University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Fall 2016 Anthony D. Joseph

Midterm Exam #1

September 28, 2016
CS162 Operating Systems

Your Name:	
SID AND 162 Login:	
TA Name:	
Discussion Section Time:	

General Information:
This is a **closed book and one 2-sided handwritten note** examination. You have 80 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	21	
2	11	
3	19	
4	22	
5	27	
TOTAL	100	

NAME: ________________________________

1. (21 points total) True/False and Why?

   a. (8 points) True/False and Why? **CIRCLE YOUR ANSWER.**

      i) When a user program performs a system call, it must first put the system call arguments onto the kernel stack and then run a special instruction to switch to kernel mode.

      TRUE FALSE

      Why?

      ii) On a system with a single processor core without hyperthreading, you can speed up a single-threaded program by utilizing multiple threads.

      TRUE FALSE

      Why?

      iii) When a user program successfully calls execv(), its process is replaced with a new program. Everything about the old process is destroyed, including all of the CPU registers, program counter, stack, heap, background threads, file descriptors, and virtual address space.

      TRUE FALSE

      Why?

NAME: ____

iv) The only way a user thread can transfer control to kernel mode is through syscalls, exceptions, and interrupts.

TRUE FALSE

Why?

b. (13 points) Short answer.

i) (4 points) Briefly, in two to three sentences, explain how an Operating System kernel uses dual mode operation to prevent user programs from overwriting kernel data structures.

ii) (3 points) Briefly, in two to three sentences, explain the purpose of the `while` loop in the context of Mesa-style condition variables.

NAME: ____

iii) (4 points) Briefly, in two to three sentences, explain the problems with the Therac-25 – be specific but brief in your answer.

iv) (2 points) In a typical OS, one way that a thread can transition from the 'running' state to the 'waiting/blocked' state is to wait on acquiring a lock. Other than calling sleep or using semaphores or monitors, what are **TWO** other different *types* of ways that a thread can transition from the 'running' state to the 'waiting/blocked' state?

NAME: ____________________________________

2. (11 points total) C Programming.

Consider the following C program:

```
int thread_count = 0;

void *thread_start(void *arg) {
   thread_count++;
   if (thread_count == 3) {
       char *argv[] = {"/bin/ls", NULL};
       execv(*argv, argv);
   }
   printf("Thread: %d\n", thread_count);
   return NULL;
}

int main(int argc, char *argv[]) {
   int i;
   for (i = 0; i < 10; i++) {
       pthread_t *thread = malloc(sizeof(pthread_t));
       pthread_create(thread, NULL, &thread_start, NULL);
       pthread_join(*thread, NULL);
   }
   return 0;
}
```

a. (6 points) When you run the program, what will be the output (assume that all system calls succed)?

b. (5 points) If we removed the line `pthread_join(*thread, NULL)`, could the output change? If so, explicitly describe the way(s) it might differ.

NAME: ____

3. (19 points total) Finite Synchronized Queue using Monitors.

In lecture we implemented an Infinite Synchronized Queue using a monitor. In this exam question you will implement a Finite Synchronized Queue (FSQ) using a monitor. The FSQ has Last-In, First-Out semantics (like a stack), a fixed maximum `capacity` and supports three operations: `CreateQueue()`, `AddToQueue()`, and `RemoveFromQueue()`.

a. (6 points) What are the correctness constraints for the Finite Synchronized Queue?

1.

2.

3.

Each element of the FSQ is a `node`:

```
typedef struct {
   int data;
   node *next;
} Node;
```

b. (3 points) Modify the `FSQ struct` to support a Finite Synchronized Queue.

You may assume that you have `struct` types for the components of a monitor.

```
typedef struct {
   Node *head;           /* pointer to head of queue */
   int length;           /* number of nodes in queue */
   int capacity;         /* max size of queue */







} FSQ;
```

We have implemented the `CreateQueue(int size)` function for you. *You may assume that the `head`, `Length`, `capacity`, and monitor components are initialized.*

NAME: ____

c. (5 points) Implement the `AddToQueue()` function. *Feel free to use pseudocode for monitor operations.*

```
void AddToQueue(FSQ *queue, Node *data) {











}
```

d. (5 points) Implement the `RemoveFromQueue()` function. *Feel free to use pseudocode for monitor operations.*

```
Node *RemoveFromQueue(FSQ *queue) {











}
```

NAME: ____

4. (22 points total) PintOS questions

a. (3 points) What is the maximum number of threads you can create in a user process in PintOS?

b. (4 points) How does PintOS preempt the currently running thread in order to schedule a new thread to run, if the currently running thread never yields?

c. (3 points) What is the purpose of the idle thread in PintOS? *Note that simply saying "the idle thread is the thread that runs when the system is idle" is not an acceptable answer.*

d. (4 points) Briefly explain what happens when a timer interrupt occurs while interrupts are disabled in PintOS.

NAME: ____

`e.` (4 points) How does `struct thread *switch_threads (struct thread *cur, struct thread *next)` work for a thread that has just been created?

f. (4 points) Consider the following code for the thread_exit() function in PintOS:

```
void thread_exit (void) {
   ASSERT (!intr_context ());

#ifdef USERPROG
   process_exit ();
#endif

    intr_disable ();
    list_remove (&thread_current()->allelem);
    thread_current ()->status = THREAD_DYING;
    schedule ();
    NOT_REACHED ();
}
```

Briefly explain what `NOT_REACHED ()` means and why this happens.

NAME: ____

5. (27 points total) Santa Claus problem (from *Operating Systems: Internals and Design Principles*)

Santa Claus sleeps in his shop and can only be woken up by either:
(1) all ten reindeer being back from their vacation or
(2) some of the elves having difficulty making toys.
The elves can only wake Santa up when 5 of them have a problem. While Santa is helping the 5 elves with their problems, any other elf who needs to meet Santa must wait till the other 5 elves have been tended to first.

If Santa wakes up to 5 elves and all ten reindeer, Santa decides that the elves must wait and focuses on getting his sled ready. The last (tenth) reindeer to arrive must get Santa while the others wait before being harnessed to the sled. *You can assume that there are only ten reindeer, but you cannot make assumptions on the number of elves.*

Assume Santa can call these methods: `prepareSleigh()`, `helpElves()`
Reindeer call `getHitched()` and Elves call `getHelp()` – both `getHitched()` and `getHelp()` are thread safe, and can be called outside of critical sections.

You MUST implement the problem using semaphores.

a. (6 points) What are the correctness constraints?

1.

2.

3.

b. (6 points) Initialize the global variables:

NAME: ____

c. (5 points) Implement the `Santa()` function:

```
Santa() {












}
```

d. (5 points) Implement the `ReindeerComesHome()` function:

```
ReindeerComesHome() {












}
```

NAME: ________________________________

e. (5 points) Implement the `ElfRequestsHelp()` function:

```
ElfRequestsHelp() {









}
```