UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, April, 2013
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 **20** total numbered pages, **11** 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	10	
Question 2	7	
Question 3	7	
Question 4	8	
Question 5	10	
Question 6	8	
Question 7	8	
Question 8	6	
Question 9	15	
Question 10	12	
Question 11	9	
Total	100	

Question 1 (10 marks, 2 marks each): True or False *(No explanations needed)*

(a) The main reason to have a multi-level page table is to speed up address translation.

□True □False

False. Main reason is to save space in memory.

(b) The main reason to have a hardware TLB is to speed up address translation.

□True □False

True. That is the point of the TLB, to cache address translations and hence speed up the entire process.

(c) Using a multi-level page table increases TLB hit rate.

□True □False

False. TLB hit time is not affected by page-table structure

(d) For better performance, OS should place i-nodes close to their indexed files on hard disk.

□True □False

True. Because the i-node and its indexed file are often accessed together.

(e) On unix file systems, the i-nodes for regular files have different formats than the i-nodes for directories.

□True □False

False.

Question 2 (7 marks): Hard disk

From a hard disk's perspective, what are the three components that make up the data accessing time? Briefly describe the meaning of each component.

Seek: move the head to the appropriate track.
Rotational delay: wait for the block to rotate under the head.
Data transfer.

Question 3 (7 marks): Virtual memory

What are the benefits of having virtual memory, instead of asking programmers to directly use physical memory? Please list two.

1. Programs use more memory than the size of physical memory.

2. Programs can use memory from 0 - XX bytes, and can be run on different machines without change.

3. Programmers do not need to worry about memory sharing with other processes.

Question 4 (8 marks): Scheduling I

In this question, you will answer some questions about scheduling algorithms. Assume that sorting is an expensive computation, and requires some amount of time per element sorted; you can estimate the time to sort *NumJobs* elements with a function *Sort(NumJobs)*. Further assume that switching between jobs takes a fixed amount of time (say, on a context switch, when one job ends and another begins, or to switch in a new job for the first time); we will call this value *Switch*.

First, assume three jobs arrive at the same time, of lengths 30, 20, and 10 (they arrive in this order). Assume that *Sort(NumJobs) = NumJobs × 10,* and that *Switch = 0* (no switching overhead). Assume we use non-preemptive SJF (Shortest Job First) Policy.

(a)(2 marks) How long will the system take to ***compute the schedule***?

Answer: Three jobs means 3 × 10 or 30

(b)(2 marks) How long will it take to run the jobs once the schedule has been computed?

Answer: Total run time: 10 + 20 + 30 or 60

Now assume that switching is expensive, too; in fact, *Switch* = 10.

(c)(2 marks) Including scheduling and switching overhead, what is the average ***turnaround*** time for each job?

Answer:

turnaround time for job 1: 30 (sorting) + 10 (switch) + 10 (execution of job 1) = 50
turnaround time for job 2: 30 (sorting) + 10 (switch) + 10 (execution of job 1) + 10 (switch) + 20 (execution time for job 2) = 80
turnaround time for job 3: 30 (sorting) + 10 (switch) + 10 (execution of job 1) + 10 (switch) + 20 (execution time for job 2) + 10 (switch) + 30 (execution for job 3) = 120
Average turnaround time: (50 + 80 + 120) / 3 = 83
(Note: it is OK to assume there is no context switch between 'sort' and 'job 1').

Now, assume we are tired of SJF, and would like to use preemptive Round Robin policy, with timeslice = 20.

(d)(2 marks) Assuming the same scheduling and switching overhead (*Sort(NumJobs)* = *NumJobs* × *10,* and *Switch* = *0*), what is **the total execution time** from the arrival of the three jobs to the completion of all three?

Answer: there is no sort time in Round Robin.
Here is the execution sequence:
0 - 20: job 1 (10 remains)
20 - 40: job 2 (job 2 finishes)
40 - 50: job 3 (job 3 finishes)
50 - 60: job 1 (job 1 finishes)

Total execution time: 60.
Note: it is OK to assume switch = 10.

Question 5 (10 marks): Unix File System

Consider the Unix File System as we discussed in the lecture. It uses i-nodes to index directories and files.

(a)(5 marks) Assume only the address of the data block for root directory (not the i-node of root directory) is cached. All other contents stored in the File System are not cached. Assume the content of directories can be stored in only one data block.

How many disk operations are needed to read the first byte of a file: /1/2/mydoc.txt? Explain your answer.

Answer:

1st: datablock of '/', get the inode location of '/1';

2nd: access the inode of '/1', get the address of datablock of '/1';

3rd: access the datablock of '/1', get the inode location of '/1/2';

4th: access the inode of '/1/2', get the address of datablock of '/1/2';

5th: access the datablock of '/1/2', get the inode location of '/1/2/mydoc.txt';

6th: access the inode '/1/2/mydoc.txt', get the address of datablock '/1/2/mydoc.txt';

7th: access the datablock of '/1/2/mydoc.txt' and read the first byte

total 7 disk accesses.

(b)(5 marks) Assume within an i-node there are 12 direct pointers, a single-indirect pointer, and a double-indirect pointer. Assume a 4KB block size, and disk addresses that are 32 bits.

What is the maximum file size (measured in number of blocks) on this Unix system?

Answer:

12 direct: 12 blocks

*1 single-indirect: 1 block (4KB) full of addresses: 4KB = 2^10 * 4bytes = 2^10 blocks = 1024 blocks*

*1 double-indirect: 1 block full of single indirect addresses: 1024 addresses --- 1024 blocks of addresses, 1024 * 1024 blocks.*

*12 + 1024 + 1024*1024 blocks.*

Question 6 (8 marks): Synchronization I

Consider the following code:

```
Thread 1:
1 lock(L1);
2 lock(L2);
3 // critical section requiring L1 and L2 locked.
4 unlock(L2);
5 unlock(L1);

Thread 2:
6 lock(L3);
7 lock(L1);
8 // critical section requiring l3 and l1 locked.
9 unlock(L1);
10 unlock(L3);

Thread 3:
11 lock(L2);
12 lock(L3);
13 // critical section requiring l2 and l3 locked.
14 unlock(L3);
15 unlock(L2);
```

In the beginning, all three locks are initialized as “not locked”. Also assume the threads can execute in any arbitrary interleavings.

(a)(4 marks) Will there be any problem with this code? If yes, show an interleaving. If no, why?

Yes. There is a deadlock. Consider the following interleaving:

thread 1:

lock (L1)

thread 2:

lock(L3);

thread 3:

lock(L2);

Now there will be a circular wait: thread 1 waiting for L2 (held by thread 3), thread 2 waiting for L1 (held by thread 1), thread 3 waiting for L3 (held by thread 2).

(b)(4 marks) If there is a problem, how do you fix it? (Note, each critical section requires two different locks, you cannot change this assumption).

Acquire the locks in order the order of L1, L2, L3.

Question 7 (8 marks): Page Replacement

When the physical memory is full, a page fault will cause the OS to evict a page. Assume your OS uses NRU (LRU approximation, not LRU clock) algorithm, and there are only 6 physical pages. The value of reference bits and NRU Counters at the moment of page fault are listed as below.

Page #	Reference bit	NRU Counter
1	0	0
2	1	6
3	1	3
4	0	1
5	0	2
6	1	0

(a)(3 marks) After you run the NRU algorithm (before swapping), what are the values of reference bits and NRU counters? Please fill the table below.

Page #	Reference bit	NRU Counter
1	0	1
2	0	0
3	0	0
4	0	2
5	0	3
6	0	0

(b)(1 marks) Which page will the OS evict?
Evict page 5.

(c)(4 marks) Now you decide to use LRU clock algorithm, and the clock-hand at the time of page fault is at Page 2. What would be states after you run LRU-clock (before swapping)? Fill in the blanks below (you don't need the NRU counter in LRU clock anymore). **Which page will you evict?**

Page #	Reference bit
1	0
2	0 (note: '1' in this entry is also fine)
3	0
4	0
5	0
6	1

Evict page 4.

Question 8 (6 marks): Scheduling II

Recall that in Multiple-level feedback queues (MLFQ) scheduling algorithm, OS uses multiple queues to represent different job types. Different queues have different priorities. In fact, different queues also have different timeslice values. For example, if a queue has a timeslice value of 8ms, then when a job in that queue gets scheduled it has a timeslice of 8ms.

Now, you are to design MLFQ algorithm for Linux.

(a)(2 marks) Do you assign larger timeslice to jobs in a higher-priority queue or a lower-prioirty queue? Why?
Answer: The high-priority queues are usually given short time slices; they are comprised of interactive jobs, after all, and thus quickly alternating between them makes sense (e.g., 10 or fewer milliseconds). The low-priority queues, in contrast, contain long-running jobs that are CPU-bound; hence, longer time slices work well (e.g., 100s of ms). [Taken from Operating Systems: Three Easy Pieces, by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau]

(b)(2 marks) At some point, you will need to move a job from a higher-priority queue to a lower-priority queue *based on its execution history*. How do you make such a decision?

Answer: again we want to reward those interactive jobs --- jobs that are frequently interrupted. Therefore, if a job always uses up its entire timeslice without being interrupted, we should lower its priority since it is not likely interactive.

(c)(2 marks) Periodically you will also want to move ***all the jobs in the lowest priority queue*** to the highest priority queue, regardless of their execution history. Why?

Answer: To avoid starvation of the jobs in the lowest priority queue.

Question 9 (15 marks): TLB and Page faults

This question asks you to consider everything that happens in a system on a memory reference. Assume the following: 32-bit virtual addresses, 4KB page size, a 32-entry TLB, 1-level page table, and LRU replacement policies whenever such a policy might be needed by either hardware or software. Assume further that there are only 1024 pages of physical memory available. In this question, you will be running some test code and tell what happens when that code is run.

Before the test code is run, the following initialization code is run once (before testing begins). This code simply allocates (NUM_PAGES*PAGE_SIZE) number of bytes and then sets the first integer on each page to 0, where PAGE_SIZE is 4KB and NUM_PAGES is a constant (defined below). Assume malloc() returns page-aligned data in this example.

```
void *orig = malloc(NUM_PAGES * PAGE_SIZE);
void *ptr = orig;
for (i = 0; i < NUM_PAGES; i++) {
 (int *) *ptr = 0; // init first value on each page
  ptr += PAGE_SIZE;
}
```

The test code we are now interested in running is the following:

```
ptr = orig;
for (i = 0; i < NUM_PAGES; i++) {
 int x = (int *) *ptr; // load value pointed to by ptr
 ptr += PAGE_SIZE;
}
```

For these questions, assume we are only interested in memory references to the malloc'd region through ptr (that is, ignore stores to x and instruction fetches).

(a)(3 marks) How many TLB hits, TLB misses, and page faults occur during the test code when (please fill in the blanks below)...

	TLB hits	TLB misses	Page faults
(a) NUM_PAGES is 16	16	0	0
(b) NUM_PAGES is 32	32	0	0
(c) NUM_PAGES is 2048	0	2048	2048

The workload just loops over some pages accessing each one once. Thus, the second time through the loop with a small number of pages (less than the TLB size, for example) just yields

TLB hits. With a large number of pages, and LRU replacement, nothing we want is ever in the cache or TLB, hence all misses/page faults.

(b)(3 marks) Assume a memory reference takes roughly time M and that a disk access takes time D. Ignore the TLB lookup time. How long does the test code take to run (approximately), in terms of M and D, when (fill the blanks with the approximate runtime below)...

	Total execution time
(a) NUM_PAGES is 16	16M
(b) NUM_PAGES is 32	32M
(c) NUM_PAGES is 2048	2048 * 2M + 2048D

A hit means a load to memory which we assume costs M and hence the TLB hits just cost M. On TLB misses we need to consult memory twice, once for the page table (minimally) and once for the actual memory load, hence 2M per TLB miss. Finally, each page fault costs us a disk access.

(c)(6 marks) Now assume we change the various replacement policies in the system to MRU --- evict the Most Recently Used. Given this change, how many hits/misses/faults, and how long does the test code take to run (approximately), in terms of M and D, when (fill in the table)...

	TLB hits	TLB misses	Page faults
(a) NUM_PAGES is 16	16	0	0
(b) NUM_PAGES is 32	32	0	0
(c) NUM_PAGES is 2048	32	2016	1024

	Total execution time
(a) NUM_PAGES is 16	16M
(b) NUM_PAGES is 32	32M

(c) NUM_PAGES is 2048	32M + 2016 * 2M + 1024D

With MRU, the first 31 accesses get lodged in the TLB, and the first 1023 pages get lodged in memory; subsequent accesses simply push the most-recently used TLB entry/page out. The next time through the loop, the first 31 accesses are thus hits, and the first 1023 page accesses are in memory; everything else are misses until the last page, which also is a hit (do a small MRU example if this doesn't make sense). Thus, 32 TLB hits (the rest misses); 1024 page faults (the rest in memory).

(d)(3 marks) Finally, assume you are to run this code on a new machine that you know very little about. In fact, you wish to use the test code to learn how big the TLB is and how much memory is on the given system. How could you use the test code above to learn these facts about the physical hardware?

Answer:

Just looking for something simple here. You could basically time how long an iteration of the loop lasts. If it lasts something like NUM PAGES times memory access time, those are TLB hits; if it lasts twice that, TLB misses, and thus you can know how big the TLB is. One more big jump occurs when NUM PAGES is finally bigger than memory and causes lots of (very slow) disk accesses.

Question 10 (12 marks): OS161

(a)(4 marks) In OS161, there is a function named “vm_fault” --- when is this get called?

Answer: this is called when the address cannot be translated from the TLB -- either there is no such entry in TLB, or the translation is not valid.

(b)(4 marks) Paging and segmentation are two techniques to implement virtual memory, yet many OS supports both at the same time. Describe how you implemented segmentation using paging in lab 3.

Answer: Segmentation is implemented on top of paging. In the address_space data structure, simply record the size of each segment. But for each virtual address, in the end it will be translated via paging. Anything makes sense is acceptable here.

(c)(4 marks) Please describe a bug that you wrote in your os161 labs which was time-consuming to debug. How did you debug it?

Question 11 (9 marks): Synchronization II

- Consider a concurrent program with three threads only. Thread 1 repeatedly calls ProcedureA, to produce items of type A and place them in an unbounded array BuffA. The code is shown in the next 2 pages. Thread 2 repeatedly calls ProcedureB, to produce items of type B and place them in an unbounded array BuffB. Thread 3 repeatedly calls ProcedureC, which consumes one item of type A or B, depending on what items are available. If an item of type A is available, ProcedureC consumes it. If no item of type A is available but an item of type B is available, ProcedureC consumes the B item. If no A or B items are available, ***ProcedureC will block until some item is available***.

  Code for ProcedureA, ProcedureB, and ProcedureC is given on the next page, but without synchronization. Add synchronization primitives to this code as needed to ensure that: (1) BuffA and BuffB are accessed by only one thread at a time, and (2) ProcedureC blocks if no items are available and wakes up when items are available.

```
/* Global variables (including lock and semaphores) are declared here: */
array *BuffA, *BuffB;

semaphore sem; /* counting semaphore, initialized to 0*/
lock LA, LB;
```

```
ProcedureA() {
 item *A;

 produce (A);

 lock(LA);
 array_add(BuffA, A);
 unlock (LA);

 V(sem);

}
```

```
ProcedureB() {
 item *B;

 produce (B);

 lock(LB);
 array_add(BuffB, B);
 unlock (LB);

 V(sem);

}
```

```
ProcedureC {
 item *C;
// Add code here to check whether to (i) remove an item from BuffA by
// calling array_remove(BuffA, C), (ii) remove an item from BuffB by calling
// array_remove(BBuff, C), or (iii) block.
// Use consume(C) to consume the item.

 P (sem);
 lock (LA);
 if (length(BuffA) > 0)
 {
    array_remove(BuffA, C);
 }
 unlock (LA);

 if (C == NULL) {
   lock (LB);
   if (length(BuffB) > 0)
   {
     array_remove(BuffB, C);
   }
   unlock (LB);
 }

 if (C == NULL) {
   // should never happen!
   panic("Procedure C is woken up, but found no available item to consume");
 }

 consume(C);









}
```

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