CSci 402 - Operating Systems

Midterm Exam (TT Section)

Spring 2022

[9:30:00am-10:10:00am), Thursday, March 24

Instructor: Bill Cheng

Teaching Assistant: Rohan Madhani

(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's say that your thread is executing C code somewhere in the middle of a C function named **foobar()** on an x86 processor. If you know that the base/frame pointer (i.e., ebp) of the x86 processor is **not** pointing anywhere within **foobar**'s stack frame, what conclusion can you make?

(1) **foobar()** does not have local variables
(2) the function return type of **foobar()** is **void**
(3) **foobar()** is a recursive function
(4) **foobar()** does not make function calls
(5) **foobar()** does not have function arguments

Answer (just give numbers): ____________________

(Q2) (2 points) Under what **general condition** would using only mutex to access shared data by concurrent threads an overkill because it's too restrictive and inefficient?

(1) when one thread is the parent thread of another thread
(2) when threads that access shared data rarely make system calls
(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 the execution order of threads is mostly predictable
(5) when some threads only want to write to the shared data and promise not to read the shared data

Answer (just give numbers): ____________________

(Q3) (2 points) In Unix, a directory is a file. Therefore, I will use the term "directory file" to refer to the **content** of a directory. Which of the following statements are correct about a Unix directory?

(1) a directory file contains a mapping from file names to disk addresses
(2) if directory Z is a subdirectory of directory X, the inode number of directory X is stored in the directory file for Z
(3) if file Y is in directory X, the file size of Y is stored in the directory file for X
(4) if directory Z is a subdirectory of directory X, the inode number of directory Z is stored in the directory file for X
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q4) (2 points) Which of the following statements are correct about a Unix **pipe**?

(1) a pipe object lives in the user space
(2) a pipe cannot be used by one user thread to send characters to the thread itself
(3) a pipe has two ends, a user program decides which end is for reading and which end is for writing
(4) a pipe can be used by one user thread to send characters to another user thread in the same process
(5) a pipe is implemented as a file object inside the kernel

Answer (just give numbers): ______________________________

(Q5) (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  4  9
4  2  8
4  1  3
1  3  4
```

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).

(Q6) (2 points) For an **open file** in Unix, what **context information** is stored in a **file object** in the kernel's system file table?

(1) user ID and group ID of the open file
(2) access mode to remember how the file was opened
(3) file cursor position
(4) file type of the open file
(5) the size of the file (number of bytes)

Answer (just give numbers): ______________________________

(Q7) (2 points) Assuming that you only have one processor, which of the following statements are correct about **interrupts**, **traps**, and **Unix signals**?

(1) an interrupt is most likely caused by the currently running program
(2) a trap can only be caused by the currently running program
(3) for all signal types, a user program can catch that signal by specifying a user space signal handler
(4) a process in the zombie state cannot cause a trap
(5) a signal can be blocked/disabled while an interrupt cannot be blocked/disabled

Answer (just give numbers): ____________________

(Q8) (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) the EBP register points to something that looks like a doubly-linked-list in the stack
(2) the values of all CPU register are automatically saved in the caller's stack frame
(3) content of the ESP register is often saved in the callee's stack frame
(4) content of the EIP register is never saved in a the stack frame
(5) content of the EAX register is never saved in a stack frame

Answer (just give numbers): ____________________

(Q9) (2 points) After a process has entered the **zombie** state, how would it exit the **zombie** state?

(1) when the INIT process has entered its zombie state
(2) when the kernel perform garbage collection (i.e., automatic freeing of useless dynamically allocatd objects)
(3) when its parent process enters a cancellation point
(4) when all its child processes have exited their zombie states
(5) none of the above is a correct answer

Answer (just give numbers): ____________________

(Q10) (2 points) What's the reason why it is not possible for a thread (with its code written in C) to **completely delete itself**?

(1) a thread cannot switch to itself
(2) a thread cannot be in user space and in kernel space simultaneously
(3) a thread cannot have two stacks
(4) a thread cannot be running inside two functions simultaneously
(5) none of the above is a correct answer

Answer (just give numbers): ______________________

(Q11) (2 points) Let's say that you use a thread to catch <Cntrl+C> using the code below (please assume that SIGINT is blocked everywhere else in the process, all the variables have been properly initialized, and you have implemented everything else correctly):

```
void *monitor() {
  int sig;
  while (1) {
    sigwait(&set, &sig);
    pthread_mutex_lock(&m);
    display(&state);
    pthread_mutex_unlock(&m);
  }
  return(0);
}
```

If the user pressed <Cntrl+C> and **sigwait()** returns. What would happen if the user pressed another <Cntrl+C> before **sigwait()** is called again in the next iteration of the **while** loop?

(1) the 2nd <Cntrl+C> becomes pending
(2) the 2nd <Cntrl+C> will cause the default SIGINT handler to get executed
(3) the 2nd <Cntrl+C> is lost
(4) SIGKILL will be delivered to your program and your program will be killed
(5) none of the above is a correct answer

Answer (just give numbers): ______________________

(Q12) (2 points) In a multi-threaded process, how can a user thread **self terminate** without killing the process it's in?

(1) returns from its first procedure
(2) calls **exit()**
(3) calls **pthread_join()** to join with itself
(4) calls **kill()** to itself a SIGTERM signal
(5) calls **pthread_exit()**

Answer (just give numbers): ______________________

(Q13) (2 points) On Unix/Linux systems, **sequential I/O** on regular files is done by calling **read/write()** system calls. Which of the following statements are true about Unix/Linux **block I/O** on regular files?

(1) block I/O is available to both kernel and user processes
(2) to perform block I/O, you can map a file (or part of a file) into your address space
(3) block I/O is performed by first calling lseek() and then read/write()
(4) using read/write() system calls can also be considered as doing block I/O
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q14) (2 points) Which statements are correct about **Unix signals**?

(1) by convention, a signal handler should return a value of zero if the signal was handled successfully and return a non-zero value otherwise
(2) when a signal is generated, if it's blocked, it can still be delivered by creating a new thread
(3) if a SIGINT handler is invoked to deliver SIGINT, SIGINT is blocked inside that SIGINT handler
(4) an application can ask the OS to not deliver certain signals
(5) when a signal is generated, if it's blocked, it is lost

Answer (just give numbers): ______________________________

(Q15) (2 points) Which of the following is part of the action that must be taken during a Unix **upcall**?

(1) keyboard interrupt to terminate a user process
(2) the kernel invokes user space code
(3) context switching from privileged mode to user mode
(4) user process makes a system call to figure out the reason for the upcall
(5) none of the above is a correct answer

Answer (just give numbers): ______________________________

(Q16) (2 points) Let's say that your **umask** is set to **0625**. (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?

(Q17) (2 points) Why must a process enter a **zombie** state immediately after it terminates?

(1) because operating system can run faster this way
(2) because its process ID needs to be recycled immediately
(3) because its children processes may not have died and they may need information from the parent's process control block
(4) because its parent process has not retrieve this process' return/exit code
(5) none of the above is a correct answer

Answer (just give numbers): ________________________________

(Q18) (2 points) In the most common OS implementation, when a user thread makes a system call, it simply becomes a kernel thread. In such an implementation, how is this kernel thread different from the original user thread?

(1) CPU mode is different
(2) they access different part of the physical memory
(3) their text segments have different virtual addresses
(4) the CPU stack pointer registers they use are different
(5) the CPU registers they use are completely different

Answer (just give numbers): ________________________________