UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2018 EXAMINATIONS

CSC369H1S

Duration - 3 hours

Aids Allowed:
One double-sided 8.5 x 11 cheat sheet (printed or handwritten)

Student number: _____

UtorID: _____

Last name: _____ First name: _____

Lecture section: L0101 L5101 (circle only the one you are enrolled in)

---

*Do **NOT** turn this page until you have received the signal to start.*

(Please fill out the identification section above, **write your name, UtorID and student#, circle your lecture section**, and read the instructions below.)

This final exam consists of **8 questions on 22 pages** (including this one and pages that have intentionally been left blank). A mark of **at least 40 out of 100 on this exam is required to pass this course.** *When you receive the signal to start, please make sure that your copy is complete.*

Answer the questions **clearly and legibly**. Answers that include both correct and incorrect or irrelevant statements will not receive full marks. **Be careful what each question asks! Be specific rather than vague in your answers!**

We have provided next to each question, a rough estimate on how much time you should spend on it. These are just guidelines and add up to less than 3 hours. **Use your time wisely** and answer first the questions that you are confident about, then come back to those you might need more time to think about. **Do not panic! Relax and keep focused!**

If you use any space for rough work, indicate clearly what you want marked.

Q1:	_____/10
Q2:	_____/16
Q3:	_____/10
Q4:	_____/12
Q5:	_____/10
Q6:	_____/12
Q7:	_____/16
Q8:	_____/14
Total:	_____/100

Good luck!

Q1. [10 marks - 1 mark each] True/False [5 minutes]

Indicate below, for each statement, whether it is (T)rue or (F)alse. Circle the correct answer.

T / F: Threads created within the same process share the same heap and global variables.

T / F: A process whose parent has died is called a zombie process.

T / F: Spinlocks should not be used to guard fairly short critical sections.

T / F: If no thread is blocked on a particular semaphore, then a signal on that semaphore will be recorded and let the first arriving thread proceed through a wait() operation on that semaphore without blocking.

T / F: Starvation is a problem that boils down to an inadequate scheduling algorithm.

T / F: In an MLFQ scheduler, the priority of a job gets increased if the job uses its entire time slice.

T / F: When a page fault occurs, a user process will modify its own page tables to bring in the page from disk.

T / F: The prefetching technique in a paged memory system relies primarily on programs exhibiting good locality.

T / F: A hard link to a directory is useful because it allows directories to be mapped in multiple places in the file system tree.

T / F: In a journaled file system like ext3, garbage collection must be performed to remove obsolete versions of inodes and file data.

Q2. [16 marks - 2 each] Reasoning questions – short answers [20 minutes]

1) [2 marks] Based on what we've learned throughout the course, how would you define an operating system? Be concise, but be specific.

2) [2 marks] Interrupt disabling and enabling can be used for implementing mutual exclusion. Name a situation where this approach does not work.

3) [2 marks] Consider a demand-paging system with a CPU utilization of only 20%. Can increasing the degree of multiprogramming (maximum number of processes allowed to use the CPU concurrently) lead to better CPU utilization? Explain in detail your rationale.

4) [2 marks] You are tasked with designing a security system application which compartmentalizes functionality into 10 standalone logical components which communicate and exchange information with each other. To execute these components concurrently, you have the option to spawn a set of processes, where each process handles one specific component, or alternatively, you are considering using threads instead of processes. What would you choose and why?

5) [2 marks] You are tasked with designing code that synchronizes several threads which access shared resources in a large-scale system. Threads which cannot proceed should be placed into a wait queue, rather than allow them to do busy-waiting. Threads cannot proceed into their critical section as long as the values of three global parameters A, B, and C are simultaneously above three predefined thresholds respectively. Decide if semaphores or condition variables are most suitable for your code. Explain *in detail* your decision.

6) [2 marks] You are collaborating with a partner on designing an OS's memory management system. The initial version uses linear page tables but your partner advocates for using multilevel page tables instead. Your partner's argument is that accessing a memory location using linear page tables is typically slower than if the system used multi-level page tables. Explain in detail why you agree (or disagree) with your partner.

7) [2 marks] Consider that you have a hard disk for storing exclusively your Bluray movie collection. Your data is stored in an ext2 file system and the disk is almost full. Given what you know about how data blocks are accessed in an ext2 inode tree, which blocks would be crucial to cache to improve performance of disk accesses.

8) [2 marks] Buffering writes can improve file system performance. What is the downside to buffering writes? Explain the tradeoff, and provide examples of applications where the downside of write buffering may or may not pose a problem.

Q3. [10 marks] Scheduling [15 minutes]

1) [3 marks] Consider the following set of processes which are not doing any I/O. Their arrival times and execution time units are included in the table below. The processes are scheduled on a single core, using an SJF scheduling policy. Draw a diagram to show how the processes get scheduled (similarly to the one we discussed in class) and calculate the average wait time.

Process	Arrival Time	Execution time units
A	0	3
B	1	5
C	2	2
D	3	4
E	4	1
F	5	3

2) [3 marks] Explain how processes could game the MLFQ scheduler, and describe the strategy that the MLFQ scheduler can enforce to prevent processes from gaming the scheduler.

3) [4 marks] The disk scheduling queue contains this sequence of requests for block numbers:
12, 48, 57, 60, 65, 83, 155
The disk head is positioned on block #60 and the current direction is down (towards decreasing block numbers).
At some point, the scheduler finishes scheduling the request for block 12, and then schedules the next block to be fetched, according to the scheduling policy. While this new block is being fetched from disk, a new request for block 5 is added to the queue. You can assume that no further disk block requests come in until all the remaining blocks in the queue are served.
Which disk scheduling algorithm (FCFS, SSTF, SCAN, or LOOK) achieves the smallest seeking time for the scenario described above? Explain your answer by showing your calculations.

Q4. [12 marks] Synchronization [20 minutes]

The code below uses semaphores and mutexes to implement an API for condition variables. The solution is however *not fully correct* in implementing the *exact semantics* of condition variables. Describe the problem(s) clearly, in detail. Add code to fix the problem(s).
You may **not** declare any global variables, you may **not** move or delete existing code, and you may **not** declare pthread_cond_t variables (that would be cheating). You are allowed to add code to the struct mycv or the functions below. The POSIX API for mutexes and semaphores is included on the last page of this exam paper.
Hint: what happens when a signal is sent to a condition variable which has no waiting threads?

```
typedef struct _my_cv {
   sem_t *sem;

} mycv;

void cv_init(mycv *cv) {
   sem_init(cv->sem, 0);

}
void cv_wait(mycv *cv, pthread_mutex_lock *mutex) {

   pthread_mutex_unlock(cv->mutex);

   sem_wait(cv->sem);

   pthread_mutex_lock(cv->mutex);

}
void cv_signal(mycv *cv) {

   sem_post(cv->sem);

}
```

[This page is left intentionally blank. Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Q5. [10 marks] Address Translation with Page Tables [15 minutes]

In this question, we will perform address translations using the physical memory dump on page 25. Feel free to detach page 25-26 to avoid having to flip back and forth. Please DO NOT DETACH ANY OTHER SHEET.

The first translation exercise uses a linear page table, and the second one uses a two-level page table.

1) [1 mark] Warmup.

For a warmup, convert the following hex numbers into binary.

0x badbeef

0x deadface

2) [4 marks] Linear page table translation.

Consider the linear page tables configuration as below:

- PTBR: 14 (i.e., the page table starts at the beginning of page frame #14)
- Page size: 16 bytes
- Virtual address space size: 1024 bytes
- Virtual address format:

[VPN5-0 | Offset3-0] => 6-bit VPN followed by 4-bit offset

- PTE size: 1 byte
- PTE format:

[V | S | D | PFN4-0] => a Valid bit, an Onswap bit, a Dirty bit, followed by a 5-bit PFN.

Find the **data** stored at virtual address **0x0af** by filling in the following table. Fill irrelevant fields with "N/A" (e.g., if it is a page fault).

Virtual Address (in hex):	**0x0af**
Content of PTE (in hex):	
PFN of data (in decimal):	
Offset (in decimal):	
Content of data (1 byte in hex)	

[This page is left intentionally blank. Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

3) [5 marks] Two-level page tables.

Consider the two-level page tables configuration, as below:

- PDBR: 26 (i.e., the page directory starts at the beginning of page frame #26)
- Page size: 16 bytes
- Virtual address space size: 1024 bytes
- Virtual address format:

[PD_INDEX2-0 | PT_INDEX2-0 | Offset3-0] => 6-bit VPN, divided into a 3-bit page directory index, and 3-bit page table index, followed by 4-bit offset.

- PDE and PTE have the same format
- PDE/PTE size: 1 byte
- PDE/PTE format:

[V | S | D | PFN4-0] => the Dirty bit, an Onswap bit, a Valid bit, then a 5 bit PFN.

Find the **data** stored at virtual address **0x2be** by filling in the following table. Fill irrelevant fields with "N/A" (e.g., if it is a page fault).

Virtual Address (in hex):	**0x2be**
Content of PDE (in hex):	
PFN of 2nd level table (in decimal):	
Content of PTE (in hex):	
PFN of data (in decimal):	
Offset (in decimal):	
Content of data (1 byte in hex)	

[This page is left intentionally blank. Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Q6. [12 marks] Memory management and TLBs [15 minutes]

1) [3 marks] A tagged TLB maintains for each entry the process id. Why would a tagged TLB be advantageous? What is the downside to a tagged TLB?

2) [9 marks] Consider the following sequence of memory accesses and their outcomes in a paged memory system. You know that the TLB uses an LRU replacement policy and that the entire memory can hold 3 pages.

```
Access 1. Virtual page 3  => Page fault
Access 2. Virtual page 1  => Page fault
Access 3. Virtual page 2  => Page fault
Access 4. Virtual page 4  => Page fault
Access 5. Virtual page 1  => TLB miss, memory hit
Access 6. Virtual page 2  => TLB miss, memory hit
Access 7. Virtual page 3  => Page fault
Access 8. Virtual page 2  => TLB hit
Access 9. Virtual page 4  => Page fault
```

a) What can you conclude about the size of the TLB? Explain your rationale clearly. Show your calculations.
b) Can you determine the memory's replacement policy? Explain your rationale clearly and show why other policies do not match the outcome of this chain of accesses. You can assume that it cannot be Random.

[This page is left intentionally blank. Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Q7. [16 marks] File systems [25 minutes]

1) [4 marks] Consider that you're implementing an ext2_restore program similar to the one from A4. You are attempting to recover the file /home/bart/file1 and go through the "gaps" in the directory entries of the "bart" directory. You end up finding a removed directory entry with a valid (non-zero) inode and the name "file1", then you go to the inode bitmap and find that the inode is in use. What can you infer from this? Explain **in detail, everything** that you can infer from this.

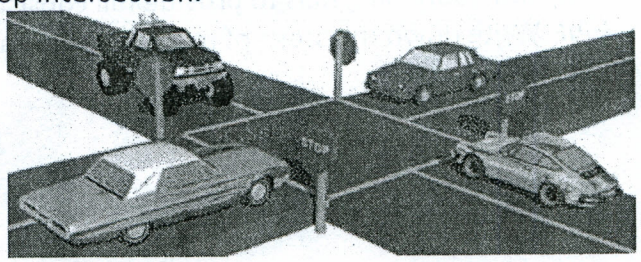
2) [5 marks] The fsck tool typically runs at boot time while the file system is not in active use by any user. User Homer finds that fsck is very slow and does not want to wait for it to run at boot time. Instead, after boot time, he runs fsck manually in a terminal, using a set of flags which enable fsck to run "online" (during normal operation rather than at boot time).
Can any problem occur in this situation? If a problem could occur, explain *in detail* what could happen with this approach. If not, then explain *in detail* why the approach is fine.

3) [5 marks] Metadata journaling provides reasonable but not full consistency guarantees like full (data and metadata) journaling. Explain in detail why this is the case, and describe a scenario where metadata journaling fails to provide full consistency.
Hint: Write down the steps of metadata journaling and compare to the steps of full journaling.

4) [2 marks] Describe a method that SSDs use to prevent wearout of the flash cells.

Q8. [14 marks] Synchronization bugs and deadlocks [20 minutes]

1) [5 marks] The diagram below illustrates a 4-way stop intersection.



The intersection enforces the following rules:

- a car cannot go straight through the intersection if there are any cars waiting on its right.
- a car cannot do a right turn if there are any cars waiting on its left.
- a car cannot do a left turn unless there are no cars waiting in any other lane (other than its own).

a) Explain **in detail** whether there are any problems with this intersection system.
b) If a deadlock is possible, explain why the necessary deadlock conditions hold. If a deadlock is not possible, then explain which condition does not hold.

2) [4 marks] Banker's Algorithm. Consider that your system is currently running 4 resource-intensive processes: P0-P3, each having strict requirements in terms of 3 types of resources: A, B, and C. At this point, the processes have *already been granted* a number of resources of each type:
P0 (1, 1, 2) P1 (1, 2, 0) P2 (1, 0, 1) P3 (0, 1, 1)
For example, process P0 holds 1 unit of resource A, 1 unit of resource B and 2 units of resource C.
The *number of resources still available* (not allocated to any process) in the system is 3 of each type.
The *number of resources necessary* for each of the processes is:
P0 (4, 4, 4) P1 (5, 4, 4) P2 (2, 6, 6) P3 (6, 1, 2)
A process cannot complete without having all its resources met, but once it completes, all its allocated resources are released and can then be used by other processes.
a) Is our resource allocation system currently in a safe state? Explain your answer.
b) If in this state, P3 requests 1 unit of resource C, should this request be granted? Explain in detail.

3) [5 marks] A bank uses the following piece of code to perform a set of bank transfers from a given bank account ("src_acct") to a bunch of other bank accounts. The amounts to be transferred into each destination account are provided in the "amounts" argument. You may assume that the "transfers" function runs in a separate thread for every src_acct. You may assume the no self-transfers will occur (src_acct will not be included in the dst_list). Also, all accounts in dst_list are unique (no duplicates). Analyze the code below and indicate whether there are any potential bugs or synchronization problems. If the code is correctly synchronized, explain *in detail why* there are no possible problems. If it's incorrect, explain *in detail what* problems could arise *and why*.

```
typedef struct {
  long acct_no;
  double balance;
  pthread_mutex_t lock;
} account;

void withdraw(account a, double amount){
  a.balance -= amount;
}

void deposit(account a, double amount){
  a.balance += amount;
}

void transfers(account src_acct, account *dst_list, double *amounts, int howmany){
  int i, j, go_ahead = 0;
  while(!go_ahead) {
    go_ahead = 1;
    pthread_mutex_lock(&(src_acct.lock));
    for(i = 0; i < howmany; i++) {
      if (pthread_mutex_trylock(&(dst_list[i].lock)) == EBUSY) {
        pthread_mutex_unlock(&(src_acct.lock));
        for(j = 0; j < i; j++) {
          pthread_mutex_unlock(&(dst_list[j].lock));
        }
        go_ahead = 0;
        break;
      }
    }
  }
  for(i = 0; i < howmany; i++) {
    src_acct.withdraw(amount[i]);
    dst_list[i].deposit(amount[i]);
    printf("Transfer from %ld to %ld successful", src.acct_no, dst_list[i].acct_no);
  }
  for(i = 0; i < howmany; i++) {
    pthread_mutex_unlock(&(dst_list[i].lock));
  }
  pthread_mutex_unlock(&(src_acct.lock));
}
```

[This page is left intentionally blank. Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Feel free to detach this page, if you find it useful to avoid flipping to the question.]

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
page	0 :	00	ea	00	33	00	1f	7g	00	1e	ff	ae	21	19	77	19	0c
page	1 :	1b	1d	05	05	1d	0b	19	23	3d	00	10	1c	9f	09	00	1f
page	2 :	12	1b	0c	06	00	1e	04	13	0f	0b	12	02	1e	0f	00	0c
page	3 :	7f	32	3a	33	cd	7f	1f	77	3b	3d	ea	96	a4	11	42	7f
page	4 :	0b	04	10	04	05	1c	13	07	1b	13	1d	0e	1b	15	01	07
page	5 :	12	1e	15	7b	08	1b	35	0e	13	0f	1d	1d	1c	ed	12	0f
page	6 :	0b	0f	05	08	ae	00	7b	0c	0d	1e	00	00	00	00	00	00
page	7 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	8 :	11	10	1a	12	0f	10	18	0a	11	15	1e	15	1d	0c	12	17
page	9 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	10 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	11 :	09	10	14	1b	04	01	1f	18	17	0e	15	0c	05	0c	18	18
page	12 :	06	0b	16	19	1c	05	14	1d	01	14	1a	0a	07	12	0d	05
page	13 :	19	10	0b	0e	00	06	14	14	0f	1d	0e	09	1a	b9	12	15
page	14 :	12	18	14	0b	00	0d	1c	0a	07	04	b6	10	02	0c	42	c4
page	15 :	ab	11	c5	0e	0d	1b	08	15	4f	2b	8a	09	09	1c	f9	2b
page	16 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	17 :	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	18 :	7f	7f	7f	7f	7f	7f	ab	7f	7f	7f	8e	7f	7f	7f	dd	7f
page	19 :	00	13	00	01	06	14	02	01	1e	0d	1b	06	0d	0b	05	0a
page	20 :	1a	19	04	02	02	0c	1d	11	08	07	03	04	19	04	1a	19
page	21 :	0b	08	1b	ae	1c	15	1e	12	1e	05	0d	11	1e	11	1a	13
page	22 :	17	13	1d	0a	12	02	11	19	06	08	15	07	08	1d	1e	42
page	23 :	1d	0f	03	0f	0b	0d	1e	1e	11	13	14	0f	0f	09	15	02
page	24 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	25 :	03	03	1c	03	1b	0e	0e	0a	0c	0b	11	0a	19	07	07	0e
page	26 :	09	0e	1d	18	08	d5	15	18	0d	0c	17	0d	07	0e	1d	04
page	27 :	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	28 :	0f	1d	0f	0a	02	11	07	0b	0b	17	07	1d	17	0e	1b	0b
page	29 :	17	08	1e	03	1b	01	07	10	12	0c	03	07	08	17	1c	12
page	30 :	7f	7f	7f	7f	7f	84	7f	7f	7f	7f	97	7f	bd	7f	7f	f4
page	31 :	7f	7f	7f	7f	7f	7f	d0	7f	7f	7f	7f	7f	7f	7f	7f	7f

[Do not use this page for anything other than scratch work. THIS PAGE WILL NOT BE MARKED]

Reference API - the following functions are simplified (compared to POSIX API) for exam purposes:

```
pthread_mutex_lock(pthread_mutex_t *m);
pthread_mutex_unlock(pthread_mutex_t *m);
pthread_mutex_trylock(pthread_mutex_t *m);

pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *m);
pthread_cond_signal(pthread_cond_t *cond);
pthread_cond_broadcast(pthread_cond_t *cond);

sem_init(sem_t *sem, unsigned int initial_value);
sem_wait(sem_t *sem);
sem_post(sem_t *sem);

//barrier will wait for num_threads when pthread_barrier_wait() is called
pthread_barrier_init(pthread_barrier_t *bar, unsigned int num_threads);
pthread_barrier_wait(pthread_barrier_t *bar);
```