UNIVERSITY OF WATERLOO
CS 350 MIDTERM :: SPRING 2012

Date: Monday, June 25, 2012
Time: 7:00 – 8:50 pm
Instructor: Dave Tompkins
Exam Type: Closed book
Additional Materials Allowed: none

Last Name: ______________________________

First Name: ______________________________

Student #: __ __ __ __ __ __ __ __

UW Login: __ __ __ __ __ __ __ __

Signature: ______________________________

Question	**Marks Given**	**Out Of**	**Marker's Initials**
1		10	
2		15	
3		5	
4		16	
5		12	
6		4	
7		18	
8		20	
Bonus			
Total		100	

INSTRUCTIONS

1. Before you begin, make certain that you have one exam booklet with 8 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 this instruction as often as needed.

Question 1 [10 Marks]

(a) [3 Marks] In OS/161 there is a `struct thread` to represent a thread context and a `struct trapframe` to represent a trap frame. Describe something that is contained in both structures, and then for each of the structures describe something that it contains that the other does not. Briefly explain why each of the 3 things you describe appears where it does.

(b) [2 Marks] On the MIPS + OS/161 system, explain why `a++` is not considered an atomic operation, yet `V(s)` is.

(c) [3 Marks] What behaviour does a lock have that a binary semaphore does not? Briefly describe a situation where you would prefer a lock and a situation where you would prefer a binary semaphore.

(d) [2 Marks] Explain what this line of code is doing and why:

```
mips_syscall(struct trapframe *tf) {
  ...
  tf->tf_epc += 4;       <---- explain this line
  ...
}
```

Question 2 [15 Marks]

(a) [4 Marks] There are three different ways a thread can transition from user mode to kernel mode. Write a small user program for OS/161 that ensures that all three would occur during its execution and identify how each would occur. If you cannot ensure a transition will occur, explain why.

(b) [5 Marks] In class, 5 different sections of an ELF file were described:

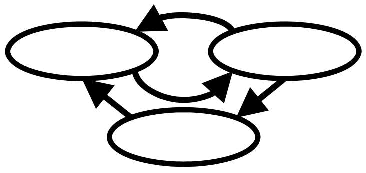
```
text  rodata  data  bss  sbss
```

Write a small user program for OS/161 that would have elements in each of the 5 section types and identify where each appears in your program.
(pro tip: you can simply have variable names that use the appropriate section name).

(c) [6 Marks] Write a single user program for OS/161 that will generate one child process, which in turn will generate a grandchild process (3 processes in total). The output should be the sum of the three `pid`s. For simplicity, assume that `pid_t` is an `int`.
The following functions might be helpful:

```
int   printf(const char *format, ...);
pid_t getpid(void);
pid_t fork(void);
pid_t waitpid(pid_t pid, int *status, int options); // opt=0
void  _exit(int exitcode);
```

Question 3 [5 Marks]



(a) Re-draw the thread state transition diagram shown in class, and label each of the 3 states and all 4 transitions.

(b) In OS/161 there is a fourth state: S_ZOMB. Add this state to your diagram and label the new transition(s).

Question 4 [16 Marks]

You are given the first 25 pages of an execution string for a process:

A B C D E B D B C D B C D B C B D E B D E B C B A

There are 5 unique pages, but it is to be run on a system that has only 4 frames of physical memory allocated to the process (and there will always be 4). None of the pages are resident at the beginning of the execution (ie: all frames are uninitialized).

(a) [8 Marks] For each of the following page replacement algorithms, state which page(s) would NOT be in resident memory at the end of the execution. If you cannot answer, state your reason (eg: "not enough information" or " a tie between X and Y")

FIFO (First In, First Out):

OPT (Theoretically Optimal):

LFU (Least Frequently Used):

LRU (Least Recently Used):

(b) [3 Marks] For the following two algorithms, how many page hits would there be?

OPT:

LRU:

(c) [2 Marks] What is WS(25,4)? (In other words, what is the working set (WS) at the end of the execution if $\Delta = 4$)?

(c) [3 Marks] At the beginning of the execution none of the pages were resident. Explain what policy the O/S would likely have in place and why that is often a good policy.

Question 5 [12 Marks]

For all parts of this question, you should assume a virtual memory system based on simple paging. All parts of this question refer to the following two page tables, one for process P1 and one for process P2. Note that the frame numbers are specified in hexadecimal, as are all virtual and physical addresses used in this question.

P1

Page #	Frame #
0	0x8d10
1	0x1004
2	0x3008
3	0x5500
4	0x2220
5	0x2221
6	0x2222
7	0x222a
8	0x5558

P2

Page #	Frame #
0	0x222b
1	0x010a
2	0x010b
3	0x3008
4	0x3001
5	0x222c

(a) [4 marks] For each of the following virtual addresses from P1's virtual address space, indicate the physical address to which it corresponds. For the purpose of this part of the question, assume that the page size is 4096 (2^{12}) bytes. Give your answers in hexadecimal. If the specified virtual address is not part of the virtual address space of P1, write "NO TRANSLATION" instead.

`0x0000022`

`0x00005ff`

`0x0001004`

`0x0006072`

(b) [4 marks] Repeat part(a), but this time for P2 and under the assumption that the page size is 256 (2^8) bytes.

`0x0000022`

`0x00003a8`

`0x00005ff`

`0x0001004`

(c) [4 marks] For each of the following physical addresses, indicate which process's virtual address space maps to that physical address, and indicate which specific virtual address maps there. If you cannot answer the question, explain why not. For the purposes of this question, assume that the page size is 4096 (2^{12}) bytes.

`0x010abcd`

`0x2222ffa`

`0x222d002`

`0x3008888`

Question 6 [4 Marks]

The following function has a critical section that should only be accessed by one thread at a time:

```
int myFunction () {
  struct semaphore *s = sem_create("my semaphore", 0);
  /* ... */
   V(s);
  /* ... critical section ... */
   P(s);
  /* ... */
}
```

Is this a good design? Briefly justify your answer

Question 7 [18 Marks]

(a) [3 Marks] The cat & mouse simulation in assignment 1 was run entirely in the kernel. Explain why it could not be run as a user process.

(b) [15 Marks] How would you modify OS/161 to enable users to run multi-threaded applications that can use semaphores for synchronization?

Notes: Use point form. You must provide sufficient functionality so that an application such as the cat & mouse simulation could be implemented (with semaphores). There are many different strategies available to solve this problem. You may assume that assignment 2 has been completed. You do not have to provide any code, just describe your changes and new features and briefly motivate them.

Question 7 (more space)

Question 8 [20 Marks]

In this question we are running a modified cat & mouse simulation that can accommodate an arbitrary number of animal types, and an arbitrary number of each type of animal.

Once they are created, animals repeat the following steps until they die: they wait to eat, they eat, and then they nap. Animals are continuously being created (it's the circle of life). Animals eat at bowls. There are a fixed number of bowls; each bowl can be occupied by at most one animal at a time, and all of the animals eating at the same time must be of the same type (otherwise there is chaos).

You must implement the function:

```
void bowlfree() {...} // called whenever bowls are available
```

You should try to implement the most **efficient** solution you can and keep *as many bowls occupied as possible*. You should **not** worry about starvation or fairness.

You may assume the following global variables and functions are available:

```
const int numbowls      // total number of bowls
const int N             // number of animal types
volatile int numfree    // number of currently free bowls
volatile int curtype    // type of animal currently eating

struct animal {...}     // data structure for an animal

void assign(*a)         // Assign animal a to a free bowl

struct queue {...}      // FIFO queue for managing animals
void qadd(*q, *a)       // Add animal a to queue q
animal* qnext(*q)       // Get & remove the next animal from the queue q
int qsize(*q)           // Return the number of animals in q

queue *waiting[]        // waiting queues for each type of animal
int maxtype()           // animal type with the largest waiting[] queue
```

- You do not have to worry about initializing any global variables you create.
- The animal "types" are integers 0..(N-1). (eg: 0=mice, 1=cats, 2=dogs, etc.)
- Assume all animals behave the same: they eat & nap for the same duration.
- When `bowlfree()` is called you can assume there is at least one bowl free and at least one animal waiting to eat. The function `bowlfree()` may be called by multiple threads simultaneously. Do not concern yourself over how or when `bowlfree()` is called or the mechanics of `assign()`.
- You do not have to track *which* bowls are free or update `numfree`. If there is a free bowl `assign()` will work properly and update `numfree` for you. If `assign()` is called and there are no free bowls, or there is a different type of animal currently eating, the system will panic (crash).
- The queues and queue functions are not synchronized. You should use a synchronization mechanism for each queue. You do not have to worry about how the `waiting[]` queues are populated and can assume they are populated using your synchronization mechanism.

(a) [15 marks]

```
// YOUR GLOBAL VARIABLES GO HERE





void bowlfree() {   // at least one bowl is free









     // this is some sample code: you can use it if you wish
     while ((numfree > 0) && (qsize(waiting[curtype]) > 0)) {
       assign(qnext(waiting[curtype]));
     }



}
```

(b) [5 marks] Justify why your solution is efficient, and describe any sacrifices in fairness you made to achieve that efficiency.

Bonus Question [1 Mark]

Please answer the following 3 questions ***honestly*** 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 of an operating system thrashing

***** 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.