Please print in pen:
Waterloo Student ID Number:

WatIAM/Quest Login Userid:

UNIVERSITY OF WATERLOO

Examination
Midterm
Spring 2018
CS 350

Closed Book

Candidates may bring no aids (no calculators).

Times: Thursday 2018-06-28 at 19:00 to 20:50 (7 to 8:50PM)
Duration: 1 hour 50 minutes (110 minutes)
Exam ID: 3810482
Sections: CS 350 LEC 001-004
Instructors: Lesley Ann Istead, Zille Huma Kamal

University of Waterloo
CS350 Midterm Examination

Spring 2018

Student Name: _____

Closed Book Exam
No Additional Materials Allowed

Please initial:

1. (18 total marks)

a. (1 marks)
How is concurrency achieved with a single CPU (i.e., $P(processors) = 1$, $C(cores) = 1$, and $M(multithreading) = 1$)?

b. (4 marks)
List the four reasons that context switches occur.

Please initial:

Please initial:

c. (2 marks)

How does the implementation of a semaphore differ from the implementation of a lock?

d. (2 marks)

A thread is a sequence of intructions. **thread_fork** creates a new thread. What sequence of instructions does the new thread execute and how does it know which sequence to execute?

Please initial:

Please initial:

e. (3 marks)

Under what circumstances is it safe to fully delete a process or zombie process?

f. (2 marks)

getppid returns the PID of the current process's parent. What should **getppid** return if the process is an orphan, that is, has no living parent?

Please initial:

g. (2 marks)
Why are user applications/processes isolated from the kernel?

h. (2 marks)
Give an advantage and disadvantage of dynamic allocation.

2. (9 total marks)
Consider the following pseudocode:

```
int volatile array[10];
Semaphore s = new Semaphore( blah, 0 );

int kernelFunction1( void * data, unsigned long i )
{
    array[i] = -i;
    thread_fork( "x", null, kernelFunction2, null, i );
}

int kernelFunction2( void * data, unsigned long i )
{
    array[i] *= -1;
    V( s );
}

int main()
{
    for ( int i = 0; i < 10; i ++ )
        thread_fork( "x", null, kernelFunction1, null, i );

    for ( int i = 0; i < 10; i ++ )
       P( s );

    int sum = 0;
    for ( int i = 0; i < 10; i ++ )
      sum += array[i];

  // HERE
}
```

a. (1 mark) What is the total number of threads?

b. (6 marks) Which of the following are possible values for array at position HERE? Clearly indicate **Yes** or **No** for each.

- 0000000000
- 0123456789
- 9876543210
- 1234567890
- 9999999999
- 1111111111

Please initial:

c. (1 mark) List all of the possible values of sum at the position HERE.

d. (1 mark) Is there a race condition in this code?

Please initial:

3. (12 total marks)

A process calls **open**, a system call to open a file. While executing **sys_open** there is a timer interrupt causing a context switch. Draw the process user and kernel stacks up to and including the point of calling **switchframe**. Recall that timer interrupts are handled by the **timer_interrupt_handler**.

Please initial:

Please initial:

4. (13 total marks)

A system uses 48-bit physical addresses and 32-bit virtual addresses. The page size is 2^{12} bytes.

a. (1 mark) How many pages of virtual memory are there?

b. (1 mark) How many frames of physical memory are there?

c. (1 mark) How many bits are needed for the page offset?

d. (1 mark) How many bits are needed for the page number, assuming a single-level page table is used?

e. (1 mark) How many bits are needed for the frame number?

Please initial:

f. (4 marks) What is the virtual page number of the following addresses?

0x0000 0000

0x0000 0ACE

0x0110 EA5E

0x0000 00C5

g. (4 marks) Suppose an address space uses the first $0 - (2^{13} - 1)$ bytes of virtual memory. Which of the addresses from (f) are valid?

Please initial:

5. (8 marks)

getpid returns the PID of the current process. **getpidof(procName)** returns the PID of the process with name **procName**. Assume that process names are unique. Assume there exists a global process table, **ProcTable**, that is a linked list of all processes, and a lock **procTableLock** for that table. List the steps, in order, required to implement **sys_getpidof**. If no process with **procName** exists, return **ENOPROC**. Do not give pseudocode.

6. (11 total marks)

No-hold-and-wait is a method to prevent deadlocks in code. To implement this strategy, a new lock function called **try_acquire** is needed. If the lock is available, **try_acquire** acquires the lock and returns true. If the lock is NOT available, **try_acquire** does not acquire the lock or put the thread the to sleep, instead, it immediately returns false.

a. (7 marks) List the steps of try_acquire. Do not call lock_acquire.

Please initial:

Please initial:

b. (4 marks) Consider the following pseudocode.

Note that FuncA and FuncB can be executed concurrently. Is there a deadlock in this code? If no, explain why. If yes, correct the code. If there exists any race condition, fix that also.

```
int total = 0;
int account = 0;
lock mutex = lock_create( "mutex" );

void FuncA(int acc)
{
    lock_acquire( mutex );
        for ( i = 0 to N )
        {
             total ++;
        }
    lock_acquire( mutex );

    account = acc;
}


void FuncB(int acc, int val)
{
    for ( i = 0 to N )
    {
        FuncA( acc );
        total = total - val;
    }
}
```


Please initial:

Please initial:

7. (8 marks)

A system uses a segmented implementation of virtual memory with 32-bit virtual and physical addresses. Each address space has 4 segments.

a. (2 marks) How many bits are used for the segment offset and segment number?

b. (2 marks) For each segment, the MMU has a relocation and limit register. If we wanted to support read-only segments, what changes to the MMU would be required?

c. (2 marks) How many memory accesses does it take to translate virtual address 0xDEADBEEF to a physical address?

d. (2 marks) On a context switch between processes, what changes to the MMU must the kernel make?

Please initial:

e. (BONUS 2 marks)

On a context switch between threads of the same process, what changes to the MMU must the kernel make?

Please initial: