Systems Fundamentals (SF)

Preamble

A computer system is a set of hardware and software infrastructures upon which applications are constructed. Computer systems have become a pillar of people's daily life. As such, learning the knowledge about computer systems, grasping the skills to use and design these systems, and understanding the fundamental rationale and principles in computer systems are essential to equip students with the necessary competency toward a career related to computer science.

In the curriculum of computer science, the study of computer systems typically spans across multiple courses, including, but not limited to, operating systems, parallel and distributed systems, communications networks, computer architecture and organization and software engineering. The System Fundamentals knowledge area, as suggested by its name, focuses on the fundamental concepts in computer systems that are shared by these courses within their respective cores. The goal of this knowledge area is 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. These concepts include an overview of computer systems, basic concepts such as state and state transition, resource allocation and scheduling, and so on.

Changes since CS 2013

Compared to CS2013, the SF knowledge area makes the following major changes to the knowledge units:

1. Added two new units: system security and system design;
2. Added a new unit of system performance, which includes the topics from the deprecated unit of proximity and the deprecated unit of virtualization and isolation;
3. Added a new unit of performance evaluation, which includes the topics from the deprecated unit of evaluation and the deprecated unit of quantitative evaluation;
4. Changed the unit of computational paradigms to overview of computer systems, deprecated some topics in the unit, and added topics from the deprecated unit of cross-layer communications;
5. Changed the unit of state and state transition to basic concepts, and added topics such as finite state machines;
6. Changed some topics in the unit of parallelism, such as simple application-level parallel processing;
7. Deprecated the unit of cross-layer communications, and moved parts of its topics to the unit of overview of computer systems;

8. Deprecated the units of evaluation and quantitative evaluation, and moved parts of their topics to the unit of performance evaluation;
9. Deprecated the units of proximity and virtualization and isolation, and moved parts of their topics to the unit of system performance;
10. Deprecated the units of parallelism, and moved parts of its topic to the unit of basic concepts;
11. Renamed the unit of reliability through redundancy to system reliability.

Core Hours

Knowledge Unit	CS Core	KA Core
Overview of Computer Systems	3	
Basic Concepts	4	
Resource Allocation and Scheduling	1	2
System Performance	2	2
Performance Evaluation	2	2
System Reliability	2	1
System Security	2	1
System Design	2	1
Total	**18**	**9**

Knowledge Units

SF-A: Overview of Computer Systems

***CS Core**:*

1. Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; datapath + control + memory)
2. Hardware as a computational paradigm: Fundamental logic building blocks; Logic expressions, minimization, sum of product forms
3. Programming abstractions, interfaces, use of libraries
4. Distinction between application and OS services, remote procedure call
5. Application-OS interaction
6. Basic concept of pipelining, overlapped processing stages
7. Basic concept of scaling: going faster vs. handling larger problems

Learning Outcomes:

1. Describe the basic building blocks of computers and their role in the historical development of computer architecture.

2. Design a simple logic circuit using the fundamental building blocks of logic design to solve a simple problem (e.g., adder).
3. Use tools for capture, synthesis, and simulation to evaluate a logic circuit design.
4. 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.
5. Describe that hardware, OS, VM, application are additional layers of interpretation/processing.
6. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers.
7. Construct a simple program (e.g., a TCP client/server) using methods of layering, error detection and recovery, and reflection of error status across layers.
8. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging.
9. Understand 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 simple, real-world examples.

SF-B: Basic Concepts

CS Core:

1. Digital vs. Analog/Discrete vs. Continuous Systems
2. Simple logic gates, logical expressions, Boolean logic simplification
3. Clocks, State, Sequencing
4. State and state transition (e.g., starting state, final state, life cycle of states)
5. Finite state machines (e.g., NFA, DFA)
6. Combinational Logic, Sequential Logic, Registers, Memories
7. Computers and Network Protocols as examples of State Machines
8. Sequential vs. parallel processing
9. Application-level sequential processing: single thread
10. Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers, pipelining

Learning Outcomes:

1. Describe the differences between digital and analog systems, and between discrete and continuous systems. Can give real-world examples of these systems.
2. Describe computations as a system characterized by a known set of configurations with transitions from one unique configuration (state) to another (state).
3. Describe the distinction between systems whose output is only a function of their input (stateless) and those with memory/history (stateful).
4. Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers).
5. Describe a computer as a state machine that interprets machine instructions.
6. 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.

7. Derive time-series behavior of a state machine from its state machine representation (e.g., TCP connection management state machine).
8. Write a simple sequential problem and a simple parallel version of the same program.
9. Evaluate the performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved.
10. Demonstrate on an execution timeline 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.

SF-C: Resource Allocation and Scheduling

***CS Core**:*

1. Different types of resources (e.g., processor share, memory, disk, net bandwidth)
2. Common scheduling algorithms (e.g., first-come-first-serve scheduling, priority-based scheduling, fair scheduling and preemptive scheduling)

KA Core:

1. Advantages and disadvantages of common scheduling algorithms

***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.
2. Describe how common scheduling algorithms work.
3. Describe the pros and cons of common scheduling algorithms
4. Implement common scheduling algorithms, and evaluate their performances.

SF-D: System Performance

[Cross-reference: AR/Memory Management, OS/Virtual Memory]

***CS Core**:*

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

KA Core:

1. Rationale of virtualization and isolation: protection and predictable performance
2. Levels of indirection, illustrated by virtual memory for managing physical memory resources
3. Methods for implementing virtual memory and virtual machines

***Learning Outcomes**:*

1. Explain the importance of locality in determining system performance.

2. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time.
3. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources.
4. 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.
5. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation.

SF-E: Performance Evaluation

***CS Core**:*

1. Performance figures of merit
2. Workloads and representative benchmarks, and methods of collecting and analyzing performance figures of merit
3. CPI (Cycles per Instruction) equation as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
4. Amdahl's Law: the part of the computation that cannot be sped up limits the effect of the parts that can
5. Order of magnitude analysis (Big O notation)
6. Analysis of slow and fast paths of a system
7. Events on their effect on performance (e.g., instruction stalls, cache misses, page faults)

KA Core:

1. Analytical tools to guide quantitative evaluation
2. Understanding layered systems, workloads, and platforms, their implications for performance, and the challenges they represent for evaluation
3. Microbenchmarking pitfalls

***Learning Outcomes**:*

1. Explain how the components of system architecture contribute to improving its performance.
2. Explain the circumstances in which a given figure of system performance metric is useful.
3. Explain the usage and inadequacies of benchmarks as a measure of system performance.
4. Describe Amdahl's law and discuss its limitations.
5. Use limit studies or simple calculations to produce order-of-magnitude estimates for a given performance metric in a given context.
6. Use software tools to profile and measure program performance.
7. Design and conduct a performance-oriented experiment of a common system (e.g., an OS and Spark).
8. Conduct a performance experiment on a layered system to determine the effect of a system parameter on system performance.

SF-F: System Reliability

***CS Core**:*

1. Distinction between bugs and faults
2. Reliability through redundancy: check and retry
3. Reliability through redundancy: redundant encoding (error correction codes, CRC, FEC)
4. Reliability through redundancy: duplication/mirroring/replicas

KA Core:

Other approaches to reliability

***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).
2. Articulate the distinction between detecting, handling, and recovering from faults, and the methods for their implementation.
3. Describe the role of error correction codes in providing error checking and correction techniques in memories, storage, and networks.
4. Apply simple algorithms for exploiting redundant information for the purposes of data correction.
5. Compare different error detection and correction methods for their data overhead, implementation complexity, and relative execution time for encoding, detecting, and correcting errors.

SF-G: System Security

***CS Core**:*

1. Common system security issues (e.g., virus, denial-of-service attack and eavesdropping)
2. Countermeasures
    - Cryptography
    - Security architecture

KA Core:

1. Countermeasures
    - Intrusion detection systems, firewalls

***Learning Outcomes**:*

1. Describe some common system security issues and give examples.
2. Describe some countermeasures against system security issues.

SF-H: System Design

***CS Core**:*

1. Common criteria of system design (e.g., liveness, safety, robustness, scalability and security)

KA Core:

1. Designs of representative systems (e.g., Apache web server, Spark and Linux)

Learning Outcomes*:*

1. Describe common criteria of system design.
2. Given the functionality requirements of a system and its key design criteria, provide a high-level design of this system.
3. Describe the design of some representative systems.

Professional Dispositions

- **Meticulousness:** students must pay attention to details of different perspectives when learning about and evaluating systems.
- **Adaptiveness**: students must be flexible and adaptive when designing systems. Different systems have different requirements, constraints and working scenarios. As such, they require different designs. Students must be able to make appropriate design decisions correspondingly.

Math Requirements

Required:

- Discrete Math:
  - Sets and relations
  - Basic graph theory
  - Basic logic
- Linear Algebra:
  - Basic matrix operations
- Probability and Statistics
  - Random variable
  - Bayes theorem
  - Expectation and Variation
  - Cumulative distribution function and probability density function

Desirable:

- Basic queueing theory
- Basic stochastic process

Shared Topics and Crosscutting Themes

Shared Topics:

- Networking and communication (NC) –
- Operating system (OS)
- Architecture and organization (AR)
- Parallel and distributed computing (PDC).

Course Packaging Suggestions

Introductory Course to include the following:

- SF-A: Overview of Computer Systems - 2 hours
- SF-B: Basic Concepts - 6 hours
- SF-C: Resource Allocation and Scheduling - 4 hours
- SF-D: System Performance - 6 hours
- SF-E: Performance Evaluation - 6 hours
- SF-F: System Reliability - 4 hours
- SF-G: System Security - 6 hours
- SF-H: System Design - 6 hours

Pre-requisites:

- Sets and relations, basic graph theory and basic logic from Discrete Math
- Basic matrix operations from Linear Algebra
- Random variable, Bayes theorem, expectation and variation, cumulative distribution function and probability density function from Probability and Statistics

Skill statement: A student who completes this course should be able to (1) understand the fundamental concepts in computer systems; (2) understand the key design principles, in terms of performance, reliability and security, when designing computer systems; (3) deploy and evaluate representative complex systems (e.g., MySQL and Spark) based on their documentations, and (4) design and implement simple computer systems (e.g., an interactive program, a simple web server, and a simple data storage system).

Advanced Course to include the following:

- SF-H: System Design - 10 hours
- KUs from OS - 2 hours
- KUs from NC - 2 hours
- KUs from PDC - 2 hours
- KUs from AR - 2 hours
- SF-F: System Reliability - 10 hours
- SF-D: System Performance - 6 hours
- SF-G: System Security - 6 hours

Pre-requisites:

- Basic queueing theory and stochastic process
- Introductory Course of the SF KA

Skill statement: A student who completes this course should be able to (1) have a deeper understanding in the key design principles of computer system design, (2) map such key principles to the designs of classic systems (e.g., Linux, SQL and TCP/IP network stack) as well as that of more recent systems (e.g., Hadoop, Spark and distributed storage systems), and (3) design and implement more complex computer systems (e.g., a file system and a high-performance web server).

Committee

Chair: Qiao Xiang, Xiamen University, Xiamen, China

Members:

- Doug Lea, State University of New York at Oswego, Oswego, NY, USA
- Monica D. Anderson, University of Alabama, Tuscaloosa, AL, USA
- Matthias Hauswirth, University of Lugano, Lugano, Switzerland
- Ennan Zhai, Alibaba Group, Hangzhou, China
- Yutong Liu, Shanghai JiaoTong University, Shanghai, China

Contributors:

- Michael S. Kirkpatrick, James Madison University, Harrisonburg, VA, USA
- Linghe Kong, Shanghai JiaoTong University, Shanghai, China