

**University of Waterloo
CS350 Midterm Examination
Winter 2019**

Student Name: _____

**Closed Book Exam
No Additional Materials Allowed
The marks for each question add up to a total of 74**

1. (6 total marks)

For the following questions provide an answer and then justify your answer with a single sentence.

a. (2 marks) Efficiency

Which is typically faster and why:

- i Printing the numbers from 1 to 1000000, one number at a time.
- ii Creating a string with the numbers from 1 to 1000000 and printing that string.

Answer:

Justification:

b. (2 marks) Concurrency

Can you still have concurrency if you have a single processor with a single core and the degree of multithreading is one (i.e. $P=1$, $C=1$, $M=1$).

Answer:

Justification:

c. (2 marks) Synchronization

Can a lock be used anywhere a binary semaphore is used?

Answer:

Justification:

2. (10 total marks) For the following questions point form answers are preferred.

a. (2 marks) **Concurrency**

List two possible advantages of concurrency.

b. (4 marks) **Context Switching** There are a number of ways that a context switch can occur.

i Which ones are prevented when interrupts are turned off.

ii Which ones are prevented when each process only has a single thread of execution.

iii Which ones are not prevented by either of the two previous conditions.

c. (4 marks) **cv_wait**

List, in order, the four steps of `cv_wait`. Do not list any of the KASSERTs.

1.

2.

3.

4.

3. (14 total marks) For the following questions a single sentence answer will suffice.

a. (1 mark)

Under what case (or cases) does disabling interrupts enforce mutual exclusion.

b. (1 mark)

Give one disadvantage of a scheduling quantum that is too short (i.e., 1ms or less).

c. (1 mark)

Give one disadvantage of a scheduling quantum that is too long (i.e., 1s or more).

d. (1 mark)

What do exceptions and interrupts have in common?

e. (1 mark)

What would be a scenario (if any) where a kernel stack would have a trapframe pushed and popped without a switchframe also being pushed and popped.

f. (1 mark)

What would be a scenario (in any) where a kernel stack would have a switchframe pushed and popped without a trapframe also being pushed and popped.

g. (1 mark)

Why can't you have a pid of 0 in os/161?

h. (2 marks)

A programmer is writing a program that requires two major (but independent) tasks to be performed and is trying to decide between

- using `fork` and assigning one task to the parent and one to the child or
- using `thread_fork` and assigning one task to each thread.

i What would be one advantage of using `fork`?

ii What would be one advantage of using `thread_fork`?

i. (3 total marks)

Draw the user and kernel stacks for a process that is executing `sys_waitpid`. Show the top of the stack (where items are push on or popped off) at the bottom of the diagram.

j. (2 total marks)

Explain how the following kernel stack could come to be.

trapframe
trapframe

4. (10 marks)

The following pseudocode makes use of a semaphore. Replace the semaphore based implementation with a condition variable based implementation that performs the same task. You may only add up to three additional variables. **Your cv may not be used with a loop.**

```
Semaphore barrier; // initialized to 0

int CS350( void * v, long n )
{
    WriteMidterm()
    V(barrier);
}

int main()
{
    for ( int i = 0; i < NUMTHREADS; i ++ )
        thread_fork( 'student', null, CS350, null, i );

    for ( int i = 0; i < NUMTHREADS; i ++ )
        P( barrier );
    MarkMidterms()
}
```


5. (10 marks)

Consider the following implementation of a lock. Assume that sem is created with an initial count of 1.

```
lock_acquire( lock * lk )
{
    KASSERT( lk != NULL );
    P( lk->sem );
    while ( lk->owner != NULL )
    {
        wchan_lock( lk->wchan );
        V( lk->sem );
        wchan_sleep( lk->wchan );
        P( lk->sem );
    }
    lk->owner = curthread;
    V( lk->sem );
}
```

```
lock_release( lock * lk )
{
    KASSERT( lk != NULL );
    V( lk->sem );
    lk->owner = NULL;
    V( lk->sem );
}
```

- (a) If this lock was used to protect a critical section, would it guarantee mutual exclusion?
- (b) If you answered yes to (a), why? If you answered no to (a), correct the code (you may edit the code in-place).
- (c) If there are any other issues with the lock not related to mutual exclusion, correct them. Otherwise, indicate the implementation is correct.

6. (8 marks)

A system uses segmented address space for its implementation of virtual memory. Suppose a process initially uses 48KB of memory for its heap. The process then runs low on heap space and requests 16 KB more space. Assume you have access to a procedure `sbrk` and that `sbrk(16*1024)` will request that the heap's space be increase by 16 KB by finding a new location in RAM for the heap segment. If successful it returns the new address, otherwise it return NULL.

In roughly 4–6 steps, describe the process that would be required to increase the process's address space.

7. (8 total marks)

A system uses 24-bit virtual addresses, 24-bit physical addresses and memory segmentation. There are four segments and two bits are used for the segment number. The relocation and limit for each of the segments are as follows.

Segment Number	Limit Register	Relocation Register
0x0	0x2 0000	0x40 0000
0x1	0xC 0000	0x70 0000
0x2	0x9 0000	0x50 0000
0x3	0xA 0000	0x60 0000

Translate the following addresses from virtual to physical. Clearly indicate what segment each address belongs to.

a. (2 marks) 0x01 D0D4

Segment Number:

Address Translation:

b. (2 marks) 0x22 CA10

Segment Number:

Address Translation:

c. (2 marks) 0x33 47B8

Segment Number:

Address Translation:

d. (2 marks) 0x1C F008

Segment Number:

Address Translation:

8. (8 total marks)

Ideally any scheme to enforce mutual exclusion should satisfy the following constraints.

- i. Only one thread is allowed in a critical section at a time.
- ii. No assumptions can be made about the order different threads will access the critical section.
- iii. A thread that is outside the critical section cannot prevent another thread from entering the critical section.
- iv. At least one thread should be making progress.
- v. There should be a bound on the time a thread must wait.

For each of four cases listed below there are only two threads, T1 and T2. Rather than use locks, the threads use the code below to provide mutual exclusion to a critical region. Each thread will be trying to access the critical region multiple time (i.e. not just once).

a. (2 marks) T1 and T2 both execute the same function.

```
#define CLOSED 0
#define OPEN 1
volatile int Lock = OPEN; // global variable

AccessCriticalRegion() {
1   while (Lock == CLOSED) {;} // busy wait
2   Lock = CLOSED;
3   CriticalSection();
4   Lock = OPEN;
}
```

Does this scheme meet all the requirements?

If not, which requirement is not satisfied? Give a scenario where that requirement would not be satisfied.

b. (2 marks) Here T1 and T2 use different functions to access the critical region.

```
volatile int Last = 2; // global variable

// Code for T1
T1AccessCriticalRegion() {
1  while (Last == 1){;}
2  CriticalSection();
3  Last = 1;
}

// Code for T2
T2AccessCriticalRegion() {
1  while (Last == 2){;}
2  CriticalSection();
3  Last = 2;
}
```

Does this scheme meet all the requirements?

If not, which requirement is not satisfied? Give a scenario where that requirement would not be satisfied.

c. (2 marks) Here T1 and T2 use different functions to access the critical region.

```
#define WANT_IN 1
volatile int T1 = ! WANT_IN; // global variables
volatile int T2 = ! WANT_IN;

// Code for T1
T1AccessCriticalRegion() {
1  T1 = WANT_IN;
2  while (T2 == WANT_IN){;}
3  CriticalSection();
4  T1 = ! WANT_IN;
}

// Code for T2
T2AccessCriticalRegion() {
1  T2 = WANT_IN;
2  while (T1 == WANT_IN){;}
3  CriticalSection();
4  T2 = ! WANT_IN;
}
```

Does this scheme meet all the requirements?

If not, which requirement is not satisfied? Give a scenario where that requirement would not be satisfied.

d. (2 marks) Here T1 and T2 use different functions to access the critical region.

```
#define WANT_IN 1
volatile int T1 = ! WANT_IN; // global variables
volatile int T2 = ! WANT_IN;
volatile int Last = 1;

// Code for T1
T1AccessCriticalRegion() {
1  while (1) {
2      T1 = WANT_IN;
3      if (T2 == !WANT_IN) {
4          break;
5      }
6      if (Last == 1) {
7          T1 = !WANT_IN;
8          while (Last == 1){;}
9      }
10 }
11 CriticalSection();
12 Last = 1;
13 T1 = !WANT_IN;
    }

// Code for T2
T2AccessCriticalRegion() {
1  while (1) {
2      T2 = WANT_IN;
3      if (T1 == !WANT_IN) {
4          break;
5      }
6      if (Last == 2) {
7          T2 = !WANT_IN;
8          while (Last == 2){;}
9      }
10 }
11 CriticalSection();
12 Last = 2;
13 T2 = !WANT_IN;
    }
```

Does this scheme meet all the requirements?

If not, which requirement is not satisfied? Give a scenario where that requirement would not be satisfied.