CS140 Operating Systems and Systems Programming
Midterm Exam

October 28th, 2005

(Total time = 50 minutes, Total Points = 50)

Name: (please print)_____

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

Signature:_____

This examination is close notes and close book. You may not collaborate in any manner on this exam. You have 50 minutes to complete the exam. Before starting, please check to make sure that you have all 7 pages.

1	
2	
3	
4	
5	
6	
Total	

Name:_____

1. (6 points total) Virtual Memory

(a) Your Pintos partner told you they had implemented an 8-bit clock algorithm for the HW3 virtual memory assignment. Unfortunately they left town before writing up their part of the design document. What advantage would you attribute to this implementation over a single bit clock algorithm when writing up the design document for the project?

(b) In addition, your partner also claimed to have implemented a local replacement policy rather than a global replacement policy. Does this make any sense with an 8-bit clock algorithm? If it does, write what you would do to justify this implementation. If it does not make sense, describe why not.

2. (8 points total) Working set

(a) Assume you switch a program from using static linking for access library routines to using dynamically linked libraries. Would you expect the working set of a process to increase or decrease or stay the same? Justify your answer.

(b) Describe why when viewing the working set of a program over time the working set may show peaks and plateaus even though the program may only be using a fixed number of pages at any given point in its execution.

3. (8 points) Linking
Using a technique called reverse engineering it is possible to take a program's object file and determine implementation details like the algorithms use or possibly even the source code. Using reverse engineering can involve both looking in the object file as well as observing the program being executed. Would reverse engineering be easier or harder if a program switched from static linking of libraries to dynamic linking of libraries? Be sure to justify your answer.

4. (9 points) Virtual Memory

Assume you are working with a Pintos-like operating system that maps itself into part of every process. A new requirement appears that the OS support processes that use the entire virtual address space (4GB). You also discover that the x86 hardware system call mechanism support the ability to switch address spaces (e.g. assign a new value to the CR3 register that points to the root of the active page table) whenever you enter or leave the kernel. Describe how you would implement the kernel code that accesses user space such as system call arguments and setting up the process initial stack now that the operating system no longer shares the address space with the application.

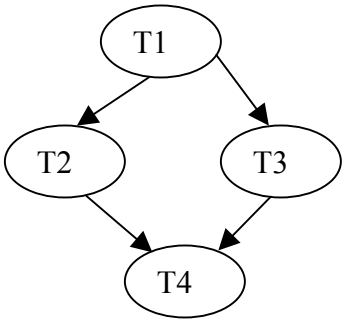
5. (9 points total) CPU Scheduling

(a) Describe how priority donation is implemented in a CPU scheduler using lottery scheduling.

(b) Frequently CPU schedulers will operate on two time scales. A long-term scheduler will select which jobs should be run-able while a short-term scheduler selects when these jobs actually gets a CPU. Give a reason why a scheduler would partition jobs like this.

(c) In a Unix like CPU scheduler (as described in class), would you expect a thrashing process to have a high priority or a low priority? Justify your answer.

6. (10 points) Synchronization
Assume you are given a graph that represents the relationship between four threads (T1, T2, T3, T4). An arrow from one thread (Tx) to another (Ty) means that thread Tx must finish its computation before Ty starts. Assume that the threads can arrive in any order. Use semaphores to enforce this relationship specified by the graph. Be sure to show the initial values and the locations of the semaphore operations.




```
// Semaphores definitions and their initial values
```

```
void T1(void)
{

// T1 computation

}
```

```
void T2(void)
{

// T2 computation

}
```

```
void T3(void)
{

// T3 computation

}
```

```
void T4(void)
{

// T4 computation

}
```