Graduate Operating Systems
Spring 2015
Final Exam

Total time: 120 minutes; Total points: 100

Name: (please print)_____

In recognition of and in the spirit of University of South Florida Honor Code, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature: _____

This examination is closed books and notes. No electronic and digital devices are allowed. You may not collaborate in any manner on this exam.

As with any exam, you should read through the questions first and start with those that you are most comfortable with. Answers will be graded for substance and clarity. Redundancy and vague generalities are considered bad form. Partial credit will be offered only for meaningful progress towards solving the problems.

1	
2	
3	
4	
5	
6	
7	
Total	

User/Kernel separation and Processes

1) Answer the following multiple-choice questions. Circle all answers that apply. For these questions all 21 statements count equally.

A. In a typical modern OS, which of the following statements are true about user applications and the kernel?

1. The kernel runs at superviser privilege level.

2. Applications run at superviser privilege level.

3. Applications may directly invoke any function calls in the kernel.

4. The kernel frequently relinquishes control and must depend on the processor to allow it to regain control.

B. Which of the following instructions should be protected, i.e., can execute only when the processor is running at superviser level?

1. MOV cr3, src; (move the value in src to control register CR3. CR3 contains the physical address of the base of the page table).

2. TRAP; (jump to kernel's syscall dispatcher with kernel privilege)

3. ADD; (add numbers)

4. JMP; (jump to a different instruction)

5. INB (contact I/O device).

C. Which of the following are elements of a typical process descriptor? (Recall that a process descriptor or process control block is the per-process state kept by the kernel.)

1. Process state (blocked/runnable);

2. Address space;

3. Disk driver;

4. File descriptor table.

D. Which of the following statements are true about processes?

1. Any process can access any other process's memory by default.

2. Timer interrupts are generally required for a kernel to correctly isolate processes.

3. The kernel must run one process to completion before it starts another process.

4. A user-level process can crash the kernel if not programmed carefully.

E. Which of the following statements are true when the kernel switches execution from the current process to another process?

1. The kernel must save the register values (including IP (Instruction Pointer), SP (stack pointer) etc.) of the current process to memory.

2. The kernel must save the content of the current process' memory on disk.

3. The kernel must close all the files opened by the current process.

4. The kernel must make the page table of the next process active.

2) [5 points] Is it possible for a CPU scheduler with a 100-millisecond time slice to spend over half its time in the OS context switch code? Assume that it takes the OS 1 millisecond to context switch the CPU. Justify your answer.

Concurrency:

3) [15 points, 5 each] Assume you are building a special operating system that requires executing the expression:

$$z = F3(F1(x),F2(F3(y)));$$

where x, y, and z are integers and F1, F2, F3 are functions.

The functions F1() and F3() must execute on thread T1 while the function F2 needs to execute on thread T2.

a. Your job is to write the code to force the expression to be evaluated regardless of the order they are run by the CPU scheduler. Note you will have to add code to all three functions below.

```
int z, x, y; //shared variables

//Declare shared variables and semaphore with initial values here.
```

`void ComputeZ() {`
 	
`StartThread(T1);`
 	
`StartThread(T2);`
 	
`return;`
`}`
	`void T1() {`
	
	`}`
		`void T2() {`
		
		`}`

b. Does your solution handle the following cases? If so, explain why. If not, explain the problem and how you could fix it.

i. Would your solution handle the case when ComputeZ() is called twice in a row like:

```
x=...;  y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z);
x=...;  y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z);
```

ii. Would your solution handle the case when ComputeZ() is called twice from two different threads? For example:

```
void Z1() {
x=...; y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z); }

void Z2() {
x=...; y=...; //Set x and y arguments
ComputeZ();
printf("z = %d\n", z); }

StartThread(Z1);
StartThread)(Z2);
```

Memory management:

4) True or False?

a. In a multiprogramming system the main memory is not generally occupied simultaneously by multiple processes.

b. A hardware mechanism is needed for translating virtual addresses to physical main memory addresses at the time of execution of the instruction that contains the reference.

c. A physical address is the location of a word relative to the beginning of the program and the processor translates that into a virtual address.

d. All segments of all processes must be of the same length.

e. It is impossible to have both paging and segmentation in the same system.

Miscellanea:

5) True or False?

a. The sharing of main memory among processes is useful to permit efficient and close interaction among processes because such sharing does not lead to any problems.

b. It is possible for one process to lock the mutex and for another process to unlock it.

c. An unsafe state is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock.

d. Deadlock avoidance is more restrictive than deadlock prevention.

File Systems:

6) [12 points, 3 each] This question examines classic file systems and how they might need to change in order to adapt to new technologies.

   (a) Classic file systems contain many policies that are tuned for hard disk drives. Consider the Berkeley Fast File System (FFS); what policies does FFS contain that are specific to hard drives?

   (b) New computers are now being sold with flash-based drives instead of conventional rotating disks. Flash has this following performance property: excellent random I/O performance for reads (nearly the same as sequential). Assume that you run FFS on top of this raw flash device. What will the impact on read performance be? Just considering read performance, is FFS well-suited to flash?

   (c) Flash also has this performance property: when writing to a page (assume a page size of 4KB), one must first erase an entire block (assume a block size of 128KB). Assume that you run FFS on top of this raw flash device. What will the impact on write performance be? Just considering write performance, is FFS well-suited to flash?

   (d) Flash also has an unusual reliability property: when a particular block is erased "too many" (i.e., 10,000 or 100,000) times, it will "wear out" and thus become unreadable. Assume you run FFS on top of this raw flash. What will the impact on reliability be? Just considering reliability, is FFS well-suited to flash?

Distributed File Systems:

7) [12 points, 3 each] This question is about NFS version 2, the classic stateless distributed file systems protocol from Sun.

(a) An application on a client issues a write to a file. That write is buffered in client memory for some time. Why do NFS clients do this? What are the advantages and disadvantages?

(b) The client at some point decides to issue the write to the server. When does the client usually do this? Why?

(c) The write reaches the server, and the server writes it to disk immediately, before acknowledging the request. Why does the server do this? What if the server acknowledged the request and then wrote the data to disk?

(d) The acknowledgement from the server gets lost, and the client resends the write protocol request. What happens in this case? (and why does it work?) What could the server do to improve performance in this situation?