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

52

Name: Hajime Hayano Student ID: 604460151

1	2	3	4	5	6	7	8
1	5	2	13	9	13	3	6

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:

```c
 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:

```c
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.

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

1) Ubuntu uses hard modularity insulating the users from the kernel by going through an intepreter like system calls

-2 soft modularity

2) The file "score" should have "four \n" written to it while the program writes to stderr about invalid use of --output for and, 7, years, and ago. This is because the get_opt_long should get input and the first file of output and all other --output files should be denied because lab0 should only take 1 output file

-0

3) The whole point of using INT for syscalls is to go from ~~user~~ mode to kernel mode to execute privileged instructions and INT take the control from the user and gives it to the kernel. When returning from an INT with RETI, the control is being given back to the user_mode from the kernel, so using RETI to program syscalls is counter intuitive and will not work.

4) a)

--creat --trunc

./simpsh --rdonly b --wronly a --pipe --command 0 3 1 \
head0 --creat --append --wronly c --pipe --creat --trunc
--command 2 6 4 sort --rdwr d
--command 5 7 ~~1~~ tail --pipe --command ~~1~~ 9 ~~1~~
cat --command 9 ~~1~~ ~~1~~ cat d --wait

4

4b) Since our simpsh was not able to take options after commands (ex. head -n 20) ~~or be able to both truncate and append to the same file (>d, >>d)~~, our simpsh will differ in the behavior such that head will take the preset amount of the starting lines of b and ~~d~~ ~~will only have the result of (head <b | sort | tail) once, not twice.~~

0

4c) If instead of piping between the two cat calls on the second line, we opened command w/ the same input and output file director, it might never read the end of a file and read/ append forever.

4

4d) Allow argument options in --command by using get_opt_long -only ~~and implement --concatenate~~

4e) ./simpsh --creat --rdwr a --command 0 0 0 echo "Fooled ya!" --abort

5

5a) depending on how many jobs we are cycling through, the average wait time changes. This is because if we are cycling through 2 jobs, we have $\frac{1}{2}$ a chance of waiting 1 more cycle. However, if we are cycling through 4 jobs, each time we risk having to wait longer, so there is only $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$ chance of running on time.

utilization?

4

RR has way better avg wait time.

5 b) Starvation refers to a job not having a chance to run at all. This is unlikely as everytime a ~~job~~ quantum expires, we have half a chance of moving on and the probability scheduler being stuck on one job diminishes by half everytime. Also when a job is done, the ~~quantum~~ scheduler moves on to the next job in the cycle so unless there is an infinitely long job and we get tails everytime we flip a coin (which $\lim_{n\to\infty} \frac{1}{2}^n$ converges to 0 so unlikely/impossible) we will not get starved

Only 1 thread

13/21

6 a) if n=255 and two threads execute 11 at the same time before executing line 13, n will add 2 before getting checked and wrap around to n=1

2

6 b) if the instruction pointer for one thread gets blocked right after reading the n value for %d but before incrementing the n.

3 ?

6c) race condition of writes where 2 writes are writing to stdout w/o atomicity and one write gets blocked right after writing the = and the other write writes "= %d"

3

6 d) If the signal is the dump core signal that calls to be run due to it priority

0

6.e) we can remove line 12 since signal does not need to be called multiple times.

5

17 a) 0 It is not going to change anything because.

b) since s is an integer, the result s& 0xff and
0 s& 0xfff is diferent, so the behavior of the code will change.

c) Theres not gonna be a difference b/c they don't need to be run in order since there are no
3 dependencies

d) yes because "6" and "7" has overlapping
0

e)

10) its going to return 4 values, 2 0's and 2 1's assuming there is no error. why

-4