CSci 402 - Operating Systems
Midterm Exam (DEN Section)
Spring 2022

[9:30:00am-10:10:00am), Wednesday, March 23

Instructor: Bill Cheng

Teaching Assistant: Zhuojin Li

(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) Let say that X, Y, Z are regular Unix user processes and X's parent is Y and Y's parent is Z. When process Y dies, what would happen?

(1) process Z becomes process X's new parent
(2) the OS kernel will terminate process X because its parent is now dead
(3) process Z becomes parentless
(4) even if process X is dead already, process X will become parentless
(5) the init process becomes process X's new parent

Answer (just give numbers): _____

(Q2) (2 points) What does **async-signal safety** mean?

(1) do as little as possible in a signal handler
(2) never use a global variable in a signal handler
(3) one should always use synchronous signals
(4) make sure that your code that handles an asynchronous signal cannot corrupt data structures shared with the rest of your program
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q3) (2 points) In the code below, **setitimer()** is used to generate SIGALRM when the alarm goes off, **pause()** is used to wait for SIGALRM to occur, and the SIGALRM handler is simple and does not do anything that can cause any problem.

```
struct itimerval timerval;
... /* setup timerval to timeout in 10ms */
sigset(SIGALRM, DoSomethingInteresting);
setitimer(ITIMER_REAL, &timerval, 0);
pause();
```

Assuming the code has no compile-time error, under what condition would the above code **freeze**?

(1) SIGALRM is generated immediately after **pause()** is called
(2) the **DoSomethingInteresting** function does not exist
(3) the thread calling **pause()** self-terminates
(4) SIGALRM is delivered after **setitimer()** is called and before **pause()** is called
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q4) (2 points) Which of the following are commonly found address space memory segments in a Unix address space?

(1) stack segment
(2) object segment
(3) thread segment
(4) upcall segment
(5) dynamic/heap segment

Answer (just give numbers): _____

(Q5) (2 points) What was involved in POSIX's **solution** to provide **thread safety** for accessing the global variable **errno** when a system call returns?

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

Answer (just give numbers): _____

(Q6) (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) **sigwait()** may never return
(2) the thread calling **sigwait()** may get terminated while other threads are not affected
(3) the thread calling **sigwait()** may become uncancellable
(4) a signal specified in **set** may be lost
(5) a signal specified in **set** may become pending forever and never get delivered

Answer (just give numbers): _____

(Q7) (2 points) Let's say that your **umask** is set to **0104**. (1) What file permissions will you get (in octal) if use an editor to create the **warmup1.c** file? (2) What file permissions will you get (in octal) if use a compiler to create the **warmup1** executable?

(Q8) (2 points) Which of the following statements are correct about kernel mutexes in **weenix**?

(1) since the **weenix** is meant to run on a uniprocessor, there is no need for kernel mutexes
(2) there is only one thread running in the **weenix** kernel, so there is really no need for kernel mutexes
(3) **weenix** uses mutexes whenever a kernel thread needs to update a kernel linked list data structure
(4) since the **weenix** kernel is preemptive, kernel mutexes must be implemented
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q9) (2 points) If **foobar** is a valid program name, you type **foobar** in a command shell and wait for **foobar** to finish executing, what **system calls** must be the **command shell** make before **foobar** finished running?

(1) **fork()**
(2) one of the **exec** system calls
(3) **exit()**
(4) **kill()**
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q10) (2 points) In a multi-threaded process, if a **detached** POSIX thread calls **pthread_exit()**, it cannot delete its user-space stack. Which of the following statements are correct about when and/or where the stack of this **detached** thread get freed up (i.e., destroyed)?

(1) such a task can only be performed inside a signal handler
(2) such a task can only be performed only when a thread in the process calls **exit()**
(3) only the kernel can perform such a task
(4) another user thread in the same process can perform such a task when it's convenient
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q11) (2 points) Let **cv** be a POSIX condition variable and **m** be a POSIX mutex. When thread X calls **pthread_cond_wait(cv,m)**, several operations are **locked** together within one **atomic operation**. Which of the following operations are included within this atomic operation?

(1) lock the mutex
(2) signal the condition
(3) make thread X goes to sleep on the cv queue
(4) broadcast the condition
(5) unlock the mutex

Answer (just give numbers): _____

(Q12) (2 points) The code below is a solution to the **readers-writers problem**.

```
reader( ) {
  when (writers == 0) [
    readers++;
  ]
  /* read */
  [readers--;]
}
```

```
writer( ) {
  when ((writers == 0) &&
      (readers == 0)) [
    writers++;
  ]
  /* write */
  [writers--;]
}
```

It has a major problem. What can be said about this major problem?

(1) a reader thread may block forever if writer threads keep arriving and there are always writer threads present
(2) some threads may **starve**, i.e., never get to run even when there are not deadlocks
(3) two reader threads may block each other out for a long time
(4) a writer thread may never get to write even if there are no reader thread present
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q13) (2 points) Why is it that when a user thread performs a system call, the corresponding kernel thread cannot use the user-space stack of the user thread and the user thread's stack pointer?

(1) the user thread stack frames may be upside-down
(2) the user thread's stack pointer may point to a bad memory location
(3) the user thread may be dead already
(4) the user thread code may not be using a compatible stack register
(5) the user thread stack space may have very little room left (i.e., it is running out of stack space)

Answer (just give numbers): _____

(Q14) (2 points) When static linking is used, which of the following are the responsibilities of a **linker**?

(1) maintain reference count for file system hard links
(2) determines final addresses for global variables that need to be relocated
(3) figure out what to do with unresolved symbols
(4) compile source code into relocatable object files
(5) allocate and manage the heap space

Answer (just give numbers): _____

(Q15) (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_detach()** on each of these threads so it doesn't have to wait
(2) call **pthread_exit()** and let the pthread library to take care of this automatically
(3) call **pthread_kill()** on each of these threads
(4) call **pthread_cancel()** on each of these threads
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q16) (2 points) Let's say that you have an infinitely fast and accurate computer and you run your warmup2 on it with the following commandline: "`./warmup2 -r 1000 -t g0.txt`" and the content of `g0.txt` is as follows:

```
4
2    2    7
4    1    4
1    3    2
1    2    3
```

How many **milliseconds** into the simulation will packet `p3` (i.e., the 3rd packet) leaves the system? Please just give an integer value answer (no partial credit for this problem).

(Q17) (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) this function does not change the content of the ebp register
(2) this function is unusually short
(3) unlike most other functions, the thread that calls this function can fall asleep inside this function and wake up at a later time
(4) this function references global variables
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q18) (2 points) Which of the following operations are **locked** together in **one atomic operation** by **sigsuspend(set)** (where **set** specifies a set of signals)?

(1) it calls all signal handlers specified in **set**
(2) it waits for a signal specified in **set** to get delivered
(3) it suspends all other threads in the same process
(4) it unblocks the signals specified in **set**
(5) it blocks all signals

Answer (just give numbers): _____