UCLA Computer Science 111 (Fall 2013) Midterm
100 minutes total, open book, open notes

Name: Matt Nguyen Student ID: 804037897

1	2	3	4	5	6		
10	2	12	6	6	18		64

4 1a (4 minutes). Give an example of a race condition involving two GNU/Linux processes invoking 'write' at about the same time. Explain what might go wrong as a result of the race.

3 1b (4 minutes). Is a similar race condition possible if one process is invoking 'write' while the other is invoking 'waitpid'? Why or why not?

3 1c (4 minutes). Likewise, for two processes both invoking 'waitpid'.

2 (10 minutes). Suppose that, instead of using the convention that the 'INT' instruction is used to make system calls, we instead use the convention that a program uses a RDTSC instruction to make a system call. The RDTSC instruction loads the processor's 64-bit time stamp counter (a count of CPU cycles, approximately) into the EDX and EAX register pair. It's normally unprivileged, but if the the TSD flag in the (privileged) CR4 register is set, RDTSC is a privileged instruction.

Will this convention work? Explain why or why not. Regardless of whether it works, explain the most serious objection to the proposal.

3 (12 minutes). Suppose we change GNU/Linux to use linear quantum (LQ) scheduling with preemption. In this approach, the quantum length varies as a linear function of the identity of the process that's currently running. If there are R runnable processes, the scheduling quantum for process P (where 1 <= P <= R) is P milliseconds. Assume that processes are sorted by process-ID when determining quantum length.

Compare and contrast LQ to round-robin (RR) with a fixed quantum of 10 ms, in terms of average wait time, average turnaround time, throughput, and fairness. State any assumptions you're making.

4. Suppose you are running on a SEASnet GNU/Linux server in an environment where 'ulimit -a' reports the following:

```
cpu time                  (seconds, -t) 5
max user processes                (-u) 10
open files                        (-n) 20
pipe size              (512 bytes, -p) 8
```

3 4a (10 minutes). What will the following program do? Describe the series of system calls that it will execute, in a plausible order. If there's more than one fundamentally different way it could behave, explain each one.

```
#include <unistd.h>
int main (void) {
  int fd[2];
  while (! pipe (fd))
    fork ();
  return 0;
}
```

3 4b (10 minutes). Likewise, for the following program:

```
#include <unistd.h>
int main (void) {
  int fd[2];
  while (! fork ())
    pipe (fd);
  return 0;
}
```

Consider the following implementation of read_sector as discussed for our paranoid application discussed in class.

```
 1  void read_sector (int s, int a)
 2  {
 3    while ((inb (0x1f7) & 0xc0) != 0x40)
 4      continue;
 5    outb (0x1f2, 1);
 6    outb (0x1f3, s);
 7    outb (0x1f4, s >> 8);
 8    outb (0x1f5, s >> 16);
 9    outb (0x1f6, s >> 24);
10    outb (0x1f7, 0x20);
11    while ((inb (0x1f7) & 0xc0) != 0x40)
12      continue;
13    insl (0x1f0, a, 128);
14  }
```

For each of the following changes to the program, explain what would happen to the application if we made that change. Assume each change happens independently of the others. If the change would break some uses of 'read_sector' but not others, say which would be broken.

5a (3 minutes). Change the two '&'s to '|'s.

5b (3 minutes). Interchange lines 5 and 9.

5c (3 minutes). Change all '>>'s to '>>='s.

5d (3 minutes). Change the 'void' to 'int', the first 'continue' to 'return 0', and append 'return 1' after line 13.

5e (3 minutes). In lines 3-13, change all occurrences of 's' to 'a'.

5f (3 minutes). Change the '1' in line 5 to '128' and change the '128' in line 13 to '1'.

5g (4 minutes). Insert 'while (rdtsc () & 0xff) continue;' after line 10, where rdtsc () is defined as follows:

```
unsigned long long rdtsc (void)
{
  unsigned long long a;
  asm volatile ("rdtsc":"=A" (a));
  return a;
}
```

(See (2) for how the RDTSC instruction works.)

6. Intel has announced a new set of instructions for the x86 and x86-64 architecture. It's called AVX-512, and among other things features operations on 32 new registers that are each 512 bits wide. It's arguably the biggest change yet to the x86 architecture, and will probably debut in 2015.

6a (5 minutes). Suppose we change WeensyOS 1 to treat these new AVX-512 registers in the same way as all the other user-mode registers. How should this affect its scheduling mechanism?

6b (5 minutes). Suppose our WeensyOS variant uses round-robin scheduling with blocking and with a clock interrupt every 10 ms, and we change its scheduling mechanism to support AVX-512 as described in (a). How will this affect performance of existing applications, which do not use AVX-512?

6c (7 minutes). Propose a change to the scheduling policy of (b) that should improve its overall system performance, without altering the scheduling mechanism of (a). Explain why it should improve performance.

6d (7 minutes). Propose a change to the scheduling mechanism of (a) that should improve overall system performance without altering the scheduling policy of (b). Explain why it should improve performance.