UNIVERSITY OF WATERLOO
CS 350 MIDTERM :: SPRING 2013

Date: Wednesday, June 19, 2013
Time: 7:00 – 9:00 pm
Instructor: Dave Tompkins
Exam type: closed book
Additional materials allowed: none

Last Name: ______________________________

First Name: ______________________________

Student #: __ __ __ __ __ __ __ __

UW Login: __ __ __ __ __ __ __ __

Signature: ______________________________

INSTRUCTIONS

1. Before you begin, make certain that you have one exam booklet with 14 pages (double sided)
2. All solutions must be placed in this booklet.
3. If you need to make an assumption to answer a question, state your assumption clearly.
4. When writing code, you should use C or C-like pseudocode. You do not have to worry about `#include` statements or semi-colons.
5. If you need more space, use the last page, and indicate that you have done so in the original question.
6. A big gap after a question does not necessarily mean that a long answer is expected.
7. Did you see in the marking guide there's a bonus question? woo-hoo! Make sure you answer it at the end.
8. Relax! Read these instructions as often as needed.

Question	Marks Given	Out Of	Marker's Initials
1		12	
2		6	
3		8	
4		7	
5		9	
Bonus			
Total		42	

Question 1 [13 * 1 = 12 Marks]

(a) Explain why registers k0 & k1 cannot be used (even temporarily) by gcc in OS/161.

(b) Explain why there are more registers stored in a trap frame than in a thread context.

(c) True or false: If there are no global variables, then no locks are necessary. Briefly justify your answer.

(d) Give one advantage and one disadvantage of having a software design with high lock granularity (many locks).

(e) Briefly explain what this line of code is doing and why:

```
mips_syscall(struct trapframe *tf) {
  ...
  tf->tf_v0 = retval;       <---- explain this line
```

(f) Briefly describe why the C `stdio` library binary is not portable between different operating systems, even on the same hardware (machine architecture).

(g) Explain the primary difference (as discussed in class) between Hoare semantics and the Mesa semantics used in OS/161.

(h) A system uses a dynamic relocation virtual address scheme. The virtual addresses are 16 bits long, but the relocation register is 32 bits long. What is the maximum amount of physical memory that can be made available to each process? How much physical RAM can the entire system support?

(i) What is the difference between internal and external memory fragmentation.

(j) Explain why dumbvm is more like dynamic relocation than paging.

(k) Give one advantage and one disadvantage of having a small quantum.

(l) Explain the significance of the return value of `fork()`.

Question 2 [6 Marks]

Given the following MIPS 64-bit TLB entry specification (with 4k page sizes)

```
VPAGE  bits  44-63
PFRAME bits  12-31
DIRTY  bit   10
VALID  bit    9
```

and the following TLB entries: (Most Significant Bit on the left)

```
0x 0000 0000 0000 6600
0x 0000 1000 0000 2200
0x 0012 3000 4564 5600
0x 0040 0000 0040 0400
0x 1000 0000 1000 0600
```

a) For each virtual address below, give the corresponding physical address. If it cannot be determined or a fault would occur reading the address, write "FAULT".

```
0x0000 0006

0x1000 0001

0x0012 3456

0x4564 5645

0x0000 1234

0x0040 0040

0x8012 3456
```

b) For each physical address, provide the corresponding virtual address. If it cannot be determined, write "UNKNOWN".

```
0x0000 0006

0x1000 0001

0x4564 5645

0x0040 0040

0x8012 3456
```

Question 3 [4+4 = 8 Marks]

(a) [4 Marks] Give a **proof** as to why *resource ordering* can prevent deadlocks. It can be informal, but it should be sound. You are not required to reference the deadlock detection algorithm, but you may reference it if you choose.

(b) [4 Marks] Here is Peterson's algorithm as presented in class:

```
volatile boolean flag[2]; // initially false
volatile int turn;
// for thread A: i = 0 & j = 1, thread B: i = 1 & j = 0

flag[i] = true;
turn = j;
while (flag[j] && turn == j) { }
//critical section
flag[i] = false;
```

Your friend has implemented Peterson's algorithm for OS/161 as follows:
(He used a thread id `tid` to identify each thread: `tid` has the value of either 0 or 1)

```
turn = 1 - tid;
flag[tid] = 1;
while (turn != tid && flag[1 - tid]) { }
//critical section
flag[tid] = 0;
```

Describe how the critical section is protected (or not protected) in this implementation. Justify your answer.

Question 4 [2+1+4 = 7 Marks]

(a) [2 Marks] *Concisely* explain how **<u>in your</u>** A1 cat/mouse solution the decision was made to switch from allowing one animal to eat (ie: cats) to the other animal eating (ie: mice). If you did not complete assignment 1, describe the naïve solution discussed in class.

Consider where there are **c** cats, **m** mice, and **b** bowls, running on at least (c+m) processors so that all threads are executing concurrently. Animals never die, the eating time **t** for both animals is the same, and there is no "sleep time" between eating (animals can immediately eat again) (note: "sleeping" is different than blocking due to synchronization). Ignore any overhead required to execute code or switch contexts.

We will measure *efficiency* as the *average* fraction of bowls in use over an extended period of time.

(b) [1 Mark] Given the above specifications and that (c >> b) and (m >> b), describe any circumstances under which your solution described in a) would achieve its maximum efficiency, and then calculate that efficiency as a formula using the variables c, m, b and t as necessary.

(c) [4 Marks] Given the above specifications, consider the following solution:

"Allow **k** mice to eat, then allow **k** cats to eat, and then allow **k** mice to eat, and so on…"

Determine the efficiency for each scenario below.

For each scenario, give the *maximum wait time* for both cats and mice. The wait time is the amount of time that elapses from when cat X finishes eating, and then cat X starts to eat again. Assume fairness amongst the animals of the same type: once cat X eats, all other cats will eat exactly once before cat X eats again.

b	c	m	t	k	Efficiency	Maximum Wait Time	
						Cats	Mice
10	5	5	1	5			
10	10	5	1	5			
10	50	25	5	25			
10	11	9	10	10			

Question 5 [9 Marks]

For this problem, you are required to use **semaphores** to simulate a school cafeteria.

- There are an arbitrary number of students. Each student is a separate thread.
- There are only student threads. There is no "coordinating" or "dispatching" thread.
- There is an arbitrary number of student threads.
- There are K stations in the cafeteria, numbered 0..K-1 (where K is a global constant)
- Only one student can occupy a station at a time.
- Students may not cut in line or skip a station. They must maintain their original ordering (sequence) and must start at station 0 and finish at station K-1.
- There may be more students than stations, so you must also maintain a queue of students waiting to get to the first station. The following **unsynchronized** list functions (similar to the linked lists shown in class) should be sufficient. These functions use a (hidden) global variable to store the list structure.
  ```
  int is_empty();
  struct student *list_peek_front();
  struct student *list_remove_front();
  void list_append(struct student *s);
  ```
- Each student may need more than one food "item" at a station. The number of items student `s` needs to acquire at station `i` is `s->items[i]`, which could be zero.
- For each item at station `i` the student must call:
  `get_item(struct student *s, int i);`
  When the function returns, the student will have the item, but it may block during the function call to wait for the food.
- You must ensure that `get_item` is synchronized and is never called more than once at the same time for either the same student `s` or the same station `i`.
- You may use `thread_sleep` and `thread_wakeup` if you wish.

`List your global variables and semaphores here.`
`Indicate the initial value of each variable & semaphone.`

```
// you can add to struct student here if you wish
struct student {
     int items[K];



}

void student_simulation(struct student *me, ...) {
  /* get in line */


















  /* completed station K-1 */

}
```

Bonus Question [1 Mark]

Please answer the following 3 questions ***<u>honestly</u>*** at the end of the exam:

This exam was too long:	This exam was too hard:	This exam was fair:
a) Strongly disagree	a) Strongly disagree	a) Strongly disagree
b) Disagree	b) Disagree	b) Disagree
c) Neutral	c) Neutral	c) Neutral
d) Agree	d) Agree	d) Agree
e) Strongly agree	e) Strongly agree	e) Strongly agree

Draw a picture that illustrates a thread switching from user mode to privileged execution (kernel) mode. Humourous illustrations encouraged.

***** END OF EXAM *****

This page has been left (mostly) blank intentionally. Use this space if necessary to complete your answers. Make sure you note on the original page that more of your solution can be found here. Do NOT remove this page from your exam.

This page has been left (mostly) blank intentionally.