UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION , April, 2016
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 2
Exam Type: A
Examiner – D. Yuan

***Please Read All Questions Carefully!*** All the relevant information is in the question. Please keep your answers as concise as possible.

*There are **15** total numbered pages, **9** 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 | 12 | |
| Question 2 | 12 | |
| Question 3 | 9 | |
| Question 4 | 15 | |
| Question 5 | 8 | |
| Question 6 | 14 | |
| Question 7 | 8 | |
| Question 8 | 12 | |
| Question 9 | 10 | |
| Total | 100 | |

**Question 1: True or false (12 marks, 2 marks each)**

(1) MIPS uses hardware managed TLB

(2) It is possible that a virtual address is the same as the physical address

(3) Using a large page size incurs more internal fragmentation

(4) Using a large page size incurs more external fragmentation

(5) Log structured file system does not use inode

(6) There can be more than one inode for one directory

**Question 2 (12 marks): Scheduling**

Consider MLFQ (multi-level feedback queue) scheduling algorithm, which is used by Unix. It consists of a number of principles. First, circle the principles that are actually part of the MLFQ policy we discussed in the lecture. *Assume that a higher Priority(Job) value indicates a higher actual priority of the job.*

(1) If Priority(A) > Priority(B), A runs (B doesn't)
(2) If Priority(A) < Priority(B), A runs (B doesn't)
(3) If Priority(A) = Priority(B), A and B runs in round-robin fashion
(4) If Priority(A) = Priority(B), A runs to completion, then B
(5) Two jobs cannot have the same priority value
(6) Once a job uses up its time slice at a given level, its priority is reduced
(7) Once a job uses up its time slice at a given level, its priority is increased
(8) Once a job uses up its time slice at a given level, it moves to the end of the round-robin queue
(9) Once a job uses up its time slice at a given level, it exits
(10) If a job has not used the CPU in the last time slice, its priority is unchanged
(11) If a job has not used the CPU in the last time slice, its priority decreases
(12) If a job has not used the CPU in the last time slice, its priority increases
(13) When a job performs I/O, it will likely result in an increase of its priority
(14) When a job performs I/O, it will likely result in a decrease of its priority

Now, write down all the principles that come into play in each of the following example traces of MLFQ behavior (use numbers from above). The X-axis denotes time. Note that * marks when A arrives, if the information is relevant. Also note that Q1-Q3 indicates three different priority queues, but you have to infer which queue corresponds to the high/medium/low priority.

```
                                                          Rule(s)?
    *
Q3: AAA
Q2:    AA
Q1:      AAAAAAAAAAA ..


           *
Q3:        AAA
Q2:           AA
Q1: BBBBBBBB


            *
Q3:         AAA
Q2:            AABB
Q1: BBBBBBBB        ABABABAB


            *
Q3:                         A
Q2:         A
Q1: BBBBBBBB  BBBBBBBBBBBBBB BBBBBB
```

**Question 3 (9 marks, 3 marks each): Filesystem**

Consider the following Linux command:

```
cp --parent /usr/bin/ls ./
```

It will make all the parent directories as needed. Assume that directory “usr” does not exist under “./”, this command will end up creating “./usr”, “./usr/bin”, and “./usr/bin/ls”.

Q1: How many writes does it take to execute this command? Explain each one. Assume each write can only write to one disk block, e.g., writing to two blocks require two writes. Assume an inode occupies one block. Assume the ls consists of 2 disk blocks.

Q2: Assume the file system guarantees the atomicity of a single block write, but not multiple writes, i.e., two consecutive writes can fail in between but a single write cannot fail in the middle. If the OS crashes in the middle of the execution of the above command, can we end up with an empty "./usr/bin/"? Why?

Q3: Can you reorder the block writes so that the above command will be atomic to the user, i.e., even if the OS crashes at an arbitrary time, after the system restart the user can only see one of the two following states: A) “./usr/bin/ls” is correctly created, or (B) there is no directory named “usr” under ./?

**Question 4 (15 marks, 3 marks each): Extent-based filesystem**

One of the new features of today's file system is to use extent instead of indirect blocks. For example, Linux introduced extent in its EXT4 file system. In extent-based filesystem, the inode structure is similar to Unix inode we discussed in the lecture and stores the metadata and 12 direct map file blocks. However, it does not have the indirect, double-indirect, and triple-indirect blocks. Instead, it has four fields, each called an "extent". An extent is a pointer-length pair; the pointer is just a disk block address, and the length is how many consecutive blocks, starting at the pointer, are part of this extent. So an extent with the value: <183200, 4> indicates that the 4 disk blocks starting at block address 183200 are part of the file.

Please answer the following questions. Assume that each disk block is 4KB, and each block address requires 4 bytes. Assume the length field in each extent has 32 bits.

In this question, we use the terminology "metadata" refers to the inode (which includes extents in extent-based filesystem), indirect, double-indirect, and triple-indirect block in a traditional Unix filesystem. Also assume the filesystem doesn't cache the block addresses from the metadata between two consecutive file data block reads, i.e., each file data block read requires to a new look-up in the metadata from the beginning.

(1) If a file's size is 7KB, what is the minimum number of reads to the "metadata" of this file that is required to read the entire file in a Unix file system? How about in an extent-based filesystem?

(2) If a file's size is 48KB, what is the minimum number of reads to the metadata of this file that is required to read the entire file in a Unix file system? How about in an extent-based filesystem?

(3) If a file's size is 4MB, what is the minimum number of reads to the metadata of this file that is required to read the entire file in a Unix file system? How about in an extent-based filesystem?

(4) If a file's size is 16MB, what is the minimum number of reads to the metadata of this file that is required to read the entire file in a Unix file system? How about in an extent-based filesystem?

(5) Do you see any problems with this extent-based filesystem?

**Question 5 (8 marks): Reverse engineering the page table**

In this problem, we consider address translation in a system with a simple one-level linear page table (an array of page table entries, or PTEs). Virtual page N is simply the Nth entry in this table.

Parameters:

- Virtual address space size is 32KB
- Page size is 4KB
- Physical memory size is 64KB

Here is a trace of virtual addresses and the physical addresses they translate to (or perhaps an invalid access):

```
  VA          PA
0x1063 --> 0x2063
0x67b4 --> 0x87b4
0x584a --> 0xe84a
0x4dfe --> Invalid
0x388a --> Invalid
0x1c6b --> 0x2c6b
0x50a9 --> 0xe0a9
0x0bc6 --> Invalid
0x2a9f --> 0x9a9f
0x742b --> Invalid
0x4b5e --> Invalid
0x5597 --> 0xe597
```

Can you reconstruct the page table entries from this? For each entry that you can construct, please do so; otherwise, mark down the entry as "UNKNOWN".

Format: valid bit followed by Physical Page Number (PPN).

```
                      Valid         PPN
Page Table Entry 0:
Page Table Entry 1:
Page Table Entry 2:
Page Table Entry 3:
Page Table Entry 4:
Page Table Entry 5:
Page Table Entry 6:
Page Table Entry 7:
```

**Question 6 (14 marks, 2 marks each): TLB**

The following question traces TLB behaviour over time. For each question you should produce a series of hits (“H”) and misses (“m”), one for each memory access. The string “mHmH” would mean “TLB miss” followed by a “TLB hit” followed by a “miss” followed by a “hit”.

In all cases, ignore instruction references (i.e., do not worry about their effects on the TLB).

Also, always assume the array (discussed below) is PAGE ALIGNED.

(1) Assume you have a 1-entry TLB. Assume you access contiguous 4-byte integers in a large array, starting at index 0 and going to the max size. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern?

Pattern:

(2) Assume you have a 2-entry TLB with LRU replacement of TLB entries. Assume you access contiguous 4-byte integers in a large array, again starting at 0. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern?

Pattern:

(3) Assume you have a 1-entry TLB. Assume you access *every other* 4-byte integer in a large contiguous array, starting at index=0, then index=2, etc. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern?

Pattern:

(4) Assume you have a 4-entry TLB with FIFO replacement. Assume you repeatedly access all 4-byte integers in a small contiguous array of 24 integers, *in a loop*. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern, for the *fourth* run through the loop?

Pattern:

(5) Assume you have a 4-entry TLB with LRU replacement. Assume you repeatedly access all 4-byte integers in a contiguous array of 40 integers, in a loop. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern, for the *fourth* run through the loop?

Pattern:

(6) Assume you have a 4-entry TLB with FIFO replacement. Assume you repeatedly access all 4-byte integers in a contiguous array of 40 integers, in a loop. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern, for the *fourth* run through the loop?

Pattern:

(7) Assume you have a 4-entry TLB with Last-In-First-Out replacement. Assume you repeatedly access all 4-byte integers in a contiguous array of 40 integers, in a loop. Assume the page size is 32-bytes. What is the hit/miss pattern for that access pattern, for the *fourth* run through the loop?

Pattern:

**Question 7 (8 marks, 4 marks each): Threads and synchronization**

Consider the following programming problem. *Assume the semantics of the synchronization libraries (e.g., lock_acquire/release, cv_wait/broadcast/signal) are exactly the same as the ones in the lab assignments.*

**(a)** Consider the Bounded Buffer (producer/consumer) problem: there are *multiple* producer threads and *multiple* consumer threads. All of them operate on a shared bounded buffer. A producer inserts data into the buffer one item at a time, and a consumer removes data from the buffer one item at a time. The correctness criteria is that the producers cannot insert into

a buffer that is full, and consumers cannot remove any items from an empty buffer.

Now consider the following code:

```
void *producer(void *arg) {
  while (true) {
    lock_acquire(&lock);
    while (number_of_items(buffer) == MAX) // MAX is the buffer length.
      cv_wait(&cv, &lock);
    insert_item(buffer); // inserts an item into shared buffer
    cv_broadcast(&cv);
    lock_release(&lock);
  }
}

void *consumer(void *arg) {
  while (true) {
    lock_acquire(&lock);
    while (number_of_items(buffer) == 0)
      cv_wait(&cv, &lock);
    tmp = consume_item(buffer); // removes an item from the shared buffer
    cv_broadcast(&cv);
    lock_release(&lock);
    kprintf("%d\n", tmp);
  }
}
```

Assume the lock and condition variable are properly initialized. Now, does this code work correctly? Why?

**(b)** Now let's make a small change to the code above: replace "`cv_broadcast`" with "`cv_signal`". The modified code is as below:

```
void *producer(void *arg) {
  while (true) {
    lock_acquire(&lock);
    while (number_of_items(buffer) == MAX) // MAX is the buffer length.
      cv_wait(&cv, &lock);
```

```
    insert_item(buffer); // inserts an item into shared buffer
    cv_signal(&cv); // here!
    lock_release(&lock);
  }
}

void *consumer(void *arg) {
  while (true) {
    lock_acquire(&lock);
    while (number_of_items(buffer) == 0)
      cv_wait(&cv, &lock);
    tmp = consume_item(buffer); // removes an item from the shared buffer
    cv_signal(&cv); // here!
    lock_release(&lock);
    kprintf("%d\n", tmp);
  }
}
```

Does this code work correctly? Why?

**Question 8 (12 marks, 4 marks each): OS161**

Answer the following questions about OS161.

(1) There is a data structure "coremap". What is it? Answer in one sentence.

(2) Describe a scenario when "coremap" is needed? Answer in no more than two sentences.

Consider the following code from OS161, answer question (3).

```
 1 void as_activate(struct addrspace *as) {
 2   int i, spl;
 3
 4   spl = splhigh();
 5
 6   for (i=0; i<NUM_TLB; i++) {
 7     TLB_Write(TLBHI_INVALID(i), TLBLO_INVALID(), i);
 8   }
 9
10   splx(spl);
11 }
```

(3) Why are line 6 - 8 necessary? (We're not asking you what line 6-8 is doing.)

**Question 9 (10 marks): Multi-level page table**

Assume you have a 15-bit virtual address, with page size = 32 bytes. Assume further a two-level page table, with a page directory (i.e., first level page table) which points to second level page tables. Each entry in page directory is 1 byte, and consists of a valid bit followed by the physical page number of the second-level page table. Each entry in the second page table is similar: a valid bit followed by the physical page number of the page where the desired data resides.

There is only one page used to store the page directory, which resides in physical page 18, and the entire page 18 is used to store page directory entries.

The following physical page contents are made available to you:

```
Page  10: 7f 7f 7f 7f 7f 7f 7f 7f 7f f0 7f a4 7f 7f 7f 7f
          7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f

Page  18: 7f c0 ea f9 ed 8b db ba d9 c1 84 8a b3 7f da eb
          9a 85 ab 87 e5 97 b1 df 86 ec e7 ad f2 b9 d5 f8

Page  30: 13 1b 03 11 1e 12 16 18 0f 08 12 10 0a 1a 0b 0e
          17 19 1b 14 07 1a 1c 16 17 0f 0f 12 04 14 1a 05

Page  57: a1 7f 7f 7f 7f 7f 7f 9e 7f 7f 7f 7f 7f 7f 7f 7f
          7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e0 7f 7f

Page  90: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
          7f e2 7f 7f 7f 7f 7f c7 7f 7f 7f 7f 7f 7f 7f 7f

Page  98: 16 0d 18 10 02 0e 01 1c 1d 0a 09 17 06 05 05 0a
          13 1d 06 1d 11 1b 19 04 14 03 00 0c 17 11 05 1a

Page 126: 16 03 14 07 07 01 0c 11 03 05 0c 00 19 05 1c 11
          09 02 13 01 0a 1e 19 16 12 13 17 1b 03 1b 1e 12
```

(1) (3 marks) In translating virtual address 0x3a3a, which physical pages are accessed?

(2) (2 marks) What is the final data value returned?

(3) (3 marks)  In translating virtual address 0x74f6, which physical pages are accessed?

(4) (2 marks) What is the final data value returned?