

COMP 2401 Introduction to Systems Programming

Study Session Questions

December 27, 2025

CH 1–2: Bits, Bytes, and Strings

1. Select all the bit models that support representing both positive and negative integer values.
 - ☐ Magnitude-only Bit Model
 - ☐ Sign-Magnitude Bit Model
 - ☐ Two's Complement Bit Model
 - ☐ Fixed-Point Bit Model
 - ☐ Floating-Point Bit Model
 - ☐ ASCII and Unicode Bit Model
2. An 8-bit representation of 10010011 could represent all of the following values. Select all that apply.
 - ☐ hexadecimal 0x93
 - ☐ hexadecimal 0x39
 - ☐ signed decimal -109
 - ☐ unsigned decimal 147
 - ☐ ASCII character F
 - ☐ ASCII character @
3. Consider unsigned integer variables `x` and `y` with:
`x = 00110110` and `y = 11001100`
Which of the following statements are CORRECT? Select all that apply.
 - ☐ `printf("%d\n", x & y); = 4`
 - ☐ `printf("%d\n", x | y); = 254`
 - ☐ `printf("%d\n", x & ~y); = 6`
 - ☐ `printf("%d\n", x | (y » 4)); = 58`
 - ☐ `printf("%d\n", (x » 3) & (y « 1)); = 24`
 - ☐ `printf("%d\n", (x ^ y) » 1); = 125`
4. Which of the following declarations define a valid null-terminated string containing "hello"?
Select all that apply.

- ☐ `char s[5] = "hello";`
- ☐ `char *s = "hello";`
- ☐ `char s[6] = "hello";`
- ☐ `char s[] = "hello";`
- ☐ `char s[10] = "hello";`
- ☐ `char s[] = {'h','e','l','l','o'};`

5. What is the exact output of this program? Type it below.

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char buf[20];

    strcpy(buf, "COMP");
    strncat(buf, "2401A", 4);

    int len = strlen(buf);
    sprintf(buf + len, "%d", len);

    int cmp = strcmp(buf, "COMP2401_8");

    printf("%s+%d", buf, cmp);
    return 0;
}
```

6. On a little-endian machine, what will this program print? Type it below.

```
#include <stdio.h>
#include <string.h>

typedef union {
    unsigned int i;
    unsigned char b[4];
} U;

int main(void) {
    U u;
    u.b[0] = 0x11;
```

```
    u.b[1] = 0x22;
    u.b[2] = 0x33;
    u.b[3] = 0x44;
    printf("0x%x", u.i);
    return 0;
}
```

9. On a 64-bit Linux machine (System V ABI), what will the following program print?

```
#include <stdio.h>

struct A {
    char c;
    int i;
};

struct B {
    int i;
    char c;
};

int main(void) {
    printf("%lu %lu\n", sizeof(struct A), sizeof(struct B));
    return 0;
}
```

- A. 16 12
- B. 20 16
- C. 20 12
- D. 14 14

CH 3: Pointers, Stack/Heap, and Memory Allocation

10. Match each memory area in C to what it stores.

- | | |
|------------------|---|
| 1. Data segment | A. stores global variables and static variables |
| 2. Code segment | B. stores program instructions and addresses of functions |
| 3. Heap segment | C. stores dynamically-allocated memory |
| 4. Stack segment | D. stores local variables and order of function calls |

11. What practices help prevent memory leaks in long-running C programs? Select all options that apply.

- ☐ always pairing each `malloc/calloc/realloc` with a matching `free()`
- ☐ using tools like Valgrind or AddressSanitizer to detect leaks
- ☐ zeroing out pointers immediately after `free(p)` (e.g., `p = NULL`)
- ☐ checking that `malloc()` returned non-null before writing to its block

12. Which of the following lines evaluate to 4? Select all that apply.

- ☐ line A
- ☐ line B
- ☐ line C
- ☐ line D
- ☐ line E
- ☐ line F

13. Match each function with what it's used for.

- | | |
|-------------------------|--|
| 1. <code>malloc</code> | A. used when you don't care about the initial contents of the memory |
| 2. <code>calloc</code> | B. used when you need zeroed memory (for arrays / structs) |
| 3. <code>realloc</code> | C. used when you need to grow or shrink an existing allocation |

14. Which of the following lines correctly declare a pointer to this function? Select any that apply.

- ☐ line A
- ☐ line B
- ☐ line C
- ☐ line D
- ☐ line E
- ☐ line F

15. Which one of these statements is true?

- A. Both `x` and `y` will be freed correctly.
 - B. Only `x` is freed; `y` is leaked.
 - C. Only `y` is freed; `x` is leaked.
 - D. Neither `x` nor `y` is freed.
16. Which one of these statements are true?
- A. `list` itself is stack-allocated; its elements live on the heap
 - B. both the `list` pointer and its elements live on the heap
 - C. `init_students` must use `struct Student **` to work
 - D. this leaks because `init_students` never `mallocs`
 - E. both the `list` pointer and its elements live on the stack
 - F. `list` itself is heap-allocated; its elements live on the stack
17. Fill in lines 12, 13, and 14 below with the correct implementation. Select the 3 answers that apply.
- ☐ `s.user = u;`
 - ☐ `s.user = &u;`
 - ☐ `s.user.uid = 13;`
 - ☐ `s.user->uid = 13;`
 - ☐ `strcpy(u.name, "Aaryan");`
 - ☐ `strcpy(u->name, "Aaryan");`
18. What does this code print?
- A. Compiler/Syntax Error
 - B. 0
 - C. 1
 - D. 2
 - E. 3
 - F. 4
19. Which line should be added to line X to prevent the memory leak?
- A. `free(score);`
 - B. `free(&score);`
 - C. `free(*score);`
 - D. `free(&*score);`
 - E. `score = NULL;`
 - F. The memory leak cannot be prevented in `main()`.

CH 4 & 7: Compilation, Linking, and Program Structure

20. Match each term with what it does.

- | | |
|-------------|--|
| 1. Editor | A. creates the source <code>.c</code> files |
| 2. Compiler | B. what <code>make</code> uses to translate C files into <code>.o</code> files |
| 3. Linker | C. combines object files and libraries into the executable |
| 4. Loader | D. executes the resulting binary on the desired platform |

21. Match each variable keyword category with its purpose.

- | | |
|----------------------------|--|
| 1. storage class specifier | A. specifies where a variable lives (i.e. <code>static</code> , <code>extern</code> , <code>register</code>) |
| 2. type qualifier | B. tells the compiler about special constraints (i.e. <code>const</code> , <code>volatile</code>) |
| 3. type modifier | C. defines how a data type's bits are interpreted (i.e. <code>unsigned</code> , <code>long</code>) |
| 4. base type | D. defines what kind of value a variable holds (i.e. <code>int</code> , <code>float</code> , <code>char</code>) |

22. A correct dependency-aware Makefile will re-compile:

- A. all `.c` files in the directory, every time you invoke `make`
- B. only the `.c` files you list on the `make` command line
- C. only those files whose corresponding `.c` file is newer than the `.o` file
- D. only those `.c` files whose corresponding `.o` file is newer than the `.c` file
- E. only those `.c` files without an associated `.o` file

23. Write an exact shell command to compile and link `x.c` and `y.c` into an executable named `file`.

24. Which of the following should go in header files (`.h`)? Select all that apply.

- ☐ global constant declarations
- ☐ forward declarations of function prototypes
- ☐ global type definitions
- ☐ function implementations
- ☐ global variable declarations
- ☐ the `main` program to start the code

25. Which scenarios are examples of concurrent computing? Select all that apply.

- ☐ a program uses `pthread`s to parallelize matrix multiplication
- ☐ a single-threaded bash script reads, processes, then writes output data
- ☐ an HTTP server `fork()`s each connection into a new process
- ☐ a program uses `malloc()` to compute Fibonacci numbers in order
- ☐ a utility processes log files sequentially in a loop
- ☐ an MPI application distributes tasks across multiple hosts

CH 5: Processes, Threads, and Synchronization

26. Match the problem in concurrent systems with its scenario.

- | | |
|-------------------|--|
| 1. Contention | A. many threads wait on a mutex held for too long, slowing throughput |
| 2. Deadlock | B. thread A holds M1 and waits for M2, while thread B holds M2 and waits for M1 |
| 3. Race condition | C. two threads increment a shared counter without locking, producing an incorrect final count |
| 4. Starvation | D. a low-priority thread never runs because higher-priority threads keep getting the CPU |

27. Select the statements that are TRUE about sharing between threads and processes.

- ☐ threads in a process share the same virtual memory but use separate stacks
- ☐ each process has its own independent virtual memory
- ☐ every thread has its own separate copy of global variables
- ☐ one process can directly read/write another process's heap without IPC
- ☐ threads share OS resources like file descriptors and signal handlers
- ☐ `fork()` is typically faster than `pthread_create()`.

28. Match each process-management system call with its definition.

- | | |
|------------------------|---|
| 1. <code>exec</code> | A. replaces the current process image with a new program's image |
| 2. <code>fork</code> | B. creates a child process that is an (almost) exact duplicate of the parent |
| 3. <code>system</code> | C. runs a shell command string and waits for it to finish |
| 4. <code>wait</code> | D. pauses the parent until one of its children terminates |

29. Sockets, signals, semaphores; select the sound statements swiftly.

- ☐ `SIGKILL` can be caught and blocked by a process.
- ☐ `sem_post()` decrements the semaphore count.
- ☐ `kill()` sends a signal to a specified process.

- ☐ Named semaphores can be shared across processes.
 - ☐ Sockets support both stream and datagram modes.
 - ☐ Signals are async notifications sent to processes.
30. Match each scenario with the IPC mechanism best-suited for it.
- | | |
|--|------------------|
| 1. notify a process asynchronously when an event occurs | A. signals |
| 2. ensure exclusive access to shared resources between processes | B. semaphores |
| 3. communicate reliably between processes on different hosts | C. sockets |
| 4. share a large data buffer efficiently among local processes | D. shared memory |
31. Select the true statements about TCP and UDP.
- ☐ TCP is a connection-oriented protocol that ensures ordered data delivery.
 - ☐ UDP is connectionless and provides no guarantees on reliability/ordering.
 - ☐ TCP is preferred for real-time/broadcast scenarios (like VoIP or streaming).
 - ☐ UDP only handles one-to-one connections.
 - ☐ TCP establishes a session via a three-way handshake before any data is sent.
 - ☐ UDP automatically retransmits lost packets until they are acknowledged.

CH 6: File I/O, Buffering, and Libraries

32. Match the buffering mode with a typical use case.
- | | |
|-------------------|---|
| 1. unbuffered | A. error messages via <code>stderr</code> |
| 2. line-buffered | B. interactive <code>stdout</code> and <code>stdin</code> in a terminal session |
| 3. fully buffered | C. bulk data writes to files or network sockets |
33. Select all the true statements about file I/O and buffering.
- ☐ In the course VM, the standard output `stdout` is unbuffered.
 - ☐ `feof()` only returns true after an attempted read has reached EOF.
 - ☐ Calling `fclose(stream)` will auto-flush its buffer before closing.
 - ☐ `fscanf()` can be used on both text and binary files.
34. Sequence these calls to read the 5th record from a binary file (called `data.bin`) into a struct `rec`. Type the correct sequence below (e.g., 3 1 4 2).
- (1) `if (!fp) { return EXIT_FAILURE; }`
 - (2) `fread(&rec, sizeof(Record), 1, fp);`
 - (3) `FILE *fp = fopen("data.bin", "rb");`
 - (4) `fseek(fp, sizeof(Record) * 4, SEEK_SET);`

35. Which of the following are *sinks* in I/O terminology?

- A. Keyboard (`stdin`)
- B. File opened with mode "`r`"
- C. Socket opened for reading
- D. File opened with mode "`w`"

36. What will be the output of this code? Type it below.