UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION , April, 2015
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 2
Exam Type: A
Examiner – D. Yuan

Please Read All Questions Carefully! Please keep your answers as concise as possible.
You do not need to fill the whole space provided for answers.

There are ***19*** *total numbered pages,* ***8*** *Questions.*
You have 2.5 hours. Budget your time carefully!

Please put your FULL NAME, UTORid, Student ID on THIS page only.

Name: ____

UTORid: ____

Student ID: ____

Grading Page

	Total Marks	Marks Received
Question 1	8	
Question 2	6	
Question 3	16	
Question 4	28	
Question 5	15	
Question 6	7	
Question 7	20	
Question 8	10	
Total	120	

Question 1 (8 marks, 1 mark each): Virtual memory

Assume you have a small virtual address space of size 64 KB. Further assume that this is a system that uses paging and that each page is of size 8 KB.

(a) How many bits are in a virtual address in this system?

(b) Recall that with paging, a virtual address is usually split into two components: a virtual page number (VPN) and an offset. How many bits are in the VPN?

(c) How many bits are in the offset?

(d) Now assume that the OS is using a *one-level page table*. How many entries does this page table contain?

Now assume again you have a small virtual address space of size 64 KB, that the system again uses paging, but that each page is of size **4 bytes (note: not KB!)**.

(e) How many bits are in a virtual address in this system?

(f) How many bits are in the VPN?

(g) How many bits are in the offset?

(h) Again assume that the OS is using a *one-level page table*. How many entries does this one-level page table contain?

Question 2 (6 marks): Inverted page table

For an OS that uses paging, besides the one-level and multi-level page table designs as we discussed in the class, another page table design is called *inverted page table (IPT)*. In an IPT, the number of entries equals to the number of physical frames in memory. Each entry in IPT contains the physical frame number, virtual page number, and other bits such as Valid, Modify, etc.

(a)(2 marks) Assume a system with virtual address space being 4GB, and the physical memory space is 4MB, and each page is of size 4KB, how many entries does an IPT have?

(b)(4 marks) Compared to one-level page table (non-inverted) as we discussed in the class, what are the *advantages* of using IPT? What are the *disadvantages*?

Question 3 (16 marks): Scheduling

Assume we have three jobs that enter a system and need to be scheduled. The first job that enters is called A, and it needs 10 seconds of CPU time. The second, which arrives just after A, is called B, and it needs 15 seconds of CPU time. The third, C, arrives just after B, and needs 10 seconds of CPU time.

For all questions involving round-robin, assume that there is no cost to context switching. Also assume that if job X arrives just before Y, a round-robin scheduler will schedule X before Y.

(a)(1.5 marks) Assuming a shortest-job-first (SJF) policy, at what time does B finish?

(b)(1.5 marks) Assuming a round-robin policy (with a time slice length of 1 second), when does job A finish?

(c)(1.5 marks) Assuming a round-robin policy (with a time-slice length of 1 second), when does job B finish?

(d)(1.5 marks) Assuming a round-robin policy (with an unknown time-slice which is some value less than or equal to 2 seconds), when does job B finish?

(e)(2 marks) Assuming a round-robin policy (with an unknown time-slice), for what values of the time-slice will B finish before C?

Now assume you have a multi-level feedback queue (MLFQ) scheduler. Recall that an MLFQ scheduler groups jobs into different priorities, and for jobs in the same priority a round-robin scheduling is used. Recall that the priority value is calculated with the following formula as we discussed in the lecture:

priority value = nice + base + (recent CPU usage/2)

where

recent CPU usage = (last value + CPU count used by this process in this time slice)/2

Assume the following:

- nice always equals to 0
- base always equals to 0
- There are only 2 levels of priorities (thus 2 queues):
    - Level 1 is for the jobs with priority value < 1,
    - Level 2 is for the jobs with priority value >= 1
- The time-slice for the jobs in level 1 is 2 seconds, the time-slice for the jobs in level 2 is 4 seconds
- The timer interrupt fires once every 0.125 second, thus the CPU count increments by 8 in every second
- Whenever a job is moved into a different queue, it is always appended to the end of the queue
- The priority value and recent CPU usage for each job are re-calculated when (1) a time slice ends, (2) a job terminates, or (3) a job blocks on I/O
- The overhead of context switching and scheduling is negligible

(f)(2 marks) Why do we want to assign smaller time-slice for jobs in Level 1?

(g)(2 marks) How does MLFQ avoid the starvation of a job?

(h)(4 marks) Now given the same three jobs, A, B, and C as above, fill in the tables below to show how they are scheduled under MLFQ. Each box represents 1 second, your job is to fill in the job that is scheduled during that second. The first 2 cells are already filled in for you, which means that for the 1st and 2nd second, job A is scheduled to use the CPU. Assume the jobs don't perform any I/O.

A	A																
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36

Question 4 (28 marks, 4 marks each question): File system

In this question, we are going to unearth the data and metadata from a very simple file system. The disk has a fixed block size of 16 bytes (pretty small!) and there are only 20 blocks overall. A picture of this disk and the contents of each block is shown below (each cell represents 4 bytes, and the ID of the block is at the bottom of each column):

HINT: Super Block			HINT: Root INODE						
0	1	1	1	0	0	0	0	0	1
1	1	1	1	2	2	1	1	1	2
2	10	18	14	11	12	3	15	19	0
3	0	0	0	17	13	0	0	0	8
Blk.0	Blk.1	Blk.2	Blk.3	Blk.4	Blk.5	Blk.6	Blk.7	Blk.8	Blk.9.

foo	3	7	hi	a	10	11	i	ECE	0
4	4	8	10	1	goo	bar	luv	7	0
bar	5	9	you	b	11	oof	ECE	344	0
5	6	10	12	2	goo	da	344	8	0
Blk.10	Blk.11	Blk.12	Blk.13	Blk.14	Blk.15	Blk.16	Blk.17	Blk.18	Blk.19.

The disk is formatted with a very simple file system. The first block (Blk. 0) is a super block. It has just four integers in it: 0, 1, 2, and 3, in that order. *The root inode (i.e., inode for "/") of this file system is in Blk. 3 in the diagram*.

The format of an inode is also quite simple:

```
  type: 0 means regular file, 1 means directory
  size: number of blocks in file (can be 0, 1, or 2)
  direct pointer: the ID of the first block of file (if there is one)
  direct pointer: the ID of the second block of file (if there is one)
```

(assume that each of these fields takes up 4 bytes of a block)

Finally, the format of a directory is also quite simple:

```
  name of file
  the block ID of the inode of the file
  name of next file
  the block ID of the inode of next file
```

(again assume that each field takes up 4 bytes of a block)

Finally, assume that in all cases, no blocks are cached in memory. Thus, you always have to read from this disk all the blocks you need to satisfy a particular request. Also assume you never have to read the super block (just to make your life easier).

Now you have to answer some questions:

(a) To read the contents of the root directory, which blocks do you need to read?

(b) Which files and directories are in the root directory? List the names of each file/directory as well as its type (e.g., file or directory).

(c) Starting at the root, what are names of *all* the reachable *regular* files in this file system?

(d) What are the names of *all* the reachable *directories*?

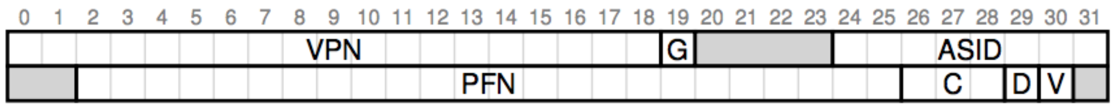
(e) What is the biggest file in the file system?

(f) What are the contents of the biggest file?

(g) What blocks are free in this file system? (that is, which blocks are not in use?)

Question 5 (15 marks): TLB structure

Assume the TLB structure looks like this:



The VPN and PFN fields should be self-explanatory, as should the V field (valid). The ASID field is an address-space identifier field used to store the PID to which the address-space belongs. The D field is a dirty bit. ignore any other fields.

On this system, the OS has a software-managed TLB. Thus, the OS is responsible for installing the correct translation when a TLB miss occurs. When finished with the update to the TLB, the OS returns from a trap, and the hardware retries the instruction.

Unfortunately, just before the OS updates the TLB, sometimes a bit gets flipped and thus the wrong translation ends up in the TLB! For each of the following fields, both (1) describe what the field is used for and (2) explain what you think would happen if a bit gets flipped in said field just before the OS installs the entry in the TLB:

(a) VPN:

i (1 mark) What is the VPN field for?

ii (2 marks) What would happen if a bit in the VPN got flipped?

(b) PFN: (same questions)

i (1 mark) What is the PFN field for?

ii (2 marks) What would happen if a bit in the PFN got flipped?

(c) ASID: (same questions)

i (1 mark) What is the ASID field for?

ii (2 marks) What would happen if a bit in the ASID got flipped?

(d) Valid: (same questions)

i (1 mark) What is the Valid bit for?

ii (2 marks) What would happen if the valid bit got flipped?

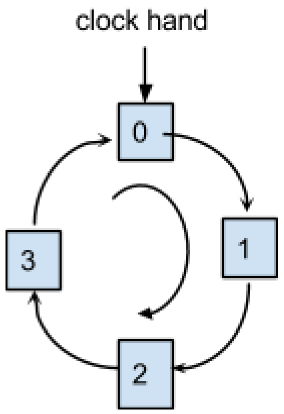
(e) Dirty: (same questions)

i (1 mark) What is the Dirty bit for?

ii (2 marks) What would happen if the dirty bit got flipped?

Question 6 (7 marks): TLB replacement

Assume we have a TLB that has 4 entries. Each entry has a unique ID with the value 0, 1, 2, or 3. The OS implements an LRU-clock algorithm as the TLB replacement policy. The diagram below shows how the OS organizes the 4 TLB entries as a circular clock. The initial location of the clock hand is at entry 0.



Answer the following questions:

(a)(2 marks) In practice, OSes uses LRU-clock algorithms to approximate the actual LRU algorithm. Why don't OS programmers implement LRU?

(b)(5 marks) The following table shows 10 memory accesses from a process. The first column shows the virtual address **(not the virtual page number)** of each memory access in hexadecimal. Fill in the last 2 columns of the table. Put into the second column whether the access results in a TLB hit or not, and put the ID of the TLB entry in the last column (in the case of a hit, this is the entry that contains the mapping, whereas in the case of a miss this is the entry that is chosen by the LRU-clock replacement algorithm).

Assume the page size is 4KB. At the beginning the TLB is empty, with the reference bits being all 0. The information for the first 4 accesses are already provided for you.

Virtual address	TLB hit?	TLB entry

0x02030F02	No	0
0x02041032	No	1
0x321A0BC1	No	2
0x31828AA3	No	3
0x31827325		
0x0204187A		
0x3182832A		
0x7A18198A		
0x7A181002		
0x02030F24		

Question 7 (20 marks): OS161

Consider the following snippet from function `vm_fault` in OS161's **dumbvm.c** (variable declarations are omitted):

```
 1: int vm_fault(int faulttype, vaddr_t faultaddress) {
 2:  spl = splhigh();
 3:
 4:  for (i=0; i<NUM_TLB; i++) {
 5:    TLB_Read(&ehi, &elo, i);
 6:    if (elo & TLBLO_VALID) {
 7:      continue;
 8:    }
 9:    ehi = faultaddress;
10:    elo = paddr | TLBLO_DIRTY | TLBLO_VALID;
11:    DEBUG(DB_VM, "dumbvm: 0x%x -> 0x%x\n", faultaddress, paddr);
12:    TLB_Write(ehi, elo, i);
13:    splx(spl);
14:    return 0;
15:  }
16:  kprintf("..."); // write this log message yourself
17:  splx(spl);
18:  return EFAULT;
19: }
```

(a)(3 marks) When is this function being called?

(b)(3 marks) What does the "for" loop from line 4 - 15 do?
Search for an invalid entry in TLB, and install a new page mapping.

(c)(3 marks) What is line 6-8 doing?
Check if this entry contains a valid mapping. if so, continue to examine the next entry.

(d)(2 marks) What does line 9, "`ehi = faultaddress;`", do?

(d)(2 marks) For line 10, why do we need to bit-wise-or `paddr` with `TLBLO_VALID`?

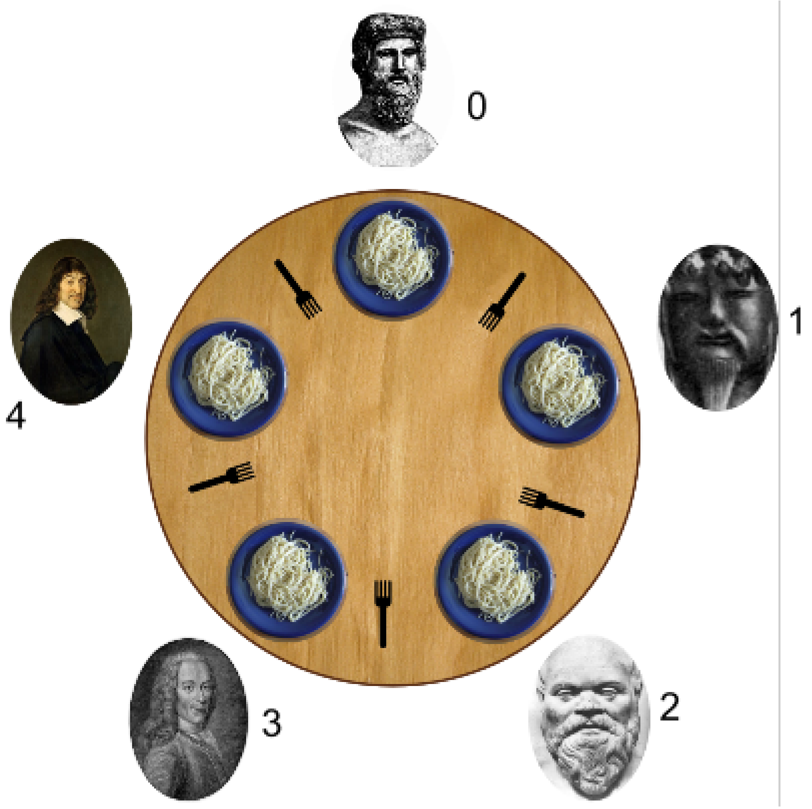
(e)(2 marks) When vm_fault returns 0 at line 14, what does it indicate?

(f)(2 marks) What should be the proper log message at line 16 (and what variables should you log)?

(g)(3 marks) Why do we need two `splx(spl)` in this code (one at line 13 and the other at line 17)? What happens if we delete the one at line 17?

Question 8 (10 marks): Synchronization

Let's consider a classic concurrency problem: the dining philosophers. Five silent philosophers sit at a table around a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers. The picture below illustrate the problem.



Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he ***has both left and right forks.*** Each fork can be held by only one philosopher and so a philosopher can use the fork only if it's not being used by another philosopher. After he finishes eating, he needs to put down both forks so they become available to others. A philosopher can grab the fork on his right or the one on his left as they become available, ***but can't start eating before getting both of them.***

Eating is not limited by the amount of spaghetti left: assume an infinite supply.

Now your job is to write a program to allow the philosophers enjoy their spaghetti! In particular, your program has to satisfy the requirements stated above. In addition, your program should further *allow concurrency: for example, philosopher 0 and 2 should be able to eat at the same time.*

The following code snippet provides the skeleton of the code. It does not work properly since it lacks the proper synchronization. Your job is to further use appropriate synchronizations to make it correct.

```
#define N 5 // 5 philosophers: YOU CANNOT CHANGE THIS LINE

int forks[N]; // the state of each fork.
         // 0 means it is available, 1 means it is in use.

void take_fork(i) {
  forks[i % N] = 1;
}

void put_fork(i) {
  forks[i % N] = 0;
}

/* You have to implement this function correctly. You are not allowed
 * to modify the parameter number nor the parameter type. */
void philosopher (int i) {
  while (1) { // YOU CANNOT CHANGE THIS WHILE LOOP
    think(); // philosopher is thinking; YOU CANNOT CHANGE THIS LINE
    take_fork(i);
    take_fork(i+1);
    eat(); // philosopher is eating; YOU CANNOT CHANGE THIS LINE
    put_fork(i);
    put_fork((i+1) % N);
  }
}

/* You cannot make any change to main()*/
void main() {
  for (int i = 0; i < N; i++) {
    // create a new thread by calling "philosopher(i)"
    thread_fork (i, philosopher);
  }
  wait(); // wait forever
}
```

NOTE: the part of code that you are not allowed to change or delete is indicated in the comment. You can make any changes to the other parts of the code (e.g., you don't have to use the "int forks[N]" at all). You can use any type of synchronization primitives we discussed in the class or the combination of them.

Your code goes here:

```
#define N 5 // 5 philosophers
















void philosopher (int i) {
  while (1) {
    think(); // philosopher is thinking;









    eat(); // philosopher is eating;











  }
}
void main() { // Don’t touch this function
  for (int i = 0; i < N; i++) {
    // create a new thread by calling “philosopher(i)”
    thread_fork (i, philosopher);
  }
  wait(); // wait forever
}
```

This page is blank. You can use it as an overflow page for your answers.