UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, April, 2017
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 4
Exam Type: A
Examiner – D. Yuan

Please Read All Questions Carefully!

For question 2, 5, 7, you will receive *20%* of the mark if you choose NOT to answer the question (i.e., leave it blank).

*There are **13** total numbered pages, 7 Questions.*

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

Name: ____

Student ID: ____

	Total Marks	Marks Received
Question 1	9	
Question 2	20	
Question 3	10	
Question 4	20	
Question 5	6	
Question 6	25	
Question 7	10	
Total	100	

Question 1 (9 marks): True or false. No explanation is needed.

(a) The page table of each process is stored in each process's address space.

(b) MIPS uses software managed TLB

(c) Inode of a file does not contain the content of the file.

(d) When we create a filesystem link, there can be multiple inodes to the same file.

(e) When you remove a file with the command 'rm filename', the disk blocks storing the file content will always be added to the free block maps of the file system.

(f) In a real-time system, earliest deadline first scheduling algorithm can always guarantee that every job meets its deadline.

(g) A page table is used to translate virtual address to physical address.

(h) A directory does not have an inode.

(i) On Unix OS, a file can have unlimited size as long as there is storage capacity.

Question 2 (20 marks, 4 marks each): OS161

Answer all of the questions below in the context of OS161 and MIPS. Your answer to each question should not exceed 140 characters.

(a) In lab 3, you were asked to implement a system call named sbrk(). What does this system call do?

(b) In lab 3 you need to implement a data structure called coremap. What does it do?

(c) Processes use virtual addresses and they are first being translated by TLB. But for the execution of kernel code, does it use virtual address? If so, how does the translation from virtual address to physical address happen?

(d) What is purpose of ELF magic number?

(e) How did you locate the arguments passed into each system call?

Question 3 (10 marks): Scheduling

Consider MLFQ (multi-level feedback queue) scheduling algorithm. 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 higher priority value indicates a higher priority):

(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 decreased unless it already has the lowest priority

(7) Once a job uses up its time slice at a given level, its priority is increased unless it already has the highest priority
(8) Once a job uses up its time slice at a given level, it is killed
(9) If a job blocks before it uses up its time slice, its priority is unchanged
(10) If a job blocks before it uses up its time slice, its priority decreases unless it already has the lowest priority
(11) If a job blocks before it uses up its time slice, its priority increases unless it already has the highest priority
(12) If a job has been waiting for a while, its priority is unchanged
(13) If a job has been waiting for a while, its priority decreases unless it already has the lowest priority
(14) If a job has been waiting for a while, its priority increases unless it already has the highest 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:             AABB
Q1: BBBBBBBB        ABABABAB
```

```
             *
Q3:          AA                A
Q2:            BB
Q1: BBBBBBBB     BBBBBBBBBBBBB BBBBBB
```

Question 4 (20 marks): Page replacement
Assume you have a small RAM that can only store 4 pages. The OS implements an LRU-clock algorithm as the page replacement policy. Each page frame in the RAM has a unique ID with the value 0, 1, 2, or 3. The initial location of the clock hand is at page frame 0. The page size is 4KB.

Answer the following questions:
(a)(2 marks) True or false (no explanation needed): each page table has exactly 4 entries for each process.

(b)(6 marks) The following table shows 12 consecutive memory accesses. The first column shows the process ID, the second 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 third column whether the access results in a page fault or not, and put the ID of the page frame in the last column that stores the page (regardless whether it's a page fault or not).

At the beginning the reference bit of the each page frame is set to 0, and none of the page frames are not mapped to any process. Assume there is no TLB or MMU on this computer, i.e., each memory access will result in a fault that invokes the OS, and the OS sets the reference bit every time a page frame is accessed.

PID	Virtual address	Page fault?	Page frame ID
3	0x02030F02	Yes	0
3	0x02030023		
4	0x02041032		
4	0x02030425		
4	0x31828AA3		
3	0x02030422		
3	0x31828325		
3	0x02030423		
4	0x3182832A		
4	0x7A181002		
4	0x02030426		
3	0x02030F24		

(c)(6 marks) Now let's add a TLB to this example. Assume the computer has a MIPS architecture. The TLB has 4 entries, does not have bits to store the PID in each entry, and uses a FIFO replacement policy. At the beginning, TLB is empty. Complete the same table again below. Note: the OS only updates the reference bit of a page frame on a TLB fault. Also, note that we are still asking you about page fault, not TLB fault.

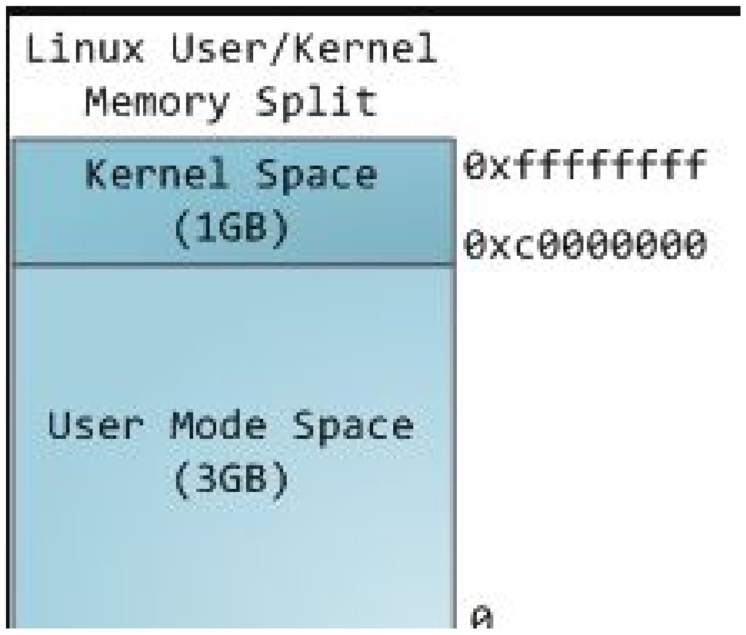
PID	Virtual address	Page fault?	Page frame ID
3	`0x02030F02`	Yes	0
3	`0x02030023`		
4	`0x02041032`		
4	`0x02030425`		
4	`0x31828AA3`		
3	`0x02030422`		
3	`0x31828325`		
3	`0x02030423`		
4	`0x3182832A`		
4	`0x7A181002`		
4	`0x02030426`		
3	`0x02030F24`		

(d) (6 marks) Now assume that the TLB becomes a tagged TLB, i.e., it has PID bits in each entry and uses it on each TLB lookup. Other assumptions are the same as the previous question. Complete the same table one more time.

PID	Virtual address	Page fault?	Page frame ID
3	`0x02030F02`	Yes	0
3	`0x02030023`		
4	`0x02041032`		

4	0x02030425		
4	0x31828AA3		
3	0x02030422		
3	0x31828325		
3	0x02030423		
4	0x3182832A		
4	0x7A181002		
4	0x02030426		
3	0x02030F24		

Question 5 (6 marks): On Linux systems, 1GB of the virtual address space of each process is reserved for OS's use. In other words, the address space of each process looks like the following:



What is the benefit of this design? (Hint: this 1GB stores the instructions or data used by system calls such as read() or write().)

Question 6 (25 marks): 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):

0	sk	usr	foo	2	1	1	log	0	0
1	2	2	4	3	1	1	stru	2	1
2	3	bin	bar	4	2	3	ctur	17	11
8	4	3	5	5	0	0	e	18	0
Blk.0	Blk.1	Blk.2	Blk.3	Blk.4	Blk.5	Blk.6	Blk.7	Blk.8	Blk.9.

lock	page	ELF	ELF	foo	0	1	i	i	ELF
unlo	tabl	0	2	4	1	1	luv	luv	0
ck	e	0	3	ls	12	14	ECE	ECE	1
0	0	0	4	7	0	0	344	344	2
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 3 integers as magic number: 0, 1, 2, and the inode-number of the root directory, which is 8 in this case.

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), 0
otherwise
```
(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 inode number of the inode of the file
  name of next file
  the inode number of the inode of next file
```
(again assume that each field takes up 4 bytes of a block)

This file system has only the following files (and the directories on their paths):
/usr/foo, /usr/bar, /bin/foo, /bin/ls

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

Inode number	Block number of this inode
4	8
5	9
7	15

Now you have to answer some questions:

(a) **(4 marks)** Write down all the names of directories on this file system.

(b) **(5 marks)** Complete the inode map by filling the empty cells of the table above. (Every inode should have an entry in the table, but you don't have to use all the rows.)

(c) **(5 marks)** What is a free map of a file system? What is the free map of this file system?

(d) **(3 marks)** What is the file content of /usr/foo?

(e) **(3 marks)** Which block(s) contain the data content of /bin/ls?

(f) **(5 marks)** Now you execute "rm /bin/foo" once, how does this file system change? Describe all the changes, including change to any block, inode map, and free map.

Question 7 (10 marks): A barrier is a primitive to synchronize multiple threads in a parallel program. When a thread reaches the barrier, it will check to see if other threads have arrived at the barrier. If one or more threads have not arrived, the thread will wait. When all threads reach the barrier, they can begin their execution on the next phase of the computation (i.e., barrier function returns). An example of using a barrier is as follows:

```
// thread code, before entering barrier
 EnterBarrier();
 // reaches here only after all threads have entered barrier
```

There are several issues that you need to consider. First, there is no master thread that controls the threads, waits for each of them to reach the barrier, and then tells them to proceed. Instead, the threads must determine themselves when they should wait or proceed. Second, the barrier mechanism should work for many dynamic programs. The number of threads during the lifetime of the parallel program is unknown in advance, since a thread can spawn another thread, which will start in the same program stage as the thread that created it. Third, a thread may end before the barrier. In all cases, all threads must wait at the barrier for all other threads that are alive before anyone is allowed to proceed. Your job is to design and implement the data structure and primitive operations for barrier. Your solution must support creation of a new thread (an additional thread that needs to synchronize), termination of a thread (one less thread that needs to synchronize), waiting when a thread reaches the barrier early, and releasing waiting threads when the last thread reaches the barrier.
You should first define the data structure of barrier and then show how to implement the following barrier primitives with semaphores pseudo code:

- `barrier_t InitBarrier();`

Initialize a barrier. barrier_t is the type of struct (see code below).

- `ThreadCreated( barrier_t );`

This primitive will be called when a thread is created. You don't need to create the actual thread. It just contains the necessary operations on barrier.

- `ThreadEnded( barrier_t );`

This primitive will be called when a thread terminates.

- `EnterBarrier( barrier_t );`

Threads will call this primitive to enter a barrier.

Your solution should never busy-wait. You should use only semaphore primitives and NOT use any other synchronization primitives in your implementation. Think carefully about efficiency and avoid unnecessary looping.

Write your code here:

```
struct barrier_t {









}

barrier_t InitBarrier(void) {









}
void ThreadCreate ( barrier_t *b) {









}
void ThreadEnded ( barrier_t *b) {
}

void EnterBarrier (barrier_t *b) {








}
```