# University of Waterloo
# CS350 Midterm Examination

## Fall 2015

**Student Name:** _____

**Closed Book Exam**
**No Additional Materials Allowed**

**1. (12 total marks)**

| Global Variables | Initialization | Function `func1` | Function `func2` |
|---|---|---|---|
| `struct semaphore *sa;`<br>`struct semaphore *sb;`<br>`struct semaphore *sc;` | `sa = sem_create("A",1);`<br>`sb = sem_create("B",1);`<br>`sc = sem_create("C",0);` | `void func1() {`<br>`P(sa);`<br>`funcA();`<br>`V(sa);`<br>`P(sc);`<br>`}` | `void func2() {`<br>`P(sb);`<br>`funcB();`<br>`V(sb);`<br>`V(sc);`<br>`}` |

Consider a concurrent program, parts of which are shown above. Suppose that the initialization code is executed one time, by the program's initial thread. The initial thread then creates many new threads, some of which run `func1`, and some of which run `func2`. `func1` and `func2` call `funcA` and `funcB`, which are not shown.

In the space below, re-implement `func1` and `func2`, without using semaphores. Instead, use `locks` and `condition variables` for synchronization. Your re-implemented functions must have the same behavior as original functions shown above. You may create as many locks, condition variables, and other global variables as you need. Be sure to show global variable declarations and initialization, as well as the implementations of `func1` and `func2`, as in code shown above.

**2. (10 total marks)**

Consider a single-level paging system with 12-bit virtual addresses, 24-bit physical addresses, and a 256 ($2^8$) byte page size.

**a. (2 marks)**

What is the maximum number of entries in a page table in this system?

**b. (4 marks)**

A process $P_1$ has the following page table. Frame numbers are given in hexadecimal notation (recall that each hexadecimal digit represents 4 bits).

| Page Number | Frame Number |
|---|---|
| 0 | 0x1010 |
| 1 | 0x2034 |
| 2 | 0x43AC |
| 3 | 0x1100 |
| 4 | 0xAC11 |
| 5 | 0x8000 |

For each of the following *physical* addresses, indicate the virtual address to which it maps. If the physical address is not part of the physical memory assigned to $P_1$, write NO TRANSLATION instead. Use hexadecimal notation for the virtual addresses.

- 0x1100A0
- 0xAC1100
- 0xBA3424
- 0x43ACA0

**c. (2 marks)**

Due to a bug in the OS161 `as_copy` function, the following is the page table of $P_1$'s child process immediately after it returns from `fork`. Mark the entries in the page table that you are certain to be incorrect.

| Page Number | Frame Number |
|---|---|
| 0 | 0x2453 |
| 1 | 0x1010 |
| 2 | 0xEA35 |
| 3 | 0x3100 |
| 4 | 0x2034 |
| 5 | 0x9012 |

**d. (2 marks)**

Name one advantage and one disadvantage of having a virtual address space that is smaller than the physical address space.

**3. (8 total marks)**

```
void bump() {
  int y;
  x++; // x is a global variable
  y=x;
  kprintf("%d",y); // print value of y
}
```

Consider the function shown above. Assume that `x` is a volatile integer global variable, and that its value is initialized to `0` before any calls to `bump`. Suppose that the `bump` function is part of a concurrent program has uses $k$ concurrent threads, and that each thread calls `bump` one time. Assume that calls to `kprintf` are atomic, i.e., internally, `kprintf` uses a lock to ensure that threads will print one at a time.

**a. (6 marks)**

Suppose that this concurrent program is running on a machine with one single-core processor, and that $k = 4$. Which of the following outputs are possible for this program? Write "yes" next to each output which is possible, and "no" next to each output which is not possible. Note that you must get at least half of these correct to receive any credit for your answer.

- 1234
- 4321
- 0123
- 2222
- 4444
- 1235
- 012
- 1124

**b. (2 marks)**

Suppose instead that the concurrent program (with $k = 4$) runs on a machine with two single-core processors. Do your answers from part (a) change? If not, write "No Change". If so, indicate one output string from part (a) for which you would give a different answer in this situation.

**4. (8 total marks)**

| Virtual Address | Physical Address |
|---|---|
| 0x0008 0AF0 | 0x0010 1AF0 |
| 0x0000 2224 | 0x0008 3224 |
| 0x0007 3234 | 0x0018 3234 |

Suppose that while a process $P$ is running, it uses the virtual addresses shown in the left column of the table above. For each of these virtual addresses, the corresponding physical address (after address translation) is also shown in the table. On the machine on which $P$ is running, both virtual addresses and physical addresses are 32 bits long.

**a. (3 marks)**

Given the virtual-to-physical address translations shown in the table, is it possible that the MMU used *dynamic relocation* to translate $P$'s virtual addresses to physical addresses? If so, write "YES" and indicate the value that must be in the MMU's relocation register while $P$ is running. If not, write "NO" and explain, briefly and clearly, how you know that dynamic relocation was not used.

**b. (3 marks)**

Is it possible that the MMU used *paging*, with a page size of 64 KB ($2^{16}$ bytes), to translate $P$'s virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

**c. (2 marks)**

Is it possible that the MMU used paging, with a page size of 4 KB ($2^{12}$ bytes), to translate $P$'s virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

**5. (10 total marks)**

**a. (3 marks)**

Is it possible for a thread's kernel stack to contain more than one trap frame? If yes, write "YES" and identify - clearly and briefly - a situation in which this could occur. If not, write "NO".

**b. (2 marks)**

Is it possible that a call to OS/161's `wchan_sleep` function will cause a thread context switch? Answer "YES" or "NO" and briefly explain your answer. (Answers without a clear explanation will receive no credit.)

**c. (2 marks)**

Each page table entry normally includes a *valid* bit. Explain, briefly and clearly, the purpose of a valid bit. What happens if a process attempts to access a virtual address on a page that is mapped by an *invalid* page table entry, i.e., one for which the valid bit is not set. Again, explain briefly and clearly.

**d. (3 marks)**

When a system call occurs in OS/161, how does the kernel know *which* system call has been requested? Explain briefly and clearly.

**6. (10 total marks)**

**a. (3 marks)**

List the different transitions between the 3 thread states. Why can't a thread go from a blocked state directly to a running state?

**b. (2 marks)**

Explain the difference between a trapframe and a switchframe. What generates a trapframe? What generates a switchframe?

**c. (3 marks)**

Describe a scenario (list of steps) in which having `spinlock_release(&sem->sem_lock)` before `wchan_lock(sem->sem_wchan)` in the semaphore P() implementation can cause a concurrency problem.

**d. (2 marks)**

What information can be found in each TLB entry?

**7. (12 total marks)**

You have been hired by Snowflake entertainment to design the matchmaking system for their new multiplayer online game. In this game, a match consists of 3 players and can only start when all 3 players are available. The company owns only one server and the server can only host one match at a time. A new match can start on the server only when (a) the previous match has finished, and (b) three players are available to play. Implement the following three functions to satisfy the specified constraints. Global variables can be defined in the provided space.

```
#define PLAYERS_PER_MATCH 3
// Define your global variables here.
```

```
// Called only once before any players have arrived.
void game_sync_init()
```

```
// Called only once when the company takes down the system for maintenance.
void game_sync_cleanup()
```

Question continues on the next page.

```
// Called once by each player, before that player starts a match
// Should block until the player's match can start (according to
// Snowflake's synchronization requirements)
void before_match()
```

```
// Called once for each player, after that player's match is finished
void after_match()
```