Midterm 1 Practice Questions

Q. You translated the following code snippet into x86 but it doesn't work as expected. Explain why.

```
uint32_t recursive (uint32_t  num) {
    if (num <= 1) return 1;
    return num + recursive(num >> 1);
}
```

```
recursive:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %ecx
    cmpl $1, %ecx
    jbe done
    movl %ecx, %edx   ①
    shrl $1, %edx
    pushl %edx
    call recursive
    addl $4, %esp
    addl %ecx, %eax   ②
    leave
    ret
done:
    movl $1, %eax
    leave
    ret
```

Code not save caller saved register ECX

so value of ecx is different for line ① or ②

Q. Translate the purposely convoluted x86 code snippet into a functionally equivalent C function. The function body is one line. Assume there isn't any integer overflow.

```
mystery:
    pushl %ebx
    leal 4(%esp), %ebx
    addl $4, %ebx
    movl (%ebx), %ebx
    shll $2, %ebx
    movl %ebx, %eax
    popl %ebx
    ret
```

```
int mystery (int para){
    return (para*4);
}
```

para<<2

```
int mystery (int num){
  switch(num){
  case 0:
    return func1();
  case 1:
    return func2();
```

Q. What is the following x86 code snippet functionally equivalent to?

```
  case 2:
    return func3();
  }
```

```
mystery:
    movl 4(%esp), %edx     num
    leal functions, %ecx   ecx
    shll $2, %edx
    addl %edx, %ecx
    jmp (%ecx)

functions:
    .long func1, func2, func3
```

```
int mystery (int num){...}
```

equivalent to a jumptable where num means (num+1)th function

jump to jumptable [0] = func1
1 2
2 3

Q. Let's implement a doubly linked list. Assume that nodes have the following form

```
VALUE = 0
NEXT = 4
PREV = 8
```

Fill in the blanks to insert `new_node` into the middle of a doubly linked list. Place it directly after the `cur` pointer. Eg. If `cur` pointed to B, then A ↔ B ↔ C would become A ↔ B ↔ X ↔ D

```
insert_middle:
    pushl %ebp
    pushl %esp, %ebp
    movl 8(%ebp), %ecx   # cur
    movl 12(%ebp), %edx  # new_node
    movl NEXT(%ecx), %eax  # next_node
    movl %ecx, PREV(%edx)
    movl %eax, NEXT(%edx)
    movl %edx, NEXT(%ecx)
    movl %edx, PREV(%eax)

    leave
    ret
```

Q. What is the correct sequence of events if we want to have our system handle IRQ6 and then IRQ1 (in that order) as fast as possible:

a. EOI is written to data bus
b. Raise IRQ1
c. Raise IRQ6
d. Processor executes corresponding IRQ6 handler
e. EOI is written to data bus
f. Processor executes corresponding IRQ1 handler

c. d. a. b. f. e.

because primary PIC has 8 IRQ lines

Q. Explain the role of the CAS bus. Why is it 3 bits? How does SP tie in to the cascaded scheme?

Used by the primary PIC to identify which secondary PIC should write to the data line

To differentiate the primary from the secondary PICs and to configure the CAS bus as either input or output

Q. There are 3 PICs, one primary and two secondary labeled PIC 1,2,3 accordingly. PIC 3 is connected to IRQ4 on PIC1 and PIC 2 is connected to IRQ2 on PIC1. PIC2 has its interrupts labeled A-H for its IRQ1->IRQ7, (meaning IRQ1 = A, IRQ2 = B… IRQ7 = H) and PIC3 has interrupts labeled I-P for its IRQ1->IRQ7. The Primary PIC has IRQ1 = U, IRQ3= T, IRQ5 = S, IRQ6 = R, and IRQ7 = Q.

PIC1 IRQ4 → PIC3 (R Q S / T; P / I)
IRQ2 → PIC2 (U; H / A)

Q. What is the correct priority order?

U > A > B > C > D > E > F > G > H > T > I > J > K > L > M > N > O > P > S > R > Q

Q. If we want Interrupt H to be serviced what should be raised on the CAS bus?

010

Q. Papa John decides that he wants to write a multiplayer game that is run in the linux kernel - the greatest snail racing game ever assembled. What makes it so great? Users can enter/exit the race whenever they wish! To do so, he decides that he will separate his game into two forms of state:

- Where each snail is located
- How many “power boosts” each snail has available to them. If a user requests a “power boost”, the snail will temporarily gain invulnerability and travel 3000x as fast.

Papa John realizes that he now has a computer systems problem on his hands - good thing he took ECE 391! He is able to identify the following points:

- To make running his game smoother with multiple players, he decides that he wants to run this on a *multicore system*
- A single kernel thread will maintain game state
- Each time a user is added, a new user process will run for them.
- All game state is stored in the kernel, and is shared by all users.
- Each user process will handle display graphics on their own - that portion of the implementation can be ignored going forwards.

He chooses to structure it in a similar way that his mentor, High Mage Kalbarczyk, told him to implement a retro missile firing game.

- Every so often, an RTC tasklet will run in the kernel, updating both forms of the game state, and allowing new users to join
- Users can use a set of *ioctls* to interact with the kernel. User processes are given 4 ioctls to begin with:
    - `enter_game`
    - `use_boost`
    - `request_entire_game_state`
    - `exit_game`
- Upon entering the game, the user process will initialize game state for the user. `use_boost` will overwrite that user’s boost counter. `request_entire_game_state` will read both the boost state and location states. `exit_game` will teardown the user’s game state, and remove them from the race.

However, it’s been a while since he’s worked on systems-level programming, and decides to hire you as a consultant. As you work with him on this system, you encounter the following challenges:

1) Papa John identifies various critical sections in his program where he wants nobody else to be able to interfere with the data. He wants to protect this region using a *cli* instruction before, and an *sti* instruction after the critical section. Is this alright?

No. STI and SLI can only prevent interfere from interruption, can not prevent interfere from other processors

2) Papa John doesn’t find the above option very appetizing, and decides he wants to use a more interesting implementation. He has heard that semaphores are a great way to lock data, and can be more efficient than spinlocks. Is Papa John right to consider using

No. Because the number of users accessible to the data can be changed as new users add in or users exit.

② 1) NO. Semaphore can't be [illegible]

~~② And semaphor can't be used by interruption handler RTL as it allow other program to run on this processor while waiting to be wake up.~~

semaphores?

3) Papa John realizes that he is completely lost, and has no idea how to synchronize his data. It is all up to *you* to synchronize the greatest snail racing game of all time. What locking mechanism do you choose for this game, and why?

Use R/W spinlock. Because it allows multi users read the shared data at the same time. And separating reading and writing process

(**Highlight the blacked out sections after finishing q3 - they've been censored so that you don't "spoil" q3's answer)**

R/W spinlock

4) After giving it some thought, you decide that a [redacted] is necessary to make this game work. You outsource the locking work to the greatest locksmith in all the land, Handyman Lumetta. Handyman Lumetta laments that you are forced to use a [redacted] instead of a [redacted]. What benefits are you missing out on? R/W spinlock R/W semaphor

R/W semaphor can prevent writer starvation

wont exit on exam

5) Handyman Lumetta is busy, and realizes that he will be able to work much faster if you are able to outline the locking requirements in your system. In whatever form is comfortable to you, outline the read/write dependencies of the kernel tasklet and each of the IOCTLs. | enter_game: write location and power boosts | exit_game: write ? | use_boost: read and write power boosts | Tasklet: read/write | request_entire_game_state: read all |

After Handyman Lumetta writes some specialized locking code for you, Papa John decides he's ready to get back in the game. He writes a prototype implementation of the game overnight, fueled by nothing but root beer and Daily Byte.

6) When you try to test his code, you find that it is running, but all of his ioctls are getting stuck in the section of the code where they try to acquire the lock. He swears up and down that every time a function acquires a lock, it is definitely releasing it when it's done. Why might this bug be happening?

The interruption caused by RTC tasklet happens after a user-level code acquire a lock, resulting into dead lock

7) After knowing what the issue is, ~~where might the issue be occurring?~~ How can you fix it?

Occurring after

mask interruption before acquiring lock, or use read_lock_irq ...

You find that Papa John is both reading *and* writing while acquiring only a writer lock. You decide to fix his code by acquiring both a reader *and* a writer lock in the same section of code, but now the game is hanging in all the writer functions.

8) High Mage Kalbarczyk chews you out for not paying attention in class. Why is your code breaking?

Writer lock can only be acquired after all reader locks have been released. So dead lock happens.

You and Papa John work together, and are able to publish your game. However, shortly after its release, you receive a call to small claims court for copyright infringement. Who sued you?