University of Waterloo
CS350 Midterm Examination

Fall 2016

Student Name:	

Closed Book Exam
No Additional Materials Allowed

1. (10 total marks)

Consider a concurrent program that includes two functions, called `funcA` and `funcB`. This program has the following synchronization requirements, *both* of which must be satisfied.

- Requirement 1: At most one thread at a time may be running `funcB`.
- Requirement 2: At most two threads at a time may be running any combination of `funcA` or `funcB`.

	`funcA`	`funcB`
`funcA`	OK	OK
`funcB`	OK	NO

Concurrent Function Execution Requirements

These requirements are summarized in the table on the right, which shows which combinations of `funcA` and `funcB` may be executed concurrently. Note that it is never OK for more than two threads to be running any combination of these functions concurrently.
Your task is to determine how to enforce these synchronization requirements using semaphores. You must not use any other synchronization primitives, e.g., spinlocks, locks, wait queues, condition variables. Your solution should not be more restrictive than necessary, and it should ensure that deadlock is not possible.

a. (2 marks)
List the semaphores that you will use in your solution. For each semaphore, state what its initial value should be.

SemA: initial value 2
SemB: initial value 1

b. (8 marks)
Show the semaphore `P` and `V` operations that threads should perform before and after each call to `funcA` and `funcB` to enforce the synchronization requirements.

```
/* show P,V calls here */


P(SemA);


funcA()
/* show P,V calls here */


V(SemA);
```

```
/* show P,V calls here */

/* order of these is important */
P(SemB);
P(SemA);

funcB()
/* show P,V calls here */

/* order of these is unimportant */
V(SemA);
V(SemB);
```

2. (6 total marks)

Suppose that a concurrent program has k threads, and that each thread is running on its own processor. The threads share access to a global variable, which is protected by a spinlock. To use the variable, each thread will first acquire the spinlock, then access the shared variable, then release the spinlock. Assume that when there is no contention (i.e., when only one thread is trying to access the shared variable), the total time required to acquire the lock, access the shared variable, and release the lock, is 10 time units.

a. (2 marks)
Suppose that each thread accesses the shared variable exactly one time, and that all k threads do so at exactly the same time, which we will refer to as time $t = 0$. At what time will the last of the threads finish releasing the spinlock?

$t = 10k$

b. (2 marks)
For the same scenario described in part (a), what is the total amount of time that the threads will spend spinning? In other words, what is the sum of the threads' spinning times?

total time $= 10\sum_{i=0}^{k-1} i$
total time $= 5(k^2 - k)$ (this closed form is not necessary)

c. (2 marks)
For this part of the question, assume that there are k threads timesharing a single processor. The first thing that each thread does when it is able to run is to acquire the spinlock and access the shared variable. Each thread accesses the shared variable one time. Assume that the scheduling quantum is larger than 10 time units. What is the total amount of time that the threads will spend spinning?

None of the threads will spin (total spinning time is zero).

3. (8 total marks)

Consider the following concurrent program:

```
volatile int numbers[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
volatile int value = 0;

static void myThreadA( void * junk, unsigned long num ) {
    (void)junk;
    numbers[num] = value;
    thread_fork( "B", null, myThreadB, null, num );
    value = value + 1;
}

static void myThreadB( void * junk, unsigned long num ) {
    (void)junk;
    numbers[num] = value;
}

int main() {
    for ( int i = 0; i < 10; i ++ )
        thread_fork( "A", null, myThreadA, null, i );
}
```

a. (2 marks)
Assuming that no errors occur, are the following values for `numbers` possible after all threads have finished executing? For each, answer "Yes" or "No", and give a brief (one sentence) explanation.

`numbers[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }`

Yes. For example, this could occur if every instance of myThreadB runs and exits before any thread running myThreadA executes the line `value = value + 1`.

b. (2 marks)
Repeat part (a), but for the following values for `numbers`:

`numbers[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 12 }`

No. Value starts at 0, and is incremented at most 10 times, so it could never be 12, regardless of thread execution order.

c. (2 marks)
Repeat part (a), but for the following values for `numbers`:

`numbers[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }`

Yes. For example, this could occur if each pair myThreadA/myThreadB executes completely before the next myThreadA executes.

d. (2 marks)
Repeat part (a), but for the following values for `numbers`:

`numbers[10] = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 }`

Yes. For example, this can occurs if threads run in decreasing order, and each threads' myThreadB executes before incrementing value, and the next myThreadA does not run until the current thread has incremented value.

4. (10 total marks)

Suppose that an application program contains a variable `a`, of type `char *`, which is a pointer to an array of characters. The program can then refer to the *i*th element of the array as `a[i]`. Each character occupies one byte, and C arrays are contiguous in the application's virtual memory.

Suppose that the system uses 32-bit virtual and physical addresses and paged virtual memory, with a page size of 4KB (2^{12} bytes). The *valid* entries in the process's page table are shown in the following chart. Assume that the entries for any pages not listed in the chart are invalid.

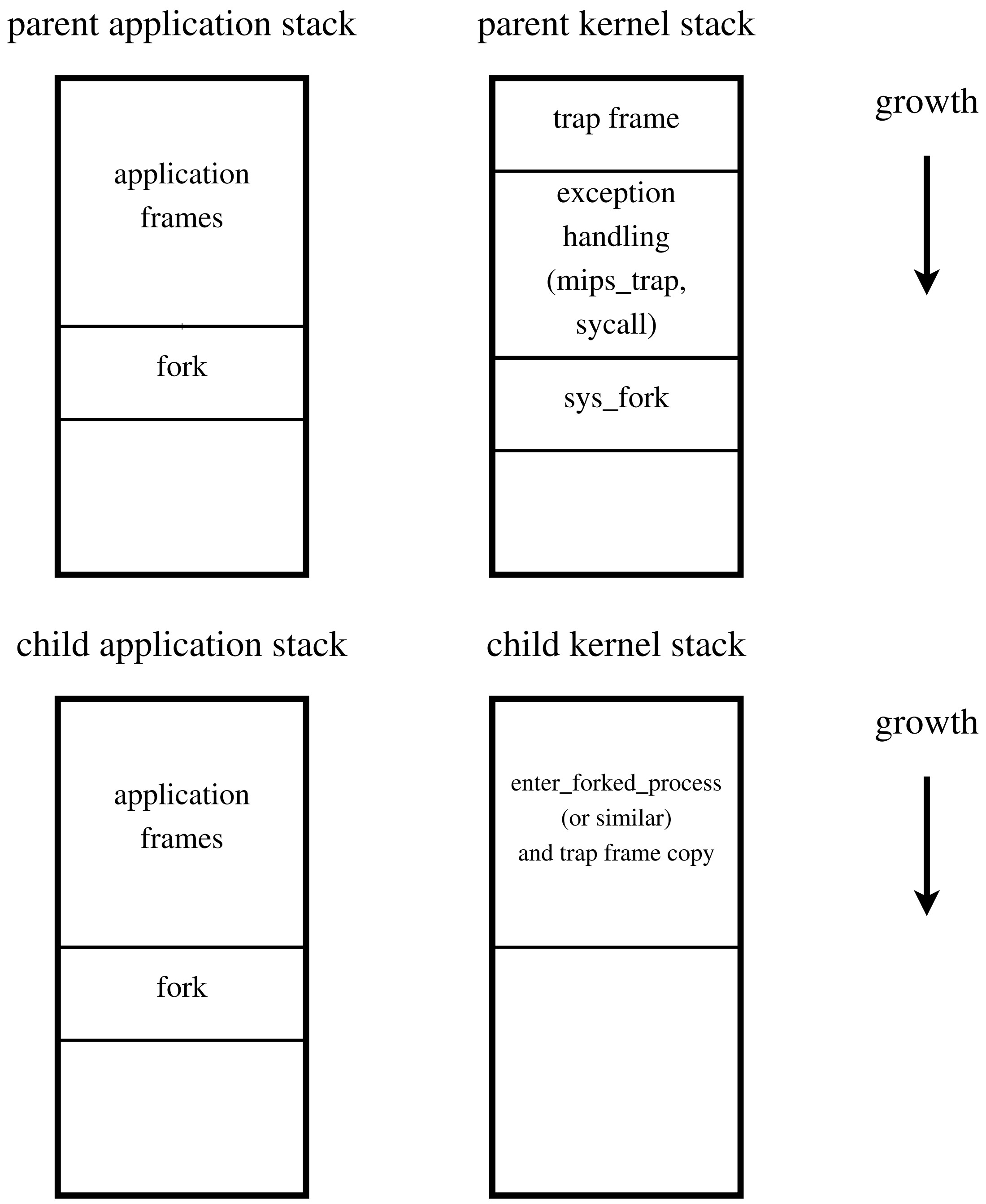
Page #	**Frame #**
0x00010	0x00032
0x00011	0x00033
0x00012	0x00010
0x00040	0x00021
0x00041	0x00022

The following table lists some possible values for the variables `a` and `i`. In each row, indicate what the *physical* address of `a[i]` will be, assuming the values of `a` and `i` indicated in that row, and the page table described above. If the virtual address of `a[i]` cannot be translated, write `"exception"`.

`a`	`i`	**physical address of** `pa[i]`
0x000100F0	0x100	*0x000321F0*
0x00012A00	0x12	*0x00010A12*
0x0001305D	0x2	*exception*
0x00040EF0	0x110	*0x00022000*
0x00041F00	0x100	*exception*

5. (8 total marks)

Draw the relevant stack frames for the application and kernel stacks for an OS161 process in the middle of calling `fork`. Assume that the parent process is in `sys_fork` (the kernel handler function for `fork`), and that the child process has been created and is about to call `mips_usermode`. Draw the stacks of both the parent and child processes.



6. (6 total marks)

a. (3 marks)
On the MIPS, the *load linked* (`ll`) and *store conditional* (`sc`) instructions are used to implement spinlocks. Suppose that two threads, T_1 and T_2, try to acquire an unlocked spinlock at the same time, and that their `ll` and `sc` instructions execute in the following order:

	T_1	T_2
	`ll`	
time		`ll`
$\downarrow$		`sc`
	`sc`	

Which thread(s) will acquire the spinlock after this sequence? Answer one of the following: T_1, T_2, both, neither.

T_2 will acquire the spinlock.

b. (3 marks)
Suppose that the MIPS spinlock was mistakenly implemented using a regular load instruction (`lw`, instead of `ll`) and a regular store instruction (`sw`, instead of `sc`). Suppose that the instruction sequence is the same as in part (a):

	T_1	T_2
	`lw`	
time		`lw`
$\downarrow$		`sw`
	`sw`	

Which thread(s) will believe that they have acquired the spinlock after this sequence? Answer one of the following: T_1, T_2, both, neither.

Both threads will believe they have acquired the spinlock.

7. (6 total marks)

a. (2 marks)
What is the difference between a thread yielding and a thread blocking?

A thread that yields goes from running to ready, it can be immediately scheduled to run again. A thread that is blocked is not running or ready, it is waiting on some resource and cannot be put onto the ready queue or selected for running until that resource is available.

b. (2 marks)
When an exception or interrupt occurs, a trap frame must be created to preserve the application's context. This trap frame is put on a separate kernel stack, instead of the application's stack: why?

Possible answers:

- The stack pointer is an application-owned register, and thus can't be trusted to be pointing at a valid stack.
- Using the application stack would expose kernel data to the application.
- Using the application stack would require the application to budget virtual memory for (unknown) kernel usage.

c. (2 marks)
Both wait channels and condition variables can be used to make threads block. How does a wait channel differ from a condition variable? In particular, how does `wchan_sleep` differ from `cv_wait`?

Condition variables are used in conjunction with an lock, which is passed as a parameter to CV operations. In addition to blocking the calling thread, `cv_wait` ensures that the associated lock is released while the thread is blocked. In contrast, `wchan_sleep` just blocks the calling thread.

8. (6 total marks)

a. (2 marks)
Process P calls the `fork` syscall and creates process C. Process P exits before process C exits. Assume that the kernel does not allow a process to call `waitpid` on any process except its children. Are any of following statements definitely true at the time that P exits? Circle any that are true.

- Process P's PID can be safely re-used by the kernel. **(Not True)**
- Process C inherits process P's PID. **(Not True)**
- Process C terminates automatically. **(Not True)**
- Process P will not be allowed to exit until C exits. **(Not True)**

b. (4 marks)
Consider a virtual memory system 64-bit virtual addresses, and a page size of 32KB (2^{15} bytes). The system uses multi-level paging. Each page table holds at most 2^{13} entries, and each page table directory holds at most 2^{12} entries. In the worst case, how many memory accesses are required to translate a virtual address to a physical address?

This will be 4-level tree - 3 levels of directories, plus a level of page tables. The number of memory accesses required in the worst case will be 4.