UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, April, 2018
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 4
Exam Type: A
Examiner – D. Yuan

*There are **13** total numbered pages, 7 Questions.* Please Read All Questions Carefully! You will likely find question 7 to be the most difficult. Budget at least 60 minutes for that question.

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

Name: _____

Student Number: _____

	Total Marks	Marks Received
Question 1	5	
Question 2	10	
Question 3	10	
Question 4	20	
Question 5	7	
Question 6	12	
Question 7	36	
Total	100	

Question 1 (5 marks): True or False

(1) Two contiguous pages in virtual address space must be mapped with two contiguous physical pages
   (a) True
   (b) False

FALSE.

(2) On x86, the hardware (MMU) makes the page replacement decision
   (a) True
   (b) False

FALSE.

(3) In Not-Recently-Used (NRU) replacement algorithm, if there is only one page with the largest counter value (no ties), NRU is essentially the same as LRU
   (a) True
   (b) False

TRUE.

(4) Every memory load and store instruction will trap into the OS kernel
   (a) True
   (b) False

FALSE.

(5) In Unix File System, a directory does not have an inode associated with it
   (a) True
   (b) False

FALSE.

Question 2 (10 marks): Multiple choices. Circle all correct answers

(1) Which of the followings are true for LRU-clock algorithm.
   (A) In the worst case, it needs to scan every physical page
   (B) In the worst case, it needs to scan every virtual page
   (C) The worst case is when the reference bit of every page is 0
   (D) The worst case is when the reference bit of every page is 1
   (E) None of the above

A, D

(2) Which of the followings are true for a scheduling algorithm.
   (A) When an algorithm improves throughput, it always improves response time

(B) When an algorithm improves response time, it always improves throughput
(C) Shortest Job First scheduling algorithm can cause a job to be starved
(D) First Come First Serve can cause a job to be starved
(E) None of the above

C (or E if they assume Non-preemptive SJF)

(3) Which of the followings are true for memory management.
(A) A process always sees the virtual address
(B) The same virtual address in two processes always maps to different physical addresses
(C) On MIPS, MMU handles page faults but not TLB faults
(D) On x86, MMU handles page faults but not TLB faults
(E) None of the above

A

(4) Which of the followings are true for fragmentation?
(A) Fixed partition can lead to internal fragmentation
(B) Fixed partition can lead to external fragmentation
(C) Variable partition can lead to external fragmentation
(D) File system defragmentation is to resolve internal fragmentation
(E) File system defragmentation is to resolve external fragmentation
(F) None of the above

A, C, E

(5) Which of the followings are true for multi-level page table?
(A) It is designed to save memory space
(B) It simplifies TLB fault handling
(C) It reduces the page translation time
(D) In the worst case, it uses more memory than single-level page table
(E) None of the above

A, D

Question 3 (10 marks): Scheduling

Consider MLFQ (multilevel feedback queue) scheduling algorithm we discussed in lecture, which is used by Unix. For all the questions below, assume that the nice value of each job is the same (i.e., users do not customize the nice value).

(1) (6 marks) Which of the followings are properties of MLFQ? Circle all correct answers.
(a) When a new job A is submitted to the system, it will always be the one chosen by the scheduler to run next

(b) Two jobs cannot have the same priority value
(c) It favors interactive jobs
(d) It favors CPU-intensive jobs
(e) If a job frequently blocks, its priority increases (i.e., priority value decreased)
(f) After a job waits a while in the ready queue, its priority decreases (i.e., priority value increased)
(g) MLFQ uses preemptive scheduling

A, C, E, G
Marking: 2 marks for each mistake made [edit distance].

(2) (4 marks) Consider the following scenario: there is a job A that has been running for a while. Starting from time T, new jobs are being submitted to the system continuously. What happens to A? Explain use 2 sentences max.

A will be starved [2marks], because new jobs arrive at the highest priority. Even A's priority increases because it waited long time, its priority will not be higher than the newly arrived job.

Question 4 (20 marks): OS161

(1) (2 marks) In os161, what type of memory management scheme was used for dumbvm?
(a) Fixed partition
(b) Variable partition
(c) Paging
(d) Segmentation
(e) None of the above

D.

In os161, there is a function:
```
int vm_fault(int faulttype, vaddr_t faultaddress)
```

(2) (2 marks) When is this function being called? 1 sentence max.
On a TLB fault.

(3) (2 marks) What are stored in the two parameters? 2 sentences max.
Fault type: the cause of the TLB fault.
Fault address: memory address that caused the TLB fault.

(4) (14 marks) Below is the implementation of this vm_fault function that is simplified from dumb_vm. However, we have left out some important parts for you to fill. For each missing part (blank), choose the proper code to complete the function.

```
int vm_fault(int faulttype, vaddr_t faultaddress) {
  paddr_t paddr;

  ____; // Blank 1
  faultaddress &= ______; // Blank 2
  paddr = …; // assume it is set properly

  for (i=0; i<NUM_TLB; i++) {
    TLB_Read(&ehi, &elo, i);
    if (elo & _____)  // Blank 3
      continue;

    ehi = ______; // Blank 4
    elo = ______ | TLBLO_DIRTY | TLBLO_VALID; // Blank 5
    TLB_Write(ehi, elo, i);
    ____; // Blank 6
    return 0;
  }
  ____; // Blank 7
  return EFAULT;
}
```

```
Blank 1:
   (a)   int spl = splhigh();
   (b)   int spl = spl0();
   (c)   No operation is needed;
```

a

```
Blank 2:
   (a)   0xfffff000
   (b)   0xffff0000
   (c)   0xfffe000
   (d)   0x0000ffff
   (e)   0xffffffff
   (f)   0x00000000
   (g)   0xffff8000
   (h)   0xfffc000
```

a

Blank 3:
   (a)   0xffffffff
   (b)   0x00000000
   (c)   TLBLO_DIRTY
   (d)   TLBLO_VALID
   (e)   TLBLO_NOCACHE
   (f)   TLBLO_PPAGE
   (g)   TLBLO_GLOBAL

d

Blank 4:
   (a)   faultaddress
   (b)   0x00000000
   (c)   0xffffffff
   (d)   paddr
   (e)   TLBHI_VPAGE

a

Blank 5:
   (a)   faultaddress
   (b)   0x00000000
   (c)   0xffffffff
   (d)   paddr
   (e)   TLBLO_PPAGE

d

Blank 6:
   (a)   splx(spl);
   (b)   splhigh();
   (c)   No operation is needed;

a

Blank 7:
   (a)   splx(spl);
   (b)   splhigh();
   (c)   No operation is needed;

a

Question 5 (7 marks): Page size
Modern OSes support two page sizes: 4KB (the default) and 2MB (huge page). Answer the following questions on the trade-offs between page sizes.

(1) **(3 marks)** Which of the followings are true? Circle all correct answers
- (A) Using huge page results in smaller page table
- (B) Using huge page results in larger page table
- (C) Using huge page leads to less TLB misses
- (D) Using huge page leads to more TLB misses
- (E) Using huge page can waste memory due to internal fragmentation
- (F) Using huge page can waste memory due to external fragmentation

A, C, E
Marking: -1 for each mistake

(2) **(4 marks)** You're only given the MIPS processor you used in the lab, can you implement huge page with all the benefits you answered in part (1)? Explain with 2 sentences max.

No. Because It requires hardware support. Your MIPS processor does not support 2MB pages, therefore its TLB cannot translate huge page VPNs.

Question 6 (12 marks, 3 marks each): TLB misses
The TLB miss rate of the following code changes with the values of MAX and STRIDE. Assume the TLB has 32 entries with a 4KB page size and uses LRU. Assume 4 byte ints and variables are page aligned (i.e., the first integer of the array is allocated on the first 4 bytes of a page)

```
1. int data[MAX];
2. int foo() {
3.   int value = 0;
4.   for (int i = 0; i < 1000; i++) {
5.     for (int j = 0; j < MAX; j += STRIDE) {
6.       value += data[j];
7.     }
8.   }
9. }
```

(1) What's the smallest values of MAX and STRIDE that cause every access to the array *data* to result in a TLB miss?

STRIDE: 1024
MAX: 32*1024+1 = 32769 (Assuming instruction/stack not occupying TLB.)
Or,
MAX: 30*1024+1=30721 (Assuming instruction/stack occupy 2 TLB entries.)
Or
Max: 28*1024+1 = 28673 (they may assume *value, i, j* each is allocated on a separate page)

(2) State each TLB miss that is NOT caused by the array *data*. For each miss clearly explain what caused the miss.

1 miss on the text/code to load instructions
1 miss on the stack to load *value*
Students may assume 2 more misses on i and j if they're allocated on different pages.

(3) If STRIDE is made 10 times the value of part (1) (smallest to cause 100% TLB miss rate) while MAX remains the same, would the TLB miss rate (a) increase, (b) decrease, or (c) remain the same? Explain with 2 sentences max.

Decrease. In part a the inner loop was only 1 entry too large to fit in the TLB, so by skipping over pages with a large stride everything would fit in the TLB. This would let you get hits on the next iterations of the outer loop.

(4) If MAX and STRIDE are both made 10 times the value of part (1), would the TLB miss rate (a) increase, (b) decrease, or (c) remain the same compared with part (1)? Explain with 2 sentences max.

Decrease. Same as above.

Question 7 (36 marks, 2 marks for each question): File system
Warning: carefully read this question; it's long!

In this question, we are going to closely consider how a very simple file system works. The disk has a fixed block size of 16 bytes (pretty small!) and we are only showing the first 20 blocks. A picture of this disk and the contents of each block is shown below (each cell represents 4 bytes, the ID of the block is at the bottom of each column, and the 4 cells in each block are ordered from top to bottom):

0	d1	d	c	1	0	1	1	i lo	file
1	2	8	5	1	1	1	1	ve e	syst
2	d2	0	0	2	9	3	12	ce34	em.
1	3	0	0	4	0	4	0	4.	
Blk.1	Blk.2	Blk.3	Blk.4	Blk.5	Blk.6	Blk.7	Blk.8	Blk.9	Blk.10.

1	a	0	Lond	fall	0	Linu	Free	0	0
1	8	2	on B	ing	1	x Wi	BSD	1	2
3	b	14	ridg	down	15	ndow	Sola	3	3
0	4	0	e is	.	0	s.	ris.	5	12
Blk.11	Blk.12	Blk.13	Blk.14	Blk.15	Blk.16	Blk.17	Blk.18	Blk.19	Blk.20.

The disk is formatted with a very simple file system. The first block (Blk. 1) is a super block. It has just 3 integers as magic number: 0, 1, 2, and the inode-number of the root directory, which is 1 in this case.

The format of an inode is also quite simple:

```
type: 0 means regular file, 1 means directory
number of links: number of directories that contain this file
direct pointer: the block number of the first block of file
direct pointer: the block number of the second block of file
```

Note that for the two direct pointer fields, 0 is not a valid block number, and it is used to indicate

that the block has not been allocated. Also 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: 0 indicates invalid
the inode number of the inode of the file; 0 indicates invalid inode
name of next file: 0 indicates invalid
the inode number of the inode of next file; 0 indicates invalid inode
```

Again assume that each field takes up 4 bytes of a block.

The inode map of this file system is given to you as follows. Recall that an inode map is a data structure that maps inode number to the block number of this inode.

Inode number	Block number of this inode
1	5
2	8
3	11
8	13
4	16
5	6

Now you have to answer some questions:

(1) Write down all the names of all files and directories on this file system.

/
/d1
/d2
/c
/d1/a
/d1/b
/d2/d

(2) What are the free blocks of this file system? Write down all the block numbers.

7, 10, 17, 18, 19, 20

(3) There are two file names that point to the same file (i.e., links). What are these two file

names? What is the content of this file?

/d1/a, /d2/d, "London Bridge is"

For each of the following operations, answer which blocks will be changed and its content after the change. Use a table to answer each question. For example, if you answer:

1	5			
2	6			
3	7			
4	8			
Blk._5_	Blk._8__	Blk.____	Blk.____	Blk.____

It means that block 5 and 8 are modified to the contents shown in the table. You don't have to use all 5 columns for each question. Each operation is conducted in isolation, not cumulative. Whenever you need to allocate a block, you allocate the free block that has the smallest block number. Whenever you need to assign a new inode number, you use the lowest inode number that is not used (inode numbers are always greater than 0). If the operation cannot be performed, e.g., unlink() a file that does not exist, simply leave the table empty.

For (4)-(8), you *don't* have to consider the order of the modification to the blocks. Also, assume the file system is not log structured. You do NOT need to consider the writes to inode map.

(4) Create a file "/d2/e". Assume you need to allocate one block for this file, and initialize this block to 0 for each byte.

d	0	0		
8	1	0		
e	10	0		
6	0	0		
Blk._3__	Blk._7___	Blk.__10__	Blk.____	Blk.____

You could also use 7 for data block and 10 for inode. In that case, the answer is:

d	0	0		
8	1	0		

e	7	0		
6	0	0		
Blk._3__	Blk._10_	Blk.__7__	Blk.____	Blk.____

(5) unlink ("/d1/a"). Assume unlink does not need to reclaim the blocks that become free.

0	0			
0	1			
b	14			
4	0			
Blk._12_	Blk._13_	Blk.____	Blk.____	Blk.____

If they write blk 12 as "b400" it is also OK.

(6) mkdir("/d2/d3") create an empty directory named "/d2/d3/".

d	1	0		
8	1	0		
d3	10	0		
6	0	0		
Blk._3__	Blk._7__	Blk._10_	Blk.____	Blk.____

They could use blk 10 for inode and blk 7 for the empty dir. In that case the answer is the following:

d	1	0		
8	1	0		
d3	7	0		
6	0	0		
Blk._3__	Blk._10__	Blk._7_	Blk.____	Blk.____

(7) rename("/d1/a", "/d2/aa") -- rename the file "/d1/a" to "/d2/aa". (So "/d1/a" no longer exists.)

0	d			
0	8			
b	aa			
4	8			
Blk._12_	Blk._3__	Blk.____	Blk.____	Blk.____

(8) Overwrite the content of file "/d1/a" by changing every letter to capital letter.

LOND				
ON B				
RIDG				
E IS				
Blk._14_	Blk.____	Blk.____	Blk.____	Blk.____

Now, for (9)-(13), assume this file system is a log structured file system. Answer the same questions with other assumptions unchanged. You only need to answer which blocks will be modified thus they are moved to the end of the log; no need to provide the content. You do not need to write the blocks that will be newly allocated; again, only the ones that are *modified*. You do NOT need to consider the writes to inode map.

(9) Create a file "/d2/e". Which blocks are modified?

3, 11

(10) unlink ("/d1/a"). Which blocks are modified?

12, 13, and 8 (because block 12 is relocated)

(11) mkdir("/d2/d3") make an empty directory named "/d2/d3". Which blocks are modified?

3 and 11

(12) rename("/d1/a", "/d1/aa") -- rename the file "/d1/a" to "/d1/aa". Which blocks are modified?

12 and 8

(13) Overwrite the content of file "/d1/a" by changing every letter to capital letter. Which blocks are modified?

14, 13

Now you need to consider *crash consistency.* The system can crash at any given time, and after you reboot, your file system can be in an inconsistent state. If your system crashes in the middle of a file system operation, we say the system is consistent when after reboot the user sees the file system in either one of the two states: (1) as if the operation has completed, or (2) as if the operation has not occurred at all. For example, if the system crashes when you're creating a file "/a/b", your file system is inconsistent if, after the reboot, you see there is an entry named "b" in the directory "/a", but it points to an invalid inode or a file block that has not been allocated. The file system is considered to be consistent, for example, if you do NOT see the file "b" under directory "/a" even though a block has been allocated for this file "/a/b". In other words, you do not need to worry about the content of blocks that are free (i.e., the blocks that are not used by any files in the file system).

As we discussed in class, you could prevent your file system goes into an inconsistent state by carefully ordering the block writes involved in certain file system operations. For each of the operations in (14)-(18), if the system crashes in the middle of the operation, answer whether you can guarantee consistency just by ordering the block writes, and if so, write down the order. No explanation is needed, just write yes or no, and the list of block numbers if the answer is yes.

Assume that each write made to a block is atomic, i.e., you can write multiple bytes in a block in a single block write and the system won't crash in between. However, the system can crash between writing to two different blocks. Also assume that the hard drive will not reorder your write requests.

For (14)-(18), you also need to consider the modification to the inode map. Assume that the inode map is stored in block 21. Note that it is OK (i.e., not considered as inconsistent) if the inode map contains some mappings that are not used by the file system.

Also assume the file system is NOT log structured.
(14) Create a file "/d2/e".

Yes. 7, 10, 21, 3
Write 3 at last.

(15) unlink ("/d1/a").

No.

(16) mkdir("/d2/d3") make an empty directory named "/d2/d3".

Yes. 7, 10, 21, 3. Write 3 at last.

(17) rename("/d1/a", "/d1/aa") -- rename the file "/d1/a" to "/d1/aa".

```
Yes. 12
```

(18) Overwrite the content of file "/d1/a" by changing every letter to capital letter.

Yes. 14