

# 241 Wrap-up lecture

# [ Today ]

- Announcements
- Practice questions for the exam
- malloc contest awards



# [ Announcements ]


- Did you have a grading exception?
  - Any “special case” that’s not on Compass
  - If you didn’t get a confirmation email from us within the last 24 hours, contact us to sort it out
- TAs’ review session: when?
  - Thu 2pm
  - Sat 2pm
  - Sun 2pm



# [ Announcements ]

- We have released practice study questions for the exam
  - Not representative of the style of the exam. Not comprehensive coverage of material.
  - From Fall 2005 final exam; but has all short answer (no multiple choice) and some different topics
  - See link in today's entry on the Schedule page on the web site





# Exam practice

# [ Pop quiz ]

- You use `sigprocmask()` to block the SIGKILL signal when your program starts running. What would happen if you send a SIGKILL signal to your running program by using `"kill -9 <your program>"`?
  - (a) Your program continues running.
  - (b) Your program terminates.
  - (c) It depends on what happens in the signal handler.



# [ Pop quiz ]

- What happens when the OS uses a smaller page size?
  - Less wasted memory
  - The page table becomes larger
  - The maximum amount of memory a program can allocate is reduced.
  - Pages can be swapped in more quickly.
  - Pointers may not be large enough to address all the pages, especially on 32-bit architectures.



# [ Practice questions ]

- What is the difference between a physical memory address and a virtual memory address?





# [ Practice questions ]

- Explain the difference between a library function and a system call.
  - Which type are malloc, pipe?
  - Which is faster, and why?



# [ Practice questions ]

- Why is it generally bad practice to acquire a lock in a signal handler?




# [ Practice questions ]

- What is select() good for?



# [ Practice questions

## ■ Does it deadlock?



```
/* Thread 1 */  
lock(mutex_a)  
lock(mutex_b)  
b = a  
unlock(mutex_a)  
lock(mutex_c)  
b = b + c  
unlock(mutex_c)  
unlock(mutex_b)
```

```
/* Thread 2 */  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```

If Thread 1 got stuck at its first lock, then it must be because Thread 2 is somewhere here. But from this point Thread 2 will clearly finish. So Thread 1 won't get permanently stuck at its first lock.



# [ Practice questions ]

- Does it deadlock?

```
/* Thread 1 */  
lock(mutex_a)  
lock(mutex_b)  
b = a  
unlock(mutex_a)  
lock(mutex_c)  
b = b + c  
unlock(mutex_c)  
unlock(mutex_b)
```

```
/* Thread 2 */  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```



# [ Practice questions ]

- Does it deadlock?

```
/* Thread 1 */  
lock(mutex_a)  
lock(mutex_b)  
b = a  
unlock(mutex_a)  
→ lock(mutex_c)  
b = b + c  
unlock(mutex_c)  
unlock(mutex_b)
```

```
/* Thread 2 */  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```



# [ Practice questions ]

- Does it deadlock?

```
/* Thread 1 */  
lock(mutex_a)  
lock(mutex_b)  
b = a  
unlock(mutex_a)  
lock(mutex_c)  
b = b + c  
unlock(mutex_c)  
unlock(mutex_b)
```

```
/* Thread 2 */  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```

**...done!**



# [ Practice questions ]

## ■ Does it deadlock?

**/\* Thread 1 \*/**

lock(mutex\_a)

lock(mutex\_b)

b = a

unlock(mutex\_a)

lock(mutex\_c)

b = b + c

unlock(mutex\_c)

unlock(mutex\_b)

**/\* Thread 2 \*/**

lock(mutex\_b)

lock(mutex\_c)

c = b

unlock(mutex\_b)

lock(mutex\_a)

a = c \* 2

unlock(mutex\_a)

unlock(mutex\_c)

**/\* Thread 2 too**

lock(mutex\_b)

lock(mutex\_c)

c = b

unlock(mutex\_b)

lock(mutex\_a)

a = c \* 2

unlock(mutex\_a)

unlock(mutex\_c)





# [ Practice questions ]

## ■ Does it deadlock?

```
/* Thread 1 */  
lock(mutex_a)  
lock(mutex_b)  
b = a  
unlock(mutex_a)  
lock(mutex_c)  
b = b + c  
unlock(mutex_c)  
unlock(mutex_b)
```

```
/* Thread 2 */  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```

```
/* Thread 2 too  
lock(mutex_b)  
lock(mutex_c)  
c = b  
unlock(mutex_b)  
lock(mutex_a)  
a = c * 2  
unlock(mutex_a)  
unlock(mutex_c)
```

Deadlock!



# [ Crowdsourcing ]

- Get together with your neighbor
- Come up with an exam question
  - No need to produce the answer!
- Then, we'll talk about some
- The rest: You're encouraged to post them to the newsgroup & discuss





# Malloc awards session!