Systems Programming Course Outline

Year 3 Computer Science - Self-Study Guide

Course Overview

Systems Programming focuses on low-level programming, operating system interfaces, and system-level software development. This course bridges the gap between high-level application programming and computer architecture.

Prerequisites: Data Structures, Computer Architecture, Operating Systems Fundamentals

---

Module 1: Introduction to Systems Programming

Duration: 2 weeks

Learning Objectives

- Understand the role of systems programming in computer science
- Differentiate between systems and application programming
- Learn about the system software stack

Topics Covered

- What is Systems Programming?
- System vs Application Programming
- The Software Stack (Hardware → OS → System Software → Applications)
- Programming Languages for Systems Programming (C, C++, Rust, Assembly)
- Development Environment Setup

Resources

- **Book:** "Computer Systems: A Programmer's Perspective" by Bryant & O'Hallaron (Chapter 1)
- **Online:** Systems Programming Course - UIC
- **Video:** Introduction to Systems Programming - MIT
- **Setup Guide:** Linux Development Environment Setup

Practice Exercises

- Set up a Linux development environment (virtual machine or WSL)
- Write a "Hello World" program in C and compile it manually
- Compare the same program written in C vs Python (performance, memory usage)

Module 2: C Programming for Systems

Duration: 3 weeks

Learning Objectives

- Master C programming language fundamentals
- Understand memory management in C
- Learn about pointers and their applications
- Handle errors and debugging in C

Topics Covered

- C Syntax and Data Types
- Pointers and Memory Addressing
- Arrays and Strings in C
- Dynamic Memory Allocation (malloc, calloc, realloc, free)
- Function Pointers
- Structures and Unions
- Error Handling in C
- Debugging with GDB

Resources

- **Book:** "The C Programming Language" by Kernighan & Ritchie
- **Online Course:** C Programming - CS50x
- **Interactive:** Learn C - Codecademy
- **Reference:** C Reference Manual
- **Debugging:** GDB Tutorial

Practice Exercises

- Implement a dynamic array in C

- Create a simple linked list with insertion and deletion
- Write a program that demonstrates memory leaks and fix them
- Practice with GDB on a buggy program

Module 3: System Calls and OS Interface

Duration: 3 weeks

Learning Objectives

- Understand the system call interface
- Learn process management system calls
- Master file I/O operations
- Handle signals and inter-process communication basics

Topics Covered

- Introduction to System Calls
- Process System Calls (fork, exec, wait, exit)
- File System Calls (open, read, write, close, lseek)
- File Permissions and Access Control
- Directory Operations
- Error Handling with errno
- Introduction to Signals
- Process Environment and Arguments

Resources

- **Book:** "Advanced Programming in the UNIX Environment" by Stevens & Rago (Chapters 1-3, 8)
- **Manual Pages:** Use `man` command on Unix systems
- **Online:** Linux System Call Table
- **Tutorial:** System Programming in C
- **Practice:** Systems Programming Assignments - UIUC

Practice Exercises

- Write a simple shell that can execute commands

- Implement file copy utility using system calls
- Create a program that monitors file changes
- Practice with process creation and management

Module 4: Process Management and Control

Duration: 2 weeks

Learning Objectives

- Deep dive into process lifecycle
- Understand process synchronization primitives
- Learn about process scheduling
- Master signal handling

Topics Covered

- Process Creation and Termination
- Process States and Transitions
- Parent-Child Relationships
- Orphan and Zombie Processes
- Process Groups and Sessions
- Signal Generation and Handling
- Signal Masks and Signal Sets
- Real-time Signals

Resources

- **Book:** "Advanced Programming in the UNIX Environment" (Chapters 8-10)
- **Online:** Process Management - Linux Documentation
- **Video:** Process Management in Operating Systems
- **Reference:** Signal Manual Pages

Practice Exercises

- Implement a process monitor utility
- Create a signal-based communication system

- Write a program that handles multiple signals gracefully
- Build a simple job control system

Module 5: Inter-Process Communication (IPC)

Duration: 3 weeks

Learning Objectives

- Master different IPC mechanisms
- Understand when to use each IPC method
- Learn about synchronization in IPC
- Handle concurrent access to shared resources

Topics Covered

- Pipes (Anonymous and Named/FIFOs)
- Message Queues
- Shared Memory (System V and POSIX)
- Semaphores
- Memory Mapping (mmap)
- Socket Programming Basics
- Synchronization Primitives
- Deadlock Prevention

Resources

- **Book:** "Advanced Programming in the UNIX Environment" (Chapters 15-17)
- **Online:** IPC Tutorial - Beej's Guide
- **Documentation:** POSIX IPC
- **Examples:** IPC Examples Repository
- **Video:** Inter Process Communication

Practice Exercises

- Build a chat application using pipes
- Implement producer-consumer problem using shared memory

- Create a client-server system with message queues
- Write a multi-process application with proper synchronization

---

Module 6: Threading and Concurrency

Duration: 3 weeks

Learning Objectives

- Understand thread concepts and lifecycle
- Master pthread programming
- Learn thread synchronization mechanisms
- Handle race conditions and deadlocks

Topics Covered

- Introduction to Threads vs Processes
- POSIX Threads (pthreads)
- Thread Creation and Termination
- Thread Synchronization (Mutexes, Condition Variables)
- Reader-Writer Locks
- Thread-Safe Programming
- Race Conditions and Data Races
- Deadlock Detection and Prevention
- Thread Pools

Resources

- **Book:** "Programming with POSIX Threads" by David R. Butenhof
- **Online:** Pthread Tutorial
- **Reference:** Pthread Manual Pages
- **Interactive:** Parallel Programming in C with OpenMP
- **Debugging:** Thread Debugging with GDB

Practice Exercises

- Implement a multi-threaded web server

- Solve the dining philosophers problem
- Create a thread-safe data structure (queue/stack)
- Build a parallel computation program (matrix multiplication)

Module 7: File Systems and I/O

Duration: 2 weeks

Learning Objectives

- Understand file system internals
- Master advanced I/O operations
- Learn about I/O multiplexing
- Handle asynchronous I/O

Topics Covered

- File System Architecture
- Inodes and File Metadata
- Directory Structure and Links
- File Locking Mechanisms
- Advanced I/O (select, poll, epoll)
- Asynchronous I/O (AIO)
- Memory-Mapped I/O
- File System Performance

Resources

- **Book:** "Understanding the Linux Kernel" by Bovet & Cesati (Chapters 12, 16)
- **Online:** Linux File System Tutorial
- **Documentation:** Linux I/O Documentation
- **Tutorial:** Epoll Tutorial

Practice Exercises

- Write a file indexing system
- Implement a simple database with file-based storage

- Create an I/O multiplexing server
- Build a file synchronization utility

Module 8: Network Programming

Duration: 3 weeks

Learning Objectives

- Master socket programming
- Understand network protocols (TCP/UDP)
- Learn about network I/O patterns
- Handle network errors and edge cases

Topics Covered

- Socket API Overview
- TCP Socket Programming
- UDP Socket Programming
- Client-Server Architecture
- Network Byte Order
- Socket Options and Configuration
- Non-blocking I/O
- Network Error Handling
- IPv4 vs IPv6 Programming

Resources

- **Book:** "Unix Network Programming" by W. Richard Stevens
- **Online:** Beej's Guide to Network Programming
- **Tutorial:** Socket Programming in C
- **Reference:** Socket Manual Pages
- **Examples:** Network Programming Examples

Practice Exercises

- Build an echo server and client

- Implement HTTP client from scratch
- Create a multi-client chat server
- Write a file transfer protocol

Module 9: Memory Management

Duration: 2 weeks

Learning Objectives

- Understand virtual memory concepts
- Learn about memory allocation strategies
- Master debugging memory issues
- Optimize memory usage

Topics Covered

- Virtual Memory System
- Memory Layout of Programs
- Dynamic Memory Allocation Internals
- Memory Leaks and Detection
- Buffer Overflows and Security
- Memory Mapping and Shared Memory
- Garbage Collection Concepts
- Memory Optimization Techniques

Resources

- **Book:** "Computer Systems: A Programmer's Perspective" (Chapter 9)
- **Tool:** Valgrind Tutorial
- **Online:** Memory Management in C
- **Security:** Memory Safety Guide

Practice Exercises

- Implement a custom memory allocator
- Debug programs with memory leaks using Valgrind

- Write buffer overflow demonstration and protection
- Create a memory profiling tool

Module 10: System Performance and Optimization

Duration: 2 weeks

Learning Objectives

- Learn performance measurement techniques
- Understand system bottlenecks
- Master profiling tools
- Apply optimization strategies

Topics Covered

- Performance Metrics and Measurement
- Profiling Tools (gprof, perf, strace)
- CPU and Memory Bottlenecks
- I/O Performance Optimization
- Cache-Friendly Programming
- System Call Optimization
- Parallel Processing Strategies
- Benchmarking Methodologies

Resources

- **Book:** "Systems Performance" by Brendan Gregg
- **Tools:** Linux Performance Tools
- **Online:** Performance Tuning Guide
- **Tutorial:** Perf Tutorial

Practice Exercises

- Profile and optimize a slow program
- Compare different algorithm implementations
- Benchmark I/O operations

- Create a system monitoring tool

Module 11: System Security

Duration: 2 weeks

Learning Objectives

- Understand system security fundamentals
- Learn about common vulnerabilities
- Master secure programming practices
- Implement security mechanisms

Topics Covered

- Security Fundamentals
- Buffer Overflows and Stack Smashing
- Format String Vulnerabilities
- Race Conditions in Security Context
- Access Control and Permissions
- Cryptographic APIs
- Secure Socket Programming
- Input Validation and Sanitization

Resources

- **Book:** "Secure Programming Cookbook for C and C++" by Viega & Messier
- **Online:** OWASP Secure Coding Practices
- **Course:** Software Security - Coursera
- **Reference:** CWE Database

Practice Exercises

- Exploit and fix buffer overflow vulnerabilities
- Implement secure authentication system
- Write a secure file transfer program
- Audit code for security vulnerabilities

Module 12: Advanced Topics and Project

Duration: 3 weeks

Learning Objectives

- Apply all learned concepts in a comprehensive project
- Explore advanced systems programming topics
- Integrate multiple system components

Topics Covered

- Project Planning and Design
- System Integration
- Advanced Debugging Techniques
- Performance Profiling and Optimization
- Code Review and Quality Assurance
- Documentation and Maintenance
- Modern Systems Programming (Rust, Go)
- Container Technologies and Systems Programming

Final Project Options

1. **Multi-threaded Web Server** with custom protocol
2. **Distributed File System** with replication
3. **Process Scheduler Simulator** with different algorithms
4. **Network Monitoring Tool** with real-time analysis
5. **Custom Shell** with advanced features

Resources

- **Project Ideas:** Systems Programming Projects
- **Modern Languages:** Rust Systems Programming
- **Containers:** Docker and Systems Programming

Assessment Methods

- **Programming Assignments (40%):** Weekly coding exercises
- **Midterm Exam (20%):** Theoretical concepts and code analysis
- **Final Project (30%):** Comprehensive systems programming project
- **Participation (10%):** Code reviews and peer feedback

Development Environment Setup

Required Software

- **Operating System:** Linux (Ubuntu 20.04+ recommended) or macOS
- **Compiler:** GCC 9.0+ or Clang
- **Debugger:** GDB
- **Editor/IDE:** VS Code, Vim, or CLion
- **Version Control:** Git
- **Profiling Tools:** Valgrind, gprof, strace

Installation Commands (Ubuntu)

```bash
sudo apt update
sudo apt install build-essential gdb valgrind strace
sudo apt install git vim code
```

Study Schedule Recommendation

Weekly Structure (16 weeks total)

- **Week 1-2:** Module 1 (Introduction)
- **Week 3-5:** Module 2 (C Programming)
- **Week 6-8:** Module 3 (System Calls)
- **Week 9-10:** Module 4 (Process Management)
- **Week 11-13:** Module 5 (IPC)
- **Week 14-16:** Module 6 (Threading)

Second Half

- **Week 17-18:** Module 7 (File Systems)
- **Week 19-21:** Module 8 (Network Programming)
- **Week 22-23:** Module 9 (Memory Management)
- **Week 24-25:** Module 10 (Performance)
- **Week 26-27:** Module 11 (Security)
- **Week 28-30:** Module 12 (Advanced Topics/Project)

Additional Resources

Books

- "Advanced Programming in the UNIX Environment" by Stevens & Rago
- "Computer Systems: A Programmer's Perspective" by Bryant & O'Hallaron
- "The Linux Programming Interface" by Michael Kerrisk
- "Unix Network Programming" by W. Richard Stevens

Online Platforms

- CS:APP Labs
- Operating Systems: Three Easy Pieces
- Linux Kernel Documentation
- GNU C Library Documentation

Practice Platforms

- HackerRank Systems Programming
- LeetCode System Design
- GitHub Systems Programming Challenges

Final Notes

This course outline is designed to take you from systems programming fundamentals to advanced topics over approximately 30 weeks of study. Each module builds upon the previous ones, so it's important to master the concepts before moving on.

Remember to:

- Practice coding regularly
- Work on real systems (Linux/Unix)
- Read man pages extensively
- Debug your programs thoroughly
- Collaborate with peers when possible
- Keep up with modern developments in systems programming

Good luck with your studies!