CSci 402 - Operating Systems

Midterm Exam

Summer 2021

(10:00:00am - 10:40:00am, Tuesday, July 6)

Instructor: Bill Cheng

Teaching Assistant: (N/A)

(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) Assuming that the **...** in the code below represents meaningful code and the program compiles perfectly.

```
int main(int argc, char *argv[])
{
  int i;
  for (i = 0; i < 10; i++)
    pthread_create(...);
  return 0;
}
```

What is the main issue/problem (select **only one**) with the above code?

(1) **exit()** may get called before any child thread get a chance to run
(2) main thread may run in parallel with the child threads
(3) cannot control the execution order of the main thread and the child threads
(4) deadlock would occur
(5) all child threads may finish before the main thread returns

Answer (just give numbers): _____

(Q2) (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) **open()**
(2) **wait()**
(3) **exit()**
(4) **fork()**
(5) **one of the exec system calls**

Answer (just give numbers): _____

(Q3) (2 points) Let's say that your **umask** is set to **0374**. (1) What file permissions will you get (in octal) if use a compiler to create the **warmup1** executable? (2) What file permissions will you get (in octal) if use an editor to create the **warmup1.c** file?

(Q4) (2 points) An object file (i.e., a **.o** file) is divided into sections that correspond to memory segments. **Within each section**, what types of information are kept there?

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

Answer (just give numbers): _____

(Q5) (2 points) Let's say that you have a uniprocessor and the kernel is currently servicing an interrupt when a **higher priority interrupt** occurs. What would typically happen?

(1) nothing, because when an interrupt handler is running, all other interrupts are blocked/disabled
(2) the higher priority interrupt will use the same kernel stack as the current interrupt handler to execute its interrupt service routine
(3) the higher priority interrupt will execute with a newly allocated kernel stack
(4) higher priority interrupt will be blocked until the current interrupt handler yields the processor voluntarily
(5) this is a violation and will cause a kernel panic

Answer (just give numbers): _____

(Q6) (2 points) Under what **general condition** would using mutex to access shared data by concurrent threads an overkill because it's too restrictive and inefficient?

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

Answer (just give numbers): _____

(Q7) (2 points) On Unix/Linux systems, **sequential I/O** on files is done by calling **read/write()** system calls. Which of the following statements are true about Unix/Linux **block I/O** on files?

(1) you can perform block I/O by calling read/write() system calls with special function arguments
(2) to perform block I/O, you must first map a file into your address space
(3) using read/write() system calls can also be considered as doing block I/O
(4) block I/O is done by first calling lseek() and then read/write()
(5) block I/O is only available to kernel processes and not available to user processes

Answer (just give numbers): _____

(Q8) (2 points) If your CPU is currently executing user space code, which of the following will **not** cause an immediate transition into the kernel?

(1) hardware interrupt
(2) signal delivery
(3) user program causes a divide-by-zero error
(4) user program calls **free()**
(5) user program executes a software interrupt machine instruction

Answer (just give numbers): _____

(Q9) (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 execute critical section code when it does not have the mutex locked
(3) thread X may sleep forever and never get unblocked
(4) thread X may cause a kernel panic when it calls pthread_cond_wait()
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q10) (2 points) Which of the following statements are correct about the scheduler in **weenix**?

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

Answer (just give numbers): _____

(Q11) (2 points) The **file descriptor table** for a process is maintained inside the kernel. What security problems could occur if such a table and **open file context** information is maintained in **user space**?

(1) a user program will be able to execute a specific file to which it has only read access
(2) a user program will be able to write to a specific file to which it has only read access
(3) a user program will be able to execute a specific file to which it has read+write access
(4) a user program will be able to read a specific file to which it does not have access
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q12) (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 1 -t g0.txt`" and the content of `g0.txt` is as follows:

```
4
2000   2    9000
7000   1   15000
1000   3    2000
1000   2    4000
```

How many seconds 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).

(Q13) (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> will cause the default SIGINT handler to execute
(2) the 2nd <Cntrl+C> is lost
(3) SIGINT will be delivered the next time **sigwait()** is called
(4) your program will crash
(5) the 2nd <Cntrl+C> becomes pending

Answer (just give numbers): _____

(Q14) (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) if file Y is in directory X, the owner ID and group ID of file Y is stored in the directory file for X
(2) a directory file contains a mapping from file names to disk addresses
(3) if file Y is in directory X, the inode number of file Y is stored in the directory file for X
(4) a directory file contains a mapping from strings to integers
(5) if file Y is in directory X, the file size of Y is stored in the directory file for X

Answer (just give numbers): _____

(Q15) (2 points) Which statements are correct about Unix signals?

(1) when a signal is generated, if it's blocked, it becomes pending
(2) when a signal is generated, if it's blocked, it is destroyed
(3) by convention, a signal handler should return a value of zero if the signal was handled successfully and return a non-zero value otherwise
(4) some signals can be ignored by an application
(5) none of the above is a correct answer

Answer (just give numbers): _____

(Q16) (2 points) What's the reason why it is not possible for a thread (with its code written in C) to **completely kill itself**?

(1) a thread cannot delete its own stack
(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) a thread cannot switch to itself

Answer (just give numbers): _____

(Q17) (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) they use different set of processor registers
(2) they use different stack pointer registers in the CPU
(3) they have different text segments
(4) processor mode is different
(5) they access different part of the physical memory

Answer (just give numbers): _____

(Q18) (2 points) After a process has entered the **zombie** state, how would it exit the **zombie** state?

(1) when its parent process calls **exit()**
(2) when all it's child processes have died
(3) when the kernel perform garbage collection
(4) when the wall clock advanced to the next day
(5) none of the above is a correct answer

Answer (just give numbers): _____