CSci 402 - Operating Systems

Midterm Exam (DEN Section)

Fall 2020

(11:00:00am - 11:40:00am, Wednesday, October 7)

Instructor: Bill Cheng

Teaching Assistant: Ben Yan

(This exam is open book and open notes.
Remember what you have promised when you signed your
Academic Integrity Honor Code Pledge.)

This study resource was shared via CourseHero.com

Time: 40 minutes

_____-
Name (please print)

Total: 32 points

_____-
Signature

Instructions

1. This is the first page of your exam. The previous page is a title page and does not have a page number. Since this is a take-home exam, no need to sign above since you won't submit this file.
2. Read problem descriptions carefully. You may not receive any credit if you answer the wrong question. Furthermore, if a problem says *"in N words or less"*, use that as a hint that N words or less are expected in the answer (your answer can be longer if you want). Please note that points may get *deducted* if you put in wrong stuff in your answer.
3. If a question doesn't say `weenix`, please do not give `weenix`-specific answers.
4. Write answers to all problems in the **answers text file**.
5. For non-multiple-choice and non-fill-in-the blank questions, please show all work (if applicable and appropriate). If you cannot finish a problem, your written work may help us to give you partial credit. We may not give full credit for answers only (i.e., for answers that do not show any work). Grading can only be based on what you wrote and cannot be based on what's on your mind when you wrote your answers.
6. Please do *not* just draw pictures to answer questions (unless you are specifically asked to draw pictures). Pictures will not be considered for grading unless they are clearly explained with words, equations, and/or formulas. It's very difficult to draw pictures in a text file and you are not permitted to submit additional files other than the answers text file.
7. For problems that have multiple parts, please clearly *label* which part you are providing answers for.
8. Please ignore minor spelling and grammatical errors. They do not make an answer invalid or incorrect.
9. During the exam, please only ask questions to *clarify* problems. Questions such as "would it be okay if I answer it this way" will not be answered (unless it can be answered to the whole class). Also, you are suppose to know the definitions and abbreviations/acronyms of *all technical terms*. We cannot "clarify" them for you. We also will **not** answer any clarification-type question for multiple choice problems since that would often give answers away.
10. Unless otherwise specified and stated explicitly, multiple choice questions have one or more correct answers. You will get points for selecting correct ones and you will lose points for selecting wrong ones.
11. When we grade your exam, we must assume that you wrote what you meant and you meant what you wrote. So, please write your answers accordingly.

This study resource was shared via CourseHero.com

https://www.coursehero.com/file/75357404/midterm-exampdf/

(Q1) (2 points) For the x86 processor, the **switch()** function is depicted below:

```
void switch(thread_t *next_thread) {
  CurrentThread->SP = SP;
  CurrentThread = next_thread;
  SP = CurrentThread->SP;
}
```

In what way is this **switch()** function different from a regular function?

(1) one thread enters the function and a different thread leaves the function
(2) this function does not touch the ebp register
(3) this function uses no local variables and that's unusual
(4) this function is unusually short
(5) none of the above is correct

Answer (just give numbers): 1

(Q2) (2 points) Every device in Unix is identified by a major device number and a minor device number. Which of the following statements are true about these device numbers?

(1) minor device number must be smaller than the major device number
(2) minor device number is rarely used in the OS
(3) minor device number identifies a device driver
(4) minor device number indicated which process is currently using the device
(5) none of the above is correct

Answer (just give numbers): 5

(Q3) (2 points) Comparing cancellation in the **weenix** kernel and pthreads cancellation, which of the following statements are correct?

(1) **weenix** kernel cancellation "state" is always enabled and "type" is always deferred
(2) just like pthreads, you can change **weenix** kernel cancellation "type" (asynchronous/deferred) programmatically
(3) since the **weenix** kernel is non-preemptive, it cannot have cancellation points
(4) just like pthreads, you can change **weenix** kernel cancellation "state" (enabled/disable) programmatically
(5) none of the above is correct

Answer (just give numbers): 5 1!

This study resource was shared via CourseHero.com

(Q4) (2 points) When a user thread makes a system call, it becomes a kernel thread (in the most common implementation). How is this kernel thread different from the original user thread?

(1) they have different text segments
(2) processor mode is different
(3) they use different stacks
(4) they access different part of the physical memory
(5) they use different set of processor registers

Answer (just give numbers): 2, 3

(Q5) (2 points) An object file (i.e., a **.o** file) is divided into sections that corresponds to memory segments. **Within each section**, what types of information are kept there?

(1) what symbols in that segment are exported
(2) owner and group information of the segment
(3) what symbols in that segment are undefined
(4) segment size
(5) initialization order for variables in the segment

Answer (just give numbers): 3, 4

(Q6) (2 points) If your CPU is currently **not** executing kernel code, which of the following **will not** cause an immediate transition into the kernel?

(1) user program makes a system call
(2) software interrupt
(3) hardware interrupt
(4) user program causes a divide-by-zero error
(5) signal delivery

Answer (just give numbers): 5

(Q7) (2 points) Which of the following statements are correct about the scheduler in **weenix**?

(1) when a kernel thread goes to sleep, it must sleep in **some** queue
(2) in **weenix**, the scheduler is responsible for handling cancellation and terminating a kernel thread
(3) **weenix** kernel uses scheduler functions such as **sched_wakeup_on()** and **sched_broadcast_on()** to wakup kernel threads waiting in a queue
(4) **weenix** scheduler is a simple sequential (e.g., first-in-first-out) scheduler
(5) none of the above is correct

Answer (just give numbers): 1 1,3,4!!!

This study resource was shared via CourseHero.com

(Q8) (2 points) Assuming that thread X calls **pthread_cond_wait()** to wait for a specified condition, what bad thing can happen if **pthread_cond_wait(cv,m)** is **not atomic** (i.e., pthread library did not implemented it correctly with respect to atomicity)? Please assume that everything else is done perfectly.

(1) thread X may never get unblocked
(2) thread X may execute critical section code without having the mutex locked
(3) thread X may miss a "wake up call" (i.e., another thread signaling the condition)
(4) thread X may cause a kernel panic
(5) none of the above is correct

Answer (just give numbers): 1,2,3 1,3!!

(Q9) (2 points) For an **open file**, what **context information** is stored in a **file object** in the kernel's system file table?

(1) file type
(2) reference count
(3) file ownership
(4) access mode
(5) file size

Answer (just give numbers): 2,4

(Q10) (2 points) Which of the following is part of an actual Unix **upcall**?

(1) the kernel invokes code that's part of a user program
(2) trap into the kernel
(3) a system call is invoked by a user process
(4) timer interrupt to switch from one user process to another
(5) keyboard interrupt to terminate a user process

Answer (just give numbers): 1,4 1!!

This study resource was shared via CourseHero.com

(Q11) (2 points) Which statements are correct about **copy-on-write** (as compared to not implementing copy-on-write in the kernel)?

(1) copy-on-write often increases physical memory usage
(2) without copy-on-write, **fork()** may take longer
(3) without copy-on-write, **execl()** may take longer
(4) copy-on-write occurs every time a user process writes to virtual memory
(5) none of the above is correct

Answer (just give numbers): 5

(Q12) (2 points) What are the operations that are **locked** together in an **atomic operation** by **sigwait(set)** (where **set** specifies a set of signals)?

(1) it waits for any signal specified in **set**
(2) it unblocked the signals specified in **set**
(3) if a signal specified in **set** is to be delivered, it calls the corresponding signal handler
(4) it clears all pending signal specified in **set**
(5) it blocked all the signals specified in **set**

Answer (just give numbers): 2 1?2!3?

(Q13) (2 points) Under what **general condition** would using mutex to access shared data by concurrent threads be an overkill because it's too restrictive and inefficient?

(1) when access pattern is random and the probability of accessing the same part of shared data simultaneously is small
(2) when the execution order of threads is mostly predictable
(3) when different threads would access different parts of shared data most of the time and only access the same parts of shared data occasionally
(4) when one thread is the parent thread of another thread
(5) when some threads just want to read the shared data

Answer (just give numbers): 5

This study resource was shared via CourseHero.com

(Q14) (2 points) Let's say that you have a uniprocessor and a user thread (thread X) is executing when a hardware interrupt occurs. Which of the following statements are true?

(1) in some OS, the interrupt handler will use the kernel stack of thread X
(2) in some OS, the interrupt becomes pending until thread X traps into the kernel, then the interrupt handler will be invoked using the kernel stack for thread X
(3) in some OS, the kernel would allocate a new kernel stack to be used by the interrupt handler
(4) in some OS, the interrupt handler will run in user mode and will use the user-space stack of thread X
(5) in some OS, a special kernel stack is shared by all interrupt handlers

Answer (just give numbers): 1

(Q15) (2 points) Let's say that you are executing a C function named **foo()** on an x86 processor and your thread is executing code somewhere in the middle of **foo()** (no interrupt is happening). Under what conditions (your answers will be logically AND'ed together) would the base/frame pointer (i.e., ebp) of the x86 processor **not** pointing at **foo**'s stack frame at this time?

(1) if **foo()** does not call another function
(2) if **foo()** is not a recursive function
(3) if **foo()** does not have local variables
(4) if **foo()** returns **void**
(5) if **foo()** does not have function arguments

Answer (just give numbers): 3,5

(Q16) (2 points) What was involved in POSIX's **solution** to provide **thread safety** for accessing the global variable **errno**?

(1) define **errno** to be a macro/function call that takes threadID as an argument
(2) generate a software interrupt when **errno** is accessed
(3) make accessing **errno** trap into the kernel
(4) use thread-specific **errno** stored in thread-specific storage inside thread control block
(5) generate a segmentation fault when **errno** is accessed

Answer (just give numbers): 1 1,4!!!

This study resource was shared via CourseHero.com