CSci 402 - Operating Systems

Common Midterm Exam

Spring 2020

(4:00pm - 4:40pm, Thursday, March 26)

Instructor: Bill Cheng

Teaching Assistant: (N/A)

(This exam is closed book, closed notes, closed everything.
No "cheat sheet" allowed.
No calculators, cell phones, or any electronic gadgets.)

Time: 40 minutes

_____-
Name (please print)

Total: 30 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. Please write and sign your name on this sheet *now*.
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 on the exam itself. If you are taking the exam *remotely* and if you need additional space, please ask for additional sheets and only write on one side since they need to be fax'ed.
5. Show all work (if applicable). If you cannot finish a problem, your written work will 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.
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.

(Q1) (2 points) Which of the following statements are correct?

(1) PIO devices can only read from main memory but cannot write to main memory
(2) DMA devices needs to be told what to do
(3) PIO devices are considered more "intelligent" than DMA devices
(4) DMA devices can perform writes to main memory
(5) PIO (Programmed I/O) devices can only work after the CPU downloads a "program" into it

Answer (just give numbers): 2,4

(Q2) (2 points) What is the difference between a **hard link** and a **soft/symbolic link** in Unix?

(1) a hard link can link to a file in another file system while a soft link cannot
(2) a hard link is an inode reference while a soft link is not
(3) a hard link can only be applied to files while a soft link is more general
(4) a hard link increases reference count in a **file object** while a soft link does not
(5) none of the above is correct

Answer (just give numbers): 3,4 2,3,4!

(Q3) (2 points) What is true about **interrupts** and **Unix signals**?

(1) a signal can be blocked/disabled while an interrupt cannot be blocked/disabled
(2) a user program can specify what interrupt service routine to use for a given interrupt
(3) an interrupt is delivered to a user program while a signal is delivered to the kernel
(4) an interrupt is delivered to the kernel while a signal is delivered to a user program
(5) a user program can specify what function to call when a signal is delivered

Answer (just give numbers): 4,5

(Q4) (2 points) In a multi-threaded process, if a **detached** thread calls **pthread_exit()**, it cannot delete its thread control block (TCB) and its stack. Which of the following statements are correct about when and/or where the TCB and the stack of this **detached** thread get cleaned up (i.e., deleted)?

(1) a reaper thread can be used to perform such a task
(2) such a task can only be performed only when the process dies
(3) such a task can only be performed inside a signal handler
(4) another thread in the same process can perform such a task when it's convenient
(5) only the kernel can perform such a task

Answer (just give numbers): 1,4

(Q5) (2 points) Let's say that a user thread (thread X) is executing when an unrelated hardware interrupt occurs. Let's further assume that the interrupt handler uses the most common implementation (especially regarding the way a stack is chosen to be used by the handler). The interrupt handler must execute till completion before thread X is resumed even though the interrupt has nothing to do with thread X. What bad thing can happen if we resume thread X before the interrupt handler finishes?

(1) thread X can cause a fault and corrupt the stack the interrupt handler is using
(2) thread X can cause a software interrupt and corrupt the stack the interrupt handler is using
(3) there is nothing thread X can to to mess up the interrupt handler
(4) thread X can make a system call and corrupt the stack the interrupt handler is using
(5) none of the above is correct

Answer (just give numbers): 1 1,2,4!!

(Q6) (2 points) Comparing cancellation in the **weenix** kernel and pthreads cancellation, which of the following statements are correct?

(1) since the **weenix** kernel is non-preemptive, it cannot have cancellation points
(2) just like pthreads, you can change **weenix** kernel cancellation "type" (asynchronous/deferred) programmatically
(3) **weenix** kernel cancellation "state" is always enabled and "type" is always deferred
(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): ? 3!!

(Q7) (2 points) Let kernel process C be the child process of kernel process P in **weenix**, which of the following statements are correct about **p_wait** (whose type is **ktqueue_t**) in the process control block of P? (Please note that since we are doing one thread per process in **weenix**, the words "process" and "thread" are considered interchangeable here.)

(1) if process C is runnable and process P calls **do_waitpid()**, process P will sleep on its own **p_wait** queue
(2) if process P is runnable and process C calls **do_waitpid()**, process C will sleep on P's **p_wait** queue
(3) when process C dies, it will add itself to process P's **p_wait** queue
(4) when process P dies, it will add itself to process C's **p_wait** queue
(5) none of the above is correct

Answer (just give numbers): 1

(Q8) (2 points) Let say that X, Y, Z are Unix user processes and X's parent is Y and Y's parent is Z. When process Y dies unexpectedly, what happens to process X?

(1) process Z becomes process X's new parent
(2) the idle process becomes process X's new parent
(3) the OS kernel will terminate process X
(4) the init process becomes process X's new parent
(5) process X becomes parentless

Answer (just give numbers): 4

(Q9) (2 points) Which statements are correct about Unix signals?

(1) when a signal is generated, if it's blocked, it is lost
(2) a signal handler cannot return a value
(3) when a signal is generated, if it's blocked, it becomes pending
(4) some signals can be ignored by an application
(5) some signals cannot be blocked

Answer (just give numbers): 3

(Q10) (2 points) What is involved in POSIX's solution to provide **thread safety** for accessing the system-call level global variable **errno** that was **not** used before multithreading was invented?

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

Answer (just give numbers): 2

(Q11) (2 points) If your main thread wants to **wait** for all the other threads in the process to die before the main thread dies, what should the main thread do to accomplish this objective **properly**?

(1) call **pthread_kill()** on each of these threads
(2) call **pthread_detach()** on each of these threads so it doesn't have to wait
(3) call **pthread_cancel()** on each of these threads
(4) call **pthread_join()** to join with each of these threads
(5) call **pthread_exit()** and let the pthread library to take care of this automatically

Answer (just give numbers): 4

(Q12) (2 points) In 3 words or less **each**, in addition to **"saved registers"**, what are the **other four** categories of things that would normally go into an **x86 stack frame**?

local variables,ebp,eip,args

(Q13) (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 use different set of processor registers
(2) they use different stacks
(3) they access different part of the physical memory
(4) they have different text segments
(5) processor mode is different

Answer (just give numbers): 2,5

(Q14) (2 points) Which of the following statements are correct about the scheduler in **weenix**?

(1) in **weenix**, the scheduler is responsible for handling cancellation and terminating a kernel thread
(2) when a kernel thread goes to sleep, it must wait in **some** queue
(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 first-come-first-served scheduler
(5) none of the above is correct

Answer (just give numbers): 2 2,3,4!!

(Q15) (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 identifies a device driver
(2) minor device number is rarely used
(3) minor device number must be smaller than the major device number
(4) minor device number indicated which process is currently using the device
(5) none of the above is correct

Answer (just give numbers): 5