Median: 49, Mean: 51.2, SD: 16

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

66
100

Name: NICOLE WONG Student ID: [redacted]

1	2	3	4	5	6	7	8
1	5	2	13	9	12	14	10

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.

it traps & gets to kernel, which is what you wanted!

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.

12

7. Consider the following implementation of read_sector:

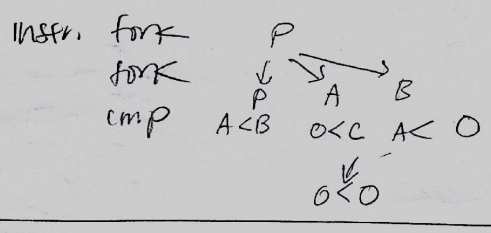
```
void wait_for_ready (void) {
  while ((inb (0x1f7) & 0xC0) != 0x40)
    continue;
}

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 (); }
```



CS111
Midterm
pg. 1

NICOLE WONG

1. Ubuntu, being a Linux system, uses hard modularity via virtualization. This OS divides instructions into privileged and non-privileged instructions that use traps to switch from user mode (non-privileged instructions only) to kernel mode (privileged and non-privileged instructions allowed). soft modularity

2. The string 'four' is echoed and used as stdin for lab0. The files score, and, 7, years, ago are created if they did not exist before, and if they already existed then they are truncated of their previous contents. Once lab0 has finished running, the file ago should contain the string 'four' since it was the last output flag used, and the lab0 program uses the last output option as stdout.

3. The xuml idea is completely crazy, because an RETI instruction will prompt the system to remain in user mode and keep executing the user program. This is effectively a no-op instruction, and does not trap into the OS kernel to perform the syscall as it is supposed to do.

4. a) ./simpsh --rdonly b --creat --trunc --wronly a
--pipe --append --creat --wronly c --pipe
--creat --trunc --wronly d --creat --trunc --wronly err
--command 0 3 1 head -n 20
--command 2 6 4 sort --command 5 7 8 tail --
./simpsh --creat --append --rdwr d --pipe
--creat --append --wronly err
--command 0 2 3 cat --command 1 0 3 cat

4b) since the simpsh program cannot write to stderr on the console, instead of errors from the shell commands being redirected to show on the console, I redirected the error

messages of the tail command and the 2nd line of the shell script to a file called err.

4.c) 0

4d) Add in a flag --stderr that allows for the simpsh program to print out error messages from commands to stderr on the console rather than redirecting to a file. 2

4e) ./simpsh --catch SIGSEGV --abort 2

use the signal # for SIGSEGV here!

5.a) The average wait time of T2R is longer than that of RR because a single process in T2R can keep running for more quantums and keep others waiting. For example, suppose long process A arrives, then processes B & C. Using T2R, A runs, and since the probability of running again is 0.5, it is possible for A to run for several more quanta. B and C subsequently have to wait for longer than in normal RR, where B would have begun executing 1 quantum after A's first quantum finished. The utilization of the CPU for both scheduling algorithms are the same, since both keep running processes in quantums. 5

5.b) Starvation is possible with T2R. Suppose we have a very long process A running, with processes B and C in the waiting queue. If the T2R scheduler happens to choose A to run for another quantum each time (since the probability 0.5 is independent of past runs), A could keep running, starving out B & C. 4

how likely?

1) 1

D

CS111
Midterm
Pg. 2

NICOLE WONG [redacted]

6.a) If 2 of these programs were running in parallel, and they both happened to store the variable n at the same location, then both could try to alter the variable at the same time, leaving the incorrect

Same Signal will not interrupt itself.

2

Suppose the user has pressed ^C, signalling SIGINT to the program and begins executing the signal handler. However, suppose that the user presses ^C again, issuing another call to the signal handler. Now suppose that both of these handlers try to alter the variable n at the same time, but they both take the value n, add 1, and store it back. This only leaves the value n+1 in n, but has printed 2 lines. This can cause the program to print more than 256 lines.

6.b) Suppose ^C was pressed, and the program calls the signal handler code, and prints out "not signal n" but doesn't increment n. Then, ^C is pressed again. This means that 2 signal handlers are running. The second handler takes the value n (which has still not been incremented) and prints out the line "int signal n=N" with the same value of n. Now there are 2 lines with the same value for N.

3

6.c) Similarly to parts a and b, if 2 handlers are running at the same time and both are trying to do printf, it is possible that the data stream to stdout is interleaved with output from the 2 handlers, leaving 2 "=" signs in the same line.

2

6.d)

0

6.e) Line 12 can be removed because you only need to set the signal handler for SIGINT once for the program, ~~Doing so again is unnecessary~~ and the behavior for catching SIGINT will be established for the entire duration of the program.

5

7.a) This ~~could~~ means that you don't wait for the disk controller to finish reading the data stored in sector 5 into the port 0x1f0, so when insl is called, you could be ~~re~~ storing gibberish into the variable a.

3

7.b) ~~This sets the incorrect~~ Since the registers for the disk controller are 1 byte each, this is harmless.

3

7.c) This is harmless since it doesn't matter what order you set the disk controller registers in, only that they be set to the correct values.

3

7.d) Line 7 issues the command to the disk controller to read the sector by setting the status control register to 0x20. However, if this is called before the sector offset registers are complete, the address for the sector is incorrect, and causes the controller to read in gibberish.

3

7.e) This is harmless. The program will just wait for the disk controller to be ready again.

2

efficiency

CS111
Midterm
pg. 3

Nicole work

8. This can be broken down into calling fork twice, then comparing the return values.
First the parent process calls fork() ~~and receives~~ twice and receives the children PIDs, and returns whether or not the PID(A) of the first fork was less than the PID of the second fork. The resulting children processes are A and B. After the first parent fork, child A receives a value of zero, and A calls fork. ~~This~~ If no error, the child A then receives the PID of child C and returns the comparison of 0 and PID(C). Child B receives a value of zero and compares PID(A) < 0. Child C ~~then forks a child doesn't~~ returns the comparison of 0 < 0.

– 0