UCLA Computer Science 111 (Winter 2017) Midterm
100 minutes total
Open book, open notes, closed computer

Name:_____ Student ID:_____

1	2	3	4	5	6	7	8

1 (3 minutes). Does Ubuntu use soft or hard modularity? Briefly explain.

2 (5 minutes). Suppose you run the following command, where 'lab0' implements Project 0.

```
echo four | \
  lab0 --output=score --output=and \
       --output=7 --output=years --output=ago
```

What behavior should you observe and why?

3 (7 minutes). Suppose the x86-based kernel Xunil is like the Linux kernel but reverses the usual pattern for system calls: in Xunil, an application issues a system call by executing an RETI (RETurn from Interrupt) instruction rather than by executing an INT (INTerrupt) instruction. Other than this difference in instruction choice, Xunil is supposed to act like Linux.

Is the Xunil idea completely crazy, or is it a valid (albeit unusual) operating system interface? Briefly explain.

4a (9 minutes). Translate the following shell script to simpsh as well as possible. Your translation should simply invoke simpsh with appropriate arguments.

```
#! /bin/sh
(head -n 20 2>a <b | sort 2>>c | tail) >d
cat <d | cat >>d
```

4b (4 minutes). How and why will your translation differ in behavior from the original?

4c (5 minutes). Give a scenario whereby the above shell script, or its simpsh near-equivalent, will loop indefinitely.

4d (5 minutes). Propose minimal upward-compatible changes to simpsh that will allow you to translate the above script to simpsh faithfully, so that its behavior is 100% compatible with the standard shell.

4e (5 minutes). Give a scenario involving a single invocation of simpsh that can first crash simpsh and cause it to dump core, and then output the message "Fooled ya!" to standard output.

5. Round Two Robin (T2R) scheduling is a preemptive scheduling algorithm, like Round Robin (RR) scheduling, but it differs in that when a quantum expires and two or more processes are in the system, then T2R does not always move the currently-running process to the end of the run queue; instead, with probability 0.5, T2R lets the currently-running process continue to run for another quantum, so that other processes continue to wait in the queue.

5a (6 minutes). Compare RR to T2R scheduling with respect to utilization and average wait time; give an example.

5b (5 minutes). Is starvation possible with T2R scheduling? Briefly explain.

6. Suppose you compile and run the following C program in a terminal session that operates on a SEASnet GNU/Linux server:

```
 1  #include <signal.h>
 2  #include <unistd.h>
 3  #include <stdio.h>
 4  static unsigned char n;
 5  void handle_sig (int sig) {
 6    printf ("Got signal! n=%d\n", n++);
 7  }
 8  int main (void) {
 9    signal (SIGINT, handle_sig);
10    do {
11      printf ("looping n=%d\n", n++);
12      signal (SIGINT, handle_sig);
13    } while (n != 0);
14    return 0;
15  }
```

Give race-condition scenarios by which this program could possibly do the following:

6a (3 minutes). Output more than 256 lines.

6b (5 minutes). Output successive lines containing "n=N" and "n=N" strings where N is the same integer in both lines.

6c (3 minutes). Output a line containing two "=" signs.

6d (5 minutes). Dump core.

6e (5 minutes): Which lines or lines of the program can you remove without changing the program's set of possible behaviors? Briefly explain.

7. Consider the following implementation of read_sector:

```
void wait_for_ready (void) {
  while ((inb (0x1f7) & 0xC0) != 0x40)
    continue;
}

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

What, if anything, would go wrong if we did the following? Briefly explain. Treat each proposed change independently of the other changes.

7a (3 minutes). Remove /*8*/.
7b (3 minutes). In /*3*/, change 0xff to 0xfff.
7c (3 minutes). Interchange /*3*/ and /*4*/.
7d (3 minutes). Interchange /*6*/ and /*7*/.
7e (3 minutes). Put a copy of /*1*/ after /*9*/.

8 (10 minutes). What does the following program do? Give a sequence of system calls that it and its subprocesses might execute.

```
#include <unistd.h>
int main (void) { return fork () < fork (); }
```