CSci 402 - Operating Systems
Midterm Exam (Alternate Exam)
Spring 2021

(7:00:00pm - 7:40:00pm, Wednesday, March 24)

Instructor: Bill Cheng

Teaching Assistant: Shashank Saurabh

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

Time: 40 minutes

—————————————-
Name (please print)

Total: 36 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.

(Q1) (2 points) How would you implement the following **guarded command** using a POSIX mutex **m**, a POSIX condition variable **c**, and calls to pthread functions (you should know what these are)? Please assume that all the variables have been properly initialized.

```
when (guard) [
  my_atomic_func();
]
```

(Q2) (2 points) Assuming regular Unix/Linux calling convention when one C function calls another, which of the following statements are true about an **x86 stack frame**?

(1) content of the EAX register is never saved in a stack frame
(2) content of the ESP register is often saved in the callee's stack frame
(3) content of the EIP register is never saved in a the stack frame
(4) the EBP register points to something that looks like a linked-list in the stack
(5) the values of all CPU register are saved in the callee's stack frame

Answer (just give numbers): ______________________________

(Q3) (2 points) Which of the following statements are correct about **weenix** kernel mutexes?

(1) since the **weenix** kernel is non-preemptive, there is no need for kernel mutexes
(2) there is only one thread running in the **weenix** kernel, so there is no need for kernel mutexes
(3) **weenix** uses mutexes to ensure that only one kernel thread is accessing a particular device at a time
(4) since the **weenix** is meant to run on a uniprocessor, there is no need for kernel mutexes
(5) **weenix** uses mutexes whenever a kernel thread needs to update a kernel linked list data structure

Answer (just give numbers): ______________________________

(Q4) (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 blocked all the signals specified in **set**
(2) it waits for any signal specified in **set**
(3) it unblocked the signals specified in **set**
(4) if a signal specified in **set** is to be delivered, it calls the corresponding signal handler
(5) it clears all pending signal specified in **set**

Answer (just give numbers): ____________________

(Q5) (2 points) If your main thread wants to **wait** for all the other threads in the process to die before the main thread itself dies, what must the main thread do to accomplish this objective **properly**?

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

Answer (just give numbers): ____________________

(Q6) (2 points) What was involved in POSIX's **solution** to provide **thread safety** for accessing the global variable **errno**?

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

Answer (just give numbers): ____________________

(Q7) (2 points) Which of the following are commonly found address space segments in a Unix address space?

(1) heap segment
(2) object segment
(3) thread segment
(4) stack segment
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q8) (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 P is runnable and process C calls **do_waitpid()**, process C will sleep on P's **p_wait** queue
(2) when process C dies, it will add itself to process P's **p_wait** queue
(3) if process C is runnable and process P calls **do_waitpid()**, process P will sleep on its own **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 a correct answer

Answer (just give numbers): ____________________

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

(1) some signals can be ignored by an application
(2) when a signal is generated, if it's blocked, it becomes pending
(3) when a signal is generated, if it's blocked, it is lost
(4) a signal handler cannot return a value
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q10) (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 have function arguments
(2) if **foo()** is not a recursive function
(3) if **foo()** does not call another function
(4) if **foo()** returns **void**
(5) if **foo()** does not have local variables

Answer (just give numbers): ____________________

(Q11) (2 points) Which of the following statements are correct?

(1) a signal is generated by the user process and can be delivered to another user process without using a system call
(2) a signal is generated by the kernel and delivered to a user process
(3) a signal is a software interrupt
(4) hardware interrupts are generated by the hardware and delivered to the kernel
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q12) (2 points) If **sigwait(set)** is implemented correctly in every way **except** that it is **not an atomic operation**, what bad things can happen (assuming that everything else is done correctly)?

(1) the thread calling **sigwait()** may get terminated while other threads are not affected
(2) **sigwait()** may never return
(3) a signal specified in **set** may become pending forever and never delivered
(4) a signal specified in **set** may be lost
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q13) (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;
}
```

If thread X calls **switch()** to switch to thread Y, which of the following statements are correct?

(1) thread X saves its stack pointer in thread X's stack
(2) thread X saves its stack pointer in thread X's thread control block
(3) thread Y restores its frame pointer from thread X's stack
(4) thread X saves its frame pointer in thread X's stack
(5) thread X saves its stack pointer in thread Y's thread control block

Answer (just give numbers): ______________________________

(Q14) (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 miss a "wake up call" (i.e., another thread signaling the condition)
(2) thread X may never get unblocked
(3) thread X may cause a kernel panic
(4) thread X may execute critical section code without having the mutex locked
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q15) (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) **weenix** kernel cancellation "state" is always enabled and "type" is always deferred
(3) just like pthreads, you can change **weenix** kernel cancellation "type" (asynchronous/deferred) programmatically
(4) just like pthreads, you can change **weenix** kernel cancellation "state" (enabled/disable) programmatically
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q16) (2 points) In a multi-threaded user process, if a thread (that's **not detached**) calls **pthread_exit()**, it goes into the zombie state. When can the **user space stack of this thread** get deleted?

(1) when another thread traps into the kernel
(2) when the process goes into the zombie state
(3) when another thread signals that it's okay for this thread to delete its stack
(4) when another thread joins with this thread
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q17) (2 points) Which statements are correct about **copy-on-write**?

(1) never writes to a shared and writable memory page, only writes to a copy of that page
(2) a private page can never be written to
(3) a shared page can never be written to more than once
(4) every time a thread writes to a private page, the page gets copied and the write goes into the copy
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q18) (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) the OS kernel will terminate process X
(2) process X becomes parentless
(3) process Z becomes process X's new parent
(4) the idle process becomes process X's new parent
(5) the init process becomes process X's new parent

Answer (just give numbers): ____________________