UCLA Computer Science 111 (fall 2019) midterm
100 minutes total, open book, open notes,
no computer or any other automatic device

Name:_ _____ Student ID _____

1	2	3	4	5	6	7	total

1. Saltzer & Kaashoek give three methods for enforcing hard modularity: client/service, hardware virtualization, and software virtualization.

1a (6 minutes). Give an example of each approach as used on SEASnet.

1b (6 minutes). Briefly discuss the pros and cons of the three approaches.

2. POSIX specifies a function 'pause' that suspends execution of the current thread until a signal arrives. When a signal is delivered, this either terminates the process or invokes the corresponding signal handler (depending on how signals are normally handled); if the latter, if the signal handler returns then 'pause' returns -1 and sets errno to EINTR.

2a (10 minutes). Suppose you're running on a stripped-down kernel that implements only the following system calls: alarm, chdir, close, dup2, execve, _exit, fork, getpid, kill, lseek, open, pipe, read, rename, signal, unlink, waitpid, and write. Write a C implementation of 'pause' as best you can, using just these system calls along with ordinary user-mode code. Assume that each system call fails and sets errno to EINTR if interrupted.

2b (5 minutes). Explain any shortcomings in your library implementation, compared to doing 'pause' via a true system call (which is what GNU/Linux does).

3 (8 minutes). Loading (and chain-loading) is common during booting; for example, traditionally the CPU runs the BIOS, which loads the MBR, which loads the VBR, etc. Does it make sense for shutdown to reverse this process? That is, is it reasonable for an operating system to do *storing* (or chain-storing) during shutdown? If so, explain what would motivate storing and/or chain-storing and how it should be implemented; if not, explain why not.

4. In fair round-robin (FRR), a newly arrived job is placed at the end of the queue of jobs waiting to run. In unfair round-robin (URR), a newly arrived job is placed at the beginning of the queue.

4a (6 minutes). For typical job mixes on SEASnet, which should be better for average wait time: FRR or URR? Which should be better for average turnaround time? Briefly explain your reasoning. Assume that SEASnet uses a 10 ms quantum with a 3 µs context switch time and that it is running typical student programs.

(This problem is continued on the next page.)

4b (12 minutes). Suppose the quantum is 10 ms and context switches consume 3 µs, so that timer interrupts occur every 10.003 ms and each job gets 10 ms of CPU time per quantum. And suppose also that the job mix is as follows:

A	0	30
B	5	20
C	15	40
D	25	10

where the columns are job ID, arrival time, and run time, respectively, where all times are in milliseconds. Calculate the average wait time and the average turnaround time for this job mix assuming FRR. Similarly, calculate the two averages assuming URR.

4c (8 minutes). Did your answer to (b) confirm the hypothesis in your answer to (a)? If so, give a job mix that disconfirms your hypothesis; if not, give a job mix that confirms your hypothesis. Use the simplest job mix that you can. Or, if it's not possible to give such a job mix, explain why not.

5. Consider the following implementation of read_sector discussed in class.

```
void read_sector (int s, char *a)
{
  /* 1*/ while ((inb (0x1f7) & 0xc0) != 0x40)
  /* 2*/   continue;
  /* 3*/ outb (0x1f2, 1);
  /* 4*/ outb (0x1f3, s & 0xff);
  /* 5*/ outb (0x1f4, (s>>8) & 0xff);
  /* 6*/ outb (0x1f5, (s>>16) & 0xff);
  /* 7*/ outb (0x1f6, (s>>24) & 0xff);
  /* 8*/ outb (0x1f7, 0x20);
  /* 9*/ while ((inb (0x1f7) & 0xc0) != 0x40)
  /*10*/   continue;
  /*11*/ insl (0x1f0, a, 128);
}
```

What, if anything, would go wrong if we did the following? Briefly explain.

5a (3 minutes). In /*1*/, change '!=' to '=='.

5b (3 minutes). In /*3*/, change the 2nd outb argument from 1 to 2.

5c (3 minutes). Put a copy of /*8*/ before /*1*/.

5d (3 minutes). Interchange /*7*/ and /*3*/.

5e (3 minutes). Insert the statement 's++;' between /*6*/ and /*7*/.

6 (12 minutes). In double buffering, the computer arranges to read input block N+1 while processing block N so that input and CPU processing can be done in parallel. Design an API involving a new function read_sector2 that supports double buffering. Keep the API as simple and as close to read_sector as you can. You may need to add more functions to your API. Implement your API in the same style as read_sector, and give sample code that invokes your API to get double buffering. To save time in writing your answer, you can use abbreviations like '/*1-5*/' to stand for lines 1 through 5 of

7. Consider the following program AD-5.2-variant.c, derived from Arpaci-Dusseaus' Figure 5.2:

```
 1  #include <stdio.h>
 2  #include <unistd.h>
 3  int main (void) {
 4    int pid = getpid ();
 5    printf ("hello world (pid:%d)\n", pid);
 6    int rc = fork ();
 7    if (rc < 0) {
 8      fprintf (stderr, "fork failed\n");
 9      return 1;
10    }
11    pid = getpid ();
12    if (rc == 0) {
13      printf ("hello, I am child (pid:%d)\n", pid);
14    } else {
15      printf ("hello, I am parent of %d (pid:%d)\n", rc, pid);
16    }
17    return 0;
18  }
```

Consider the following behavior that I got when I compiled and ran this program on lnxsrv07.seas.ucla.edu:

```
 1  $ gcc -O2 -Wall AD-5.2-variant.c
 2  $ ./a.out # First program run
 3  hello world (pid:20600)
 4  hello, I am parent of 20601 (pid:20600)
 5  hello, I am child (pid:20601)
 6  $ ./a.out | cat # Second program run
 7  hello world (pid:20605)
 8  hello, I am parent of 20607 (pid:20605)
 9  hello world (pid:20605)
10  hello, I am child (pid:20607)
```

7a (4 minutes). What race could cause the output to look substantially different from either the first or the second run, and what would this output look like?

(This problem is continued on the next page.)

7b (8 minutes). Explain each difference in the outputs of the two program runs.