the behavior of a machine language program.

Module: Basic Circuits

Instructional objectives:
- Describe the basic building blocks and components of a computer (gates, adders, multiplexers, decoders, encoders, flip-flops, registers)
- Design a simple logic circuit using the fundamental building blocks of logic design
- Describe the internal architecture of the CPU (Registers + Datapath + Control)
- Describe how control signals are used to implement specific operations inside the CPU
- Use software simulators to design, and simulate digital circuits

Prerequisite material:
- Knowledge of basic Boolean algebra
- Binary number representation and arithmetic

Themes addressed:
- Information = Bits + Context
- Systems Programs are a Foundation
- Digital/Discrete Models of an Analog/Continuous World
- Computer as Other
- From Abstract Specifications to Complete Implementations
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:
- Use circuit simulation software tools to build a combinational digital circuit implementing a specific decision-making problem based on some input signals/data from the external environment.
- Write the proper sequence of control signals required to accomplish a certain task inside the CPU

Module: Threads vs. Processes

Instructional objectives:
- Summarize the life cycle of a process
- Describe the notion of a virtual address space
- Describe the notion of a thread of execution
- Compare and contrast threads, processes, and tasks
- Summarize the concept of a context switch
- Compare and contrast different OS distinctions between threads and processes
- Explain the advantages and disadvantages of different kernel-to-user thread models
- Compare and contrast APIs and system calls for creating new threads, processes, or tasks
- Describe the differences and similarities between explicit and implicit threading models
- Justify appropriate uses of different threading models
- Explain the limits of multithreading

Prerequisite material:
- von Neumann cycle
- Memory organization
- C basics

Themes addressed:
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- Use the Right Tool for the Job

Skills and assessments:
- Write a C program that uses fork() to create a new process and explores memory locations of common variables
- Write a C program that uses an explicit threading library, such as Pthreads; the program should avoid any shared memory or synchronization problems
- Write a C program that uses an implicit threading library, such as OpenMP; the program should avoid any shared memory or synchronization problems

Module: Interrupts and OS principles

Instructional objectives:
- Describe the concept of an interrupt and how it is handled
- Distinguish user- and kernel-mode
- Summarize the notion of interrupt-driven I/O and DMA
- Justify the need for kernel-mode software
- Compare and contrast system calls with interrupts
- Compare and contrast system calls with function calls
- Summarize common kernel design methodologies and principles
- Explain the bootstrap procedure for a modern interrupt-driven system
- Trace a library function (e.g., printf()) through multiple calls until it invokes a system call
- Summarize the steps performed for processing a system call

Prerequisite material:
- von Neumann cycle
- Memory organization
- C basics
- Basic circuits

Themes addressed:
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Resources must be Shared
- Use the Right Tool for the Job

Skills and assessments:
- Analyze the output of strace for a particular process and explain the system calls that were invoked
- Trace a function call through library code to the point where it invokes a system call
- Write a C program that emulates the system call procedure; one portion of the code will set up the stack for the system call, then another portion will pull information from the stack and process the system call

Module: System Software Design and Evaluation

Instructional objectives:
- Summarize principles of good system design
- Paraphrase the contents of a system specification
- Distinguish the notions of design and specification
- Identify tacit assumptions made in a system specification
- Explain metrics for determining software complexity and quality
- Explain the results of automated tools for static and dynamic analysis
- Evaluate an implementation for correctness and compliance to a specification
- Analyze a specification for alignment with known problems
- Analyze quantitative benchmark results
- Judge competing proposed implementations for relative merit
- Justify modifications to an implementation according to desired behavior

Prerequisite material:
- Software engineering methodologies
- Fundamental concepts of systems and systems programming
- Basic statistics (e.g., mean vs. median)
- Classic problems of synchronization
- Architectural patterns in computing
- Mathematical models of systems

Themes addressed:
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- From Abstract Specifications to Complete Implementations
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:
- Use automated tools to aid in software evaluation
- Complete an assignment in which they model data/control flow according to a specification
- Complete an assignment in which they are given a specification and multiple implementations; rank the implementations according to a variety of metrics, including correctness and quantitative results
- Complete an assignment in which they are given a specification and a poor implementation; the assignment would involve explaining why the implementation behaves poorly and modifying it; the deliverable would involve quantification of the improvement in the form of (micro)benchmark results

CS 361: Systems II

Module: Architectural Analysis, Evaluation and Design

Instructional objectives:

- Describe the difference between and architectural style/idiom and an architecture
- Understand the different uses of the term architecture (e.g., instruction set architecture, microarchitecture, application architecture, network architecture)
- Describe the advantages and disadvantages of layered architectures
- Compare the use of a layered architecture in an application program and in an operating system
- Describe the advantages and disadvantages of client-server architectures
- Describe the advantages and disadvantages of peer-to-peer architectures
- Compare the client-server architectural style to the peer-to-peer architectural style
- Describe a stream
- Describe the pipe and filter architectural style
- Model different architectures using UML diagrams
- Describe the advantages and disadvantages of pipe and filter architectures
- Illustrate the Single Instruction, Single Data (SISD), Multiple Instruction, Single Data (MISD), Single Instruction, Multiple Data (SIMD) and Multiple Instruction, Multiple Data (MIMD) architectural styles using UML sequence/collaboration diagrams
- Describe different component topologies (e.g., linear, tree, star) and when they arise
- Describe situations in which placement/location matters (e.g., chip design) and in which it doesn't (e.g., application programming)
- Evaluate the appropriateness of different architectural styles in different situations
- Choose an appropriate architectural style for a particular situation
- Design an architecture for a simple system

Prerequisite material:

- Hardware basics
- UML sequence/collaboration diagrams
- Command shell basics
- Event-driven programming
- Basic graph theory

Themes addressed:

- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Reliability can be Elusive

Skills and assessments:

- Complete an assignment in which they have to identify the type of architecture style used in several real-world examples and identify the participants
- Write a shell script that uses pipes and redirection
- Complete an assignment in which they have to evaluate the applicability of a given architectural style in a given situation

Module: State Models

Instructional objectives:

- Explain the difference between the static characteristics of a system and the dynamic behavior of a system
- Define the phrase "state of a system"
- Describe different examples of state information
- Explain how state information is maintained in programs
- Explain why state spaces can "explode"
- Discuss the implications of "state space explosion" and the role of abstraction
- Describe examples of state transitions
- Categorize different kinds of state information and transitions
- Describe the elements of a UML statechart diagram
- Explain why it is sometimes necessary (but not always necessary) to have an initial state and a final state
- Interpret a UML statechart diagram
- Compare actions on transitions and actions in states
- Use state generalization and aggregation to simplify a statechart diagram
- Identify the missing states/transitions in an existing statechart diagram
- Evaluate the descriptive power of a statechart diagram
- Implement the dynamic aspects of a system defined using a statechart diagram
- Create a statechart diagram from a textual description of a simple system

Prerequisite material:

- Elementary combinatorics
- Elementary set theory

Themes addressed:

- Information = Bits + Context
- The Semiotics of Systems
- Systems Programs are a Foundation
- Digital/Discrete Models of an Analog/Continuous World
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Reliability can be Elusive

Skills and assessments:

- Complete a writing assignment in which they must describe the important states and transitions of a simple system
- Complete a programming assignment in which they must encapsulate the states and transitions in a textual description of a system
- Complete a writing assignment in which they must discuss alternative ways of encapsulating state and evaluate those alternatives, taking into account the number of states

- Complete an assignment in which they must identify the parts of a UML statechart diagram, interpret the diagram, and evaluate the descriptive power of the diagram (compared to a textual description of the system)
- Complete a programming assignment in which they must implement the dynamic behavior of a system based on a statechart diagram
- Complete a writing assignment in which they must create a UML statechart diagram from a textual description of a system

Module: Mathematical Modeling

Instructional objectives:

- Describe the difference between mathematical models and other kinds of models
- Describe the difference between static models and dynamic models
- Describe the difference between transient states and steady states
- Describe the difference between positive/descriptive and normative/prescriptive models
- Describe the difference between analytic models and numerical models
- Describe the difference between time-based models and event-based models
- Explain the value of modeling (and mathematical modeling in particular)
- Explain an existing deterministic model that makes use of algebra, concepts from discrete mathematics, concepts from calculus, and subscripted variables
- Explain an existing probabilistic model that makes use of a discrete sample space and random variables defined on that sample space
- Explain an existing probabilistic model that makes use of a continuous sample space and random variables defined on that sample space
- Compare a deterministic and probabilistic model of the same system
- Use a model to understand the properties of a system
- Understand the importance of incorporating stochasticity in models of some systems
- Evaluate the appropriateness of deterministic and probabilistic models when trying to understand the properties of a system
- Create a deterministic model of a simple system
- Create a probabilistic model that can be used to determine the expected values of properties of a simple system
- Explain the limitations of calculus when trying to find the optimal solutions of a model

Prerequisite material:

- Calculus
- Discrete Mathematics
- Probability and Statistics
- Elementary systems theory

Themes addressed:

- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Digital/Discrete Models of an Analog/Continuous World
- From Abstract Specifications to Complete Implementations
- Reliability can be Elusive

Skills and assessments:

- Complete an assignment in which they explain a deterministic model and a probabilistic model of a simple system, use the models to determine the properties of the system, and assess the value of the models
- Complete a writing assignment in which they must discover a mathematical model of a computer system and describe the properties of the model

- Complete an assignment in which they must write a program that implements a dynamic, probabilistic model of a system
- Complete an assignment in which they must write two programs, one of which is a time-based model of a dynamic system and the other is an event-based model of a dynamic system
- Complete an assignment in which they must create a probabilistic model of a simple system can calculate the expected value of various properties of that system

Module: Information Exchange

Instructional objectives:

- Describe the differences in the scope of the information exchange: Within same thread, vs. among threads of same process, vs. among processes on same computer, vs. among applications across a network
- Contrast the different levels of synchronization among the entities exchanging the information: i.e. Blocking vs. Non-Blocking send/receive primitives
- Specify the level of multiplicity of the entities exchanging the information: i.e. 1-to-1 (private exchange), vs. 1-to-many (broadcast), vs. many-to-many (the Google Docs model)
- Select the appropriate access privilege (read-only, vs. write-only, read and write) for each entity involved in the information exchange
- Choose the appropriate exchange mechanism when sending data to a function within the same thread: call-by-value vs. call-by reference. Also, understand the differences and effect of thread-safety of how data can be returned from the called function: return a copy of the data, vs. returning a reference to a static global variable (the errno syndrome), vs. returning a dynamically-created object
- Identify and apply the appropriate synchronization mechanism when exchanging information among threads of the same process
- Apply different frameworks (e.g. message queues, shared memory, pipes, etc..) for exchanging information among processes within the same computer system
- Demonstrate the use of the the socket API to exchange information among entities residing on different, but networked-, computer systems
- Predict and handle possible errors that may occur during the exchange of information: e.g. data validation, failure of one or more of the involved entities, etc.

Prerequisite material:

- Programming in C/C++ using pointers, structures, recursion
- Basic assembly language programming, addressing modes
- Data structures: stacks, queues, linked lists, graphs
- Build software using multiple source files, and linking with standard libraries

Themes addressed:

- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:

- Complete programming assignments demonstrating:

- Information exchange within the same thread of execution illustrating sending information using call-by-value vs. call-by-reference. Also illustrate how information can be returned to the caller (e.g. return a copy, return a dynamically-created object, or use of static objects) and the effect on the thread-safety of the called function. In addition, learn how the compiler translates these schemes into machine instructions. Finally, demonstrate the buffer overflow problem and how to avoid it.
- Information exchange among threads belonging to the same process using global variables, and semaphores for concurrency control
- Information exchange among processes within the same computer system using shared memory/message queues/pipes
- Information exchange among processes residing on separate computer systems across a network
- The use of synchronization techniques to coordinate the operation of the two or more parties involved in the exchange of information and to enforce access control policies
- Handling of information exchange errors such as failure of one/more party(ies), corruption/invalidity of the information

Module: Synchronization Primitives and Problems

Instructional objectives:
- Distinguish between locks, semaphores, spinlocks, and monitors
- Explain the need for hardware mechanisms for atomic execution
- Explain the use of barriers in concurrent execution
- Explain the notion of a race condition
- Summarize the notions of mutual exclusion, progress, and bounded waiting
- Distinguish shared memory and message passing communication mechanisms
- Demonstrate the use of synchronization primitive software libraries
- Describe the readers-writers, producer-consumer, and dining philosopher problems
- Demonstrate the use of finite state machines to model concurrent execution
- Compare and contrast shared memory and message passing approaches to communication
- Classify concurrent access to resources according to synchronization problems
- Identify the characteristics of synchronization that lead to deadlock
- Summarize approaches for deadlock avoidance, prevention, and detection
- Implement a multi-threaded program using shared memory synchronization
- Implement a multi-process program using message passing IPC
- Classify an existing software implementation according to synchronization problem
- Evaluate proposed shared access to resources and design an appropriate solution for synchronized access

Prerequisite material:
- Threads vs. processes
- System software design and evaluation
- von Neumann instruction cycle and the semantic gap between HLL and machine language
- Finite state machines

Themes addressed:
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- Appearances can be Deceiving
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:
- Given a set of shared resources (software and otherwise), classify the situation according to the synchronization problem and select an appropriate synchronization primitive to ensure safe access
- Complete a multithreaded programming project that involves selecting the appropriate synchronization problem and deploying both locks and semaphores to ensure safe access

- Complete a multiprocess programming project that involves message passing communication mechanisms; extend the implementation by replacing same-system IPC with socket communication on different machines

Module: Parallel Decomposition and Communication

Instructional objectives:
- Compare and contrast the data and task parallel execution models
- Describe parallel implementation design patterns, such as fork-join, parallel loops, master-worker, SPMD, and map/reduce
- Distinguish parallel programming libraries, including OpenMP, Pthreads, CUDA
- Explain the relationship between parallel software and distributed systems
- Summarize Amdahl's law and its limitations
- Select appropriate synchronization mechanisms, execution models, and parallel design patterns for specific problems
- Implement parallel design patterns using appropriate synchronization and communication techniques
- Analyze sequential programs to identify opportunities for parallel speed-up
- Evaluate theoretical speed-up of a parallel implementation for a given program
- Evaluate empirical performance measurements of parallel vs. sequential implementations
- Justify the design choices of parallel implementation

Prerequisite material:
- Synchronization primitives
- Threads vs. processes
- Interprocess and inter-thread communication techniques, including message passing, sockets, pipes, and shared memory

Themes addressed:
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Computer as Other
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:
- Complete a programming project involving task parallel computation; ideally, the project would follow either the consumer-producer or readers-writers design pattern and demonstrate both actors and reactive processes; communication can accomplished using IPC, sockets, pipes, or some other message passing mechanism
- Complete a programming project involving data parallel computation; communication should involve shared memory and appropriate synchronization primitives
- Given a sequential implementation of a calculation, design and implement a parallel version by selecting the algorithmic structure (task vs. data), implementation strategies (parallel loop, master-worker, fork-join, etc.), and concurrent execution techniques (semaphores, locks, barriers, message passing, shared memory, etc.)

Module: Protocol Analysis, Evaluation, and Design

Instructional objectives:
- Describe the definition of a communications protocol
- Contrast communications protocols to protocols in other disciplines
- Identify the elements of a communications protocol
- Describe (in words) a UML model of a protocol
- Create a UML statechart diagram for a given protocol
- Contrast a protocol and a service
- Describe how a particular protocol impacts communications delays
- Contrast message-based protocols and streaming protocols
- Contrast simplex, half duplex, and full duplex communications
- Contrast pulling and pushing messages
- Contrast connection-oriented and connection-less protocols
- Contrast clients and servers
- Identify the appropriateness of a client-server architecture in different situations
- Contrast different mechanisms for flow control
- Describe the concept of utilization
- Calculate the utilization for different flow control mechanisms
- Describe the relationship between flow control and reliability
- Contrast acknowledgments and negative-acknowledgments
- Describe the definition of handshaking
- Demonstrate the need for handshaking and state information in achieving various objectives
- Evaluate the ability of a protocol to achieve its stated objectives
- Categorize different kinds of messages
- Describe standard metrics for assessing protocols

Prerequisite material:
- Finite state machines and/or UML statechart diagrams
- Binary representations of information
- Abstract data types
- Probability
- Analog/Digital Data and Analog/Digital Signals
- Understanding of the different kinds of delays in communications systems
- Error checking and correction

Themes addressed:
- Information = Bits + Context
- The Semiotics of Systems
- Systems Programs are a Foundation
- Digital/Discrete Models of an Analog/Continuous World
- Computer as Other
- From Abstract Specifications to Concrete Implementations
- Communication has Multiple Dimensions
- Reliability can be Elusive

Skills and assessments:

- Write a textual description of a protocol given a UML statechart diagram
- Write a textual description of a protocol given a UML timing diagram
- Trace the messages exchanged using a given protocol in a given situation
- Calculate the probability of a successful conversation using a particular protocol given information about the reliability of the channel
- Calculate the utilization for different protocols
- Evaluate the applicability of a given protocol in a given situation

Module: The Internet

Instructional objectives:
- Describe the role of standards organizations in creating protocols
- Identify the important standards organizations
- Describe the 5-layer (or 7-layer) model of the Internet
- Identify three or more application-layer protocols
- Contrast electronic mail and instant messaging systems
- Identify two or more transport layer protocols
- Identify two or more network/internetwork layer protocols
- Identify two or more link/physical layer protocols
- Describe the purpose and operation of the Domain Name System (DNS)
- Describe the purpose of Dynamic Host Configuration Protocol (DHCP)
- Describe the purpose and operation of the Transmission Control Protocol (TCP)
- Describe the purpose and operation of the User Datagram Protocol (UDP)
- Contrast UDP and TCP
- Contrast different TCP congestion control systems
- Describe the purpose and operation of the Internet Protocol (IP)
- Contrast different routing algorithms
- Describe the purpose and operation of Ethernet (IEEE 802.3)
- Describe the purpose and operation of WiFi (IEEE 802.11)
- Contrast Ethernet and Wifi
- Describe the need for the Address Resolution Protocol (ARP)
- Contrast class-full and class-less IP addresses
- Contrast the layered Internet with a single-layer inter-network
- Demonstrate the ability to select an appropriate protocol (or protocols) to address a problem
- Identify a protocol from observed message traces
- Describe confidentiality, integrity, authentication and availability
- Describe different types of attacks
- Describe the use of encryption in ensuring confidentiality, integrity, and authentication

Prerequisite material:
- Protocol Analysis, Evaluation and Design

Themes addressed:
- Information = Bits + Context
- The Semiotics of Systems
- Systems Programs are a Foundation
- Appearances can be Deceiving
- From Abstract Specifications to Concrete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Skills and assessments:

- Write a textual comparison of protocols
- Trace the messages exchanged using a given protocol in a given situation
- Construct/deconstruct packets/frames into their constituent parts (given descriptions of the packets/frames)
- List the characteristics/properties of a given communications protocol
- Write a textual description of the need for and operation of ports
- Calculate routes
- Use bitwise AND and OR operations to manipulate flags