Princeton University
COS 217: Introduction to Programming Systems
Fall 2008 Final Exam Preparation

Topics

*You are responsible for all material covered in lectures, precepts, assignments, and required readings. This is a non-exhaustive list of topics that were covered. Topics that were covered after the midterm exam are in **boldface**.*

1. Number Systems

   - The binary, octal, and hexadecimal number systems
   - Finite representation of integers
   - Representation of negative integers
   - Binary arithmetic
   - Bitwise operators

2. C Programming

   - The program preparation process: preprocess, compile, assemble, link
   - Program structure: multi-file programs using header files
   - Process memory layout: text, stack, heap, rodata, data, bss sections
   - Data types
   - Variable declarations and definitions
   - Variable scope, linkage, and duration/extent
   - Constants: #define, constant variables, enumerations
   - Operators and statements
   - Function declarations and definitions
   - Pointers; call-by-reference
   - Arrays: arrays and pointers, arrays as parameters, strings
   - Command-line arguments
   - Input/output functions
   - Text files
   - Structures
   - Dynamic memory mgmt.: malloc(), calloc(), realloc(), free()
   - Dynamic memory mgmt. errors: dangling pointer, memory leak, double free
   - Abstract data types; opaque pointers
   - Void pointers
   - Function pointers and function callbacks
   - Parameterized macros and their dangers (see King Section 14.3)

3. Programming-in-the-Large

   - Testing
     - External testing taxonomy: boundary condition, statement, path, stress
     - Internal testing techniques: testing invariants, verifying conservation properties, checking function return values, changing code temporarily, leaving testing code intact
     - General testing strategies: testing incrementally, comparing implementations, automation, bug-driven testing, fault injection
   - Debugging heuristics

    - Understand error messages, think before writing, look for familiar bugs, divide and conquer, add more internal tests, display output, use a debugger, focus on recent changes
- Building
    - Automated builds, partial builds
- Performance improvement techniques
    - Execution efficiency: do timing studies, identify hot spots, use a better algorithm or data structure, enable compiler speed optimization, tune the code
    - Space efficiency: use a smaller data type, compute instead of storing, enable compiler size optimization
- Program and programming style
    - Top-down design
- Data structures and algorithms
    - Linked lists, hash tables, memory ownership
- Module qualities:
    - Separates interface and implementation, encapsulates data, manages resources consistently, is consistent, has a minimal interface, reports errors to clients, establishes contracts, has strong cohesion, has weak coupling
- Generics
    - Generic data structures via void pointers, generic algorithms via function pointers, wrappers
- **Portable programming**
    - **General heuristics**
    - **Heuristics related to differences in hardware, operating systems, compilers, libraries, and cultures**

4. Under the Hood: Toward the Hardware

- **Computer architectures and the IA-32 computer architecture**
    - **The Von Neumann architecture**
    - **Control unit vs. ALU**
    - **Little-endian vs. big-endian byte order**
    - **Language levels: high-level vs. assembly vs. machine**
- **Assembly languages and the IA-32 assembly language**
    - **Directives (.section, .asciz, .long, etc.)**
    - **Mnemonics (movl, addl, call, etc.)**
    - **Jump instructions and condition codes**
    - **Instruction operands: immediate, register, memory**
    - **Memory addressing modes: direct, indirect, indexed, base pointer**
    - **The stack and local variables**
    - **The stack and function calls: the C function call convention**
- **Machine language**
    - **Opcodes**
    - **The ModR/M byte**
    - **The SIB byte**
    - **Immediate, register, memory, displacement operands**
- **Assemblers**
    - **The forward reference problem**
    - **Pass 1: Create symbol table**
    - **Pass 2: Use symbol table to generate data section, rodata section, bss section, text section, relocation records**
- **Linkers**
    - **Resolution: Fetch library code**
    - **Relocation: Use relocation records and symbol table to patch code**

5. **Under the Hood: Toward the Operating System**

- **Virtual Memory**
  - **The memory hierarchy: registers vs. cache vs. memory vs. local secondary storage vs. remote secondary storage**
  - **Locality of reference**
  - **Page faults**
- **Dynamic memory management**
  - **Memory allocation strategies**
  - **Free block management**
  - **Optimizing malloc() and free()**
- **Unix system calls**
  - **For process control**
    - **The process abstraction**
    - **The process lifecycle**
    - **Context switches**
    - **The getpid(), execvp(), fork(), and wait() system calls**
    - **The exit() and system() functions**
  - **For interacting with the file system**
    - **The stream abstraction**
    - **The open(), creat(), close(), read(), write(), and lseek() system calls**
  - **For inter-process communication**
    - **The dup(), dup2(), and pipe() system calls**
- **Unix signals**
  - **Sending signals via keystrokes, the kill command, and the raise() and kill() functions**
  - **Installing signal handler functions: the signal() and sigaction() functions**
  - **Ignoring signals**
  - **Race conditions**
  - **Blocking signals: the sigprocmask() function**
- **Unix alarms and timers**
  - **The alarm() function**

6. Applications

- De-commenting
- Lexical analysis via finite state automata
- String manipulation
- Symbol tables, linked lists, hash tables
- Dynamically expanding arrays
- Buffer overrun attacks
- Unix shells

7. Tools

- The Unix/GNU programming environment
  - The Make tool

Readings

*As specified by the course "Schedule" Web page. Readings that were assigned after the midterm exam are in **boldface**.*

Required:

- *C Programming* (King): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22
- *Computer Systems* (Bryant & O'Hallaron): 1, **3 (OK to skip 3.14 and 3.15), 8, 10**
- ***Communications of the ACM* "Detection and Prevention of Stack Buffer Overflow Attacks"**
- ***The C Programming Language* (Kernighan & Ritchie) 8.7**

Recommended:

- *Computer Systems* (Bryant & O'Hallaron): 2, **5, 7, 11**
- *The Practice of Programming* (Kernighan & Pike): 1, 2, 4, 5, 6, 7, 8
- *Programming with GNU Software* (Loukides & Oram): 1, 2, 3, 4, 6, 7, 8, 9