ECE391: Computer Systems Engineering **Fall 2023**
Midterm 1 Study Guide

Topics and Sample Questions for the First Midterm

This guide contains example questions to help you study for the upcoming midterm. The first page and a half list the topics and terminology that we intend to test with the exam. Roughly speaking, all materials covered in class, in the notes (to the point reached in class before the test), on the problem sets, or as part of MP1 are fair game. The rest of the guide provides sample questions. Most of the questions here are short answer, whereas the test will include a larger portion of questions that require some code or analysis of code. The questions here are also not as carefully designed for clarity, in part so as to encourage you to think about different angles. On tests, we try to make our questions more precise. We suggest that you also attempt the previous exams available to you on the class web page.

In general, we are less interested in whether you can memorize system-specific details than in whether you understand how systems are designed and why they are designed in a particular way. For example, the "tools and class environment" may seem a little vague if one assumes that we could ask obscure questions about command or implementation details. Rather, we may ask questions like, "How does source control help programmers to better manage software projects?" On the flip side, vague answers to such a question, for example, "It helps to control the source code," will receive no credit.

You may bring one 8.5x11" sheet (both sides) of **handwritten** notes to the test. No other materials are allowed. No calculators or other electronic devices are allowed.

- tools and class environment
  - virtual machine environment QEMU
  - ? Makefiles (automatic dependency tracking)
  - source control gitlab
  - source debuggers gdb
  - debugging and test methods used in MP1
- x86 assembly
  - data types and conversion
  - ? simple data structures (arrays, linked lists, structs)
  - calling convention
  - linkage (such as between C and assembly)
  - inline assembly code (in GCC)
- indirection
  - vector/jump tables
  - VGA palette
  - lookup tables for device abstraction
- interrupts, exceptions, and system calls
  - differences and similarities
  - how/when they occur (and implications)
  - source of vector numbers
- synchronization and critical sections
  - shared resources: types, methods for handling
  - concept and need for atomicity
  - lock functionality (basic protocol and implementation)
  - semaphores
  - reader/writer synchronization
  - systematic approach for shared memory data
  - synchronization problems: unordered ? locks, deadlock, livelock, starvation
  - single-producer, single-consumer queues
- PIC
  - design and motivation
  - cascading: rationale and issues
  - system integration (how PICs are integrated into a computer)

The terminology list below is replicated from the second set of notes.

- procedural abstractions
  - system call
  - interrupt
  - exception
  - handler (function)
- processor support
  - vector table
  - Interrupt Descriptor Table (IDT)
  - interrupt enable flag (IF)
  - non-maskable interrupt (NMI)
- input/output concepts
  - I/O port space
  - independent vs. memory-mapped I/O
- software and programming abstractions
  - application programming interface (API)
  - linkage
  - jump tables
- synchronization concepts
  - critical section
  - atomicity
- synchronization mechanisms
  - mutex
  - spin lock: lock/obtain and unlock/release operations
  - semaphore: down and up operations
  - reader/writer lock
- synchronization problems
  - race condition
  - deadlock
  - starvation
- interface between processor and devices
  - interrupt controller
  - Programmable Interrupt Controller (PIC), such as Intel's 8259A
  - end-of-interrupt (EOI) signal
- interrupt abstractions in software
  - interrupt chaining
  - hard vs. soft interrupts
  - Linux' tasklet abstraction

The remainder of this guide provides sample questions.

1. As you may recall, QEMU is a simulator that runs most of the simulated code on the hardware itself, resorting to software emulation only for those instructions for which the additional level of virtualization makes emulation necessary. For example, a MOV instruction that writes into a memory-mapped I/O register on the emulated network card, telling it to send a packet, must be handled by delivering the packet to the real network card. Compare the performance of this type of simulation with both full software emulation and with execution without a virtual machine layer, then describe a couple of advantages of using a virtual machine over using the computer hardware directly.

   faster than full software emulation
   slower than without virtual machine
   ① less probability to harm hardware
   ② easier to test the code

2. Explain the basic abstraction supported by the `make` tool and give an example of how it can be used to express the process of compiling a software system consisting of several headers and source code files.

3. Describe two scenarios in which use of a source control tool can make a programmer's life easier.

   ① A group of stu writing codes together ② lots of bug in lots of codes

4. You compile a program, and it crashes due to a segmentation violation error. Estimate the time necessary to find the source code line that generated the error using print statements, assuming that the error is completely deterministic and that the program otherwise generates no output. State any assumptions that you find it necessary to make. Do the same exercise assuming the use of a symbolic debugger. Exact numbers are not required - simply describe what you would need to do to find the bug using each process.

5. Breaking a software system into modules that can be tested individually can reduce the complexity of testing and debugging. Are there drawbacks to such design? If so, describe one.

   The amount of code will increase as we need to add the interface between them.

6. Why is processor support for multiple data sizes useful (for example, 8-, 16-, and 32-bit 2's complement data types)?

   Because some old programs only use 8 bit architecture this can make them compatible.

7. Explain or write x86 assembly (or RTL at the level of individual instructions) to find a field at offset 2 bytes of the element N in an array of structures consisting of 8 bytes each. Do the same for a linked list of structures, in which each structure described above has been extended with a 4-byte `next` pointer (at the end).

   fields?

   movl $8, %EDX
   IMULL $N, %EDX, %EDX
   MOVL 8(%EBP), %EAX

```
ADDL  %EDX,%EAX
MOVL  2(%EAX),%EAX
        ans
```

8. Translate the following C function into x86 assembly. Do not optimize the assembly.

```
int
max_of_three (int a, int b, int c)
{
    int first;

    if (a > b)
        first = a;
    else
        first = b;
    if (first > c)
        return first;
    return c;
}
```

```
push  %ebp
movl  %esp, %ebp
subl  $12, %esp
movl  8(%ebp), %eax
movl  %eax, -4(%ebp)
movl  12(%ebp), %eax
movl  %eax, -8(%ebp)
movl  16(%esp), %eax
movl  %eax, -12(%ebp)
movl  -4(%ebp),%eax
movl  -8(%ebp),%edx
cmpl  %eax,%edx
jl    compare_c
movl  %edx,%eax
compare_c:
  movl -12(%ebp),%edx
  cmpl %eax,%edx
  jl   ret
  movl %edx,%eax
ret:
  addl $12,%esp
  RET
LEAVE
```

9. Given a pointer to an array of integers in EBX and the length of the array in ECX, write x86 assembly code that uses the `max_of_three` function from the previous problem to find the largest value in the array.

```
9.
      push  %esi
      movl  $0x80000000, %ebi
loop: cmpl  $0, %ECX
      je    exit
      movl  (%EBX), %EAX
      push  %esi
      push  %EAX
      push  $0
      call  max_of_three
      add   $12,%esp
      movl  %EAX, %ebi
      addl  $1, %EBX
      subl  $1, %ECX
      jmp   loop
ret:
      movl  ebi, %EAX
      pop   %esi
```

10. Write the assembly code generated for the body (not including the stack frame creation/tear down) of the following function, then explain its return value.

```
int
mystery (int value)
{
    int rval;

    asm volatile ("
        movl %1,%0
        subl $1,%0
        andl %1,%0
        je 1f           # to label "1", where "f" means forward
        movl $1,%0
      1:
    " :  "=r" (rval)
      :  "r" (value)
      :  "cc" );

    return rval;
}
```

11. Explain the differences between disabling an interrupt in software and executing CLI.
all processor may not be interrupt; target on one processor

12. While developing a storage model for a personal collection of MP3 songs, you opt to store a binary version of the structure used to describe each song in the collection. What happens when someone adds a new field to the structure, and what might you do to reduce the potential impact of such a change?
structure will break it use array   use linked list

13. When does the direction used in copying data from one place in memory to another matter?

14. Some of the higher resolution color modes supported by our emulated video cards support 24-bit color, in which each pixel is specified by 8-bit red, green, and blue values. Suggest two C data structures that you could use to hold a 16-pixel-wide by 12-pixel-high image for such a mode.

15. Why is double-buffering useful? make the frames look smoother

16. Explain how a jump table is defined and used in a program. arrays contain pointer to each function add offset then go there

17. If color bits could be packed tightly into bytes, how much space is saved by using an 8-bit palette rather than individual pixel colors (6-bit R, G, and B) in mode X (320 by 200 pixels)? How much more space is needed to extend the color space to 24-bit (8:8:8 R:G:B) using the same palette?

$320 \times 200 \times (3 \times 6 - 8)$

$320 \times 200 \times (3 \times 8 - 8)$

Flags   interruption occur between   cmpl je

18. What processor state must be saved when an exception occurs. Justify your answer with an example in which that state proves useful.

19. Are system calls synchronous or asynchonous with respect to a program? Are they expected or unexpected? Explain any implications of your answers.

Already used by Intel

20. Your friend suggests that mapping the 8259A interrupts into the 0x00 to 0x0F range of the Interrupt Descriptor Table can save time on translations between vector number and IRQ. Explain why such a mapping does not work as expected.

21. ? As part of debugging your device driver, you decide to count the number of command requests made to the device that have yet to generate an interrupt. One approach to doing so is to use a shared variable incremented by the command request code and decremented by the interrupt handler. Suggest a better method that does not involve sharing data between these two pieces of code. modify PIL and store vari in PIC?

22. Suppose that you had decided to go with the increment/decrement approach in the previous problem. You declare a shared integer variable, `n_outstanding`, and execute `n_outstanding++` in the command request code and `n_outstanding--` in the interrupt handler. Explain the meaning of the term atomicity in this context, why atomicity is necessary to make this code work properly, and what mechanisms you can use to ensure that atomicity. be not interrupted ? use CLI STI to modify IF

23. Three programs running on a multiprocessor arrive at critical sections protected by a common lock. Assuming that program A arrives first, followed by B, followed by C, and that a Linux spin lock is used to protect the critical sections, list the possible orderings of critical section executions (for example, C followed by B followed by A, which is not possible) and explain why the order is not uniquely determined by the order of arrival at the lock calls. A B C or A C B us after A, B-C may wait together

24. Describe an example in which a semaphore is useful. Explain the drawbacks of alternatives that use only mutually exclusive locks. when the number of devices to get shared sources is determined

increasing waiting time decreases performace / use semaphore

25. What are the rules for obtaining a reader/writer lock for read access? What about write access? If a reader/writer lock implementation never prevents accesses that obey these rules, what synchronization problem can occur when the lock is heavily contended (in other words, has many programs trying to use it)? no write when no reader

shared data broken

26. Consider the dining philosophers problem, in which five philosophers sit around a round table with one chopstick between each pair. Each philosopher picks up the chopstick on her left, then picks up the chopstick on her right, eats, and puts the two chopsticks down. The chopsticks are analogous to locks. Explain the meaning of deadlock, how this protocol can result in deadlock, and how you can fix the problem using an ordering on the locks. blocks

everyone hold left chopstick waiting ?

27. ? Describe the assembly linkage used to map hard interrupts into calls to `do_IRQ`.

waits more time   easier to trade dead locks

28. Linux 2.0 had a single, big kernel lock. Linux 2.2 (and later versions) has many more locks protecting smaller amounts of shared data. Explain the tradeoffs. when two shared data need each other

29. Why can you not use semaphores within an interrupt handler in Linux?

semaphores allow other program use this processor breaking interrupt

30. ? When should the *_irq version of spinlock functions be used? When should the *_irqsave/*_irqrestore be used?

31. What are the differences between spinlocks and semaphores?

not allow   allow, [illegible]

~~32.~~ Describe an example in which a semaphore is useful. Explain the drawbacks of alternatives that use only mutually exclusive locks.

33. Determine the potential data sharing conflicts between functions acting on the same account in the code below, then use locks associated with the accounts to prevent any problems from arising. Use enough locks to allow maximal parallel execution of the functions.

lock init

```
int
get_balance (account_t* a)
{
    int rval;        <get lock
    rval = a->assets - a->debt;   <interrupt here
    if (rval < 0) {
        /* apply overdraft fee */
        a->debt += 20;
        a->n_fees++;
    }   < release lock
    return rval;
}

void
deposit (account_t* a, int amt)
{
    a->assets += amt;
}

void
withdraw (account_t* a, int amt)
{
    a->debt += amt;
}

int
pay_debt (account_t* a, int amt)
{
    if (a->debt < amt || a->assets < amt)
        return -1;
    a->assets -= amt;
    a->debt -= amt;
    return 0;
}

int
check_penalties (account_t* a)
{
    return a->n_fees;
}
```

find which device interrupt

handle multiple interruption in a order specific

34. Explain two advantages of using an interrupt controller over an OR gate for bringing device interrupt lines into a processor's interrupt input.

35. Why is the question of the vector number to be used for an interrupt not clear to any given PIC in a cascaded set of PICs, and how is this issue addressed in the 8259A design? use jump table

模拟化

36. Draw logic to connect an 8259A to ports (0x43 and 0x44). Note that the first port is odd, not even.

37. How does software disablement of hard interrupts work in Linux? Explain also how software disablement is supported if the system's PIC does not support masking of individual interrupts. In particular, how is a hard interrupt deferred, and when and how is it eventually executed?

38. Explain the benefits of using linked lists of handlers (actions) to support interrupt chaining. Comment on the drawbacks of chained handlers in general.