Homework 5

Never due, but look at these before exam time

You need not hand these in, but time will be set aside in class on October 17 to discuss these questions and any other topic from the first half of the course. You are strongly encougared to look at these before our next class meeting.

1. Tanenbaum, Question 15, p. 187.
2. Tanenbaum, Question 16, p. 187.
3. Tanenbaum, Question 20, p. 187.
4. Tanenbaum, Question 1, p. 263.
5. Tanenbaum, Question 5, p. 264.
6. Tanenbaum, Question 8, p. 264.
7. Consider a logical-address space of eight pages of 1024 words each, mapped into a physical memory of 32 frames. How many bits are in the logical address? How many bits are in the physical address?

Questions from last year's midterm exam:

1. Miscellaneous. (10 points)

a. What signal number is sent by the Unix command `kill -STOP`? Or, more simply, what did I give in class as the answer to the first question on the exam? (2 points)

b. Modern PC operating systems include complexities formerly used only by large mainframe operating systems. Why? (4 points)

c. Why is the separation of mechanism and policy important feature of an operating system? (4 points)

2. CPU Scheduling (25 points).

The following table contains arrival times and CPU burst times for four processes.

process	arrival time	CPU burst time
P1	0	9
P2	2	2
P3	4	6
P4	7	3

First, assume the scheduling discipline is *First Come First Served* (FCFS).

a. Draw a Gantt chart showing the process occupying the CPU at each time using FCFS. Label each context switch with its time (assume context switch time is zero). (2 points)

b. What are the total waiting times and turnaround times for each process? (2 points)

c. What is the mean system throughput over this time? (2 points)

For parts d, e, and f, assume the scheduling discipline is *Round Robin* (RR), with a time quantum of 3 units. Assume the same process arrival table:

process	arrival time	CPU burst time
P1	0	9
P2	2	2
P3	4	6
P4	7	3

d. Draw a Gantt chart showing the process occupying the CPU at each time using RR scheduling. Label each context switch with its time (assume context switch time is zero). (2 points)

e. What are the total waiting times and turnaround times for each process? (2 points)

f. What is the mean system throughput over this time? (2 points)

g. Name two reasons why the Shortest Job First CPU scheduling algorithm can be problematic. (4 points)

h. In a loaded system using a simple Round Robin scheduling algorithm, what is the tendency of CPU utilization as the time quantum decreases? (4 points)

i. Suppose that a short-term CPU scheduling algorithm favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs? (5 points)

3. Resource Allocation. (15 points)

Let R1, R2, and R3 be three different non-preemptible resource types. Assume maximum claims and current resource assignments for four processes P1, P2, P3, and P4, are given in the following table, along with the total number of each resource (including allocated resources).

	Maximum Claims				Current Use				
Resource	P1	P2	P3	P4	P1	P2	P3	P4	Total
R1	3	2	6	1	1	1	3	0	8
R2	2	1	4	3	0	1	0	3	6
R3	5	3	5	2	2	0	4	2	8

a. How many units of each resource are currently free? (1 point)

b. Is this system currently in a safe state? If it is, give a safe sequence. If it isn't, what processes may end up in deadlock? (4 points)

c. If P3 requests 3 instances of R1, should the request be granted? Why or why not? (4 points)

d. Consider a system consisting of four resources of the same type shared by three processes, each of which needs at most two resources. Explain why this system is deadlock-free. (6 points)

4. Memory Management (20 points)

a. Consider a paging system with the page table stored in memory. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take? (3 points)

b. If we add TLBs to the system from part (a), and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? Assume that finding a page table entry in the TLBs takes zero time, if the entry is there. (4 points)

c. Consider the following page-reference string:

1, 1, 1, 2, 6, 3, 4, 2, 3, 2, 4, 5, 1, 1, 1, 2, 3, 1, 2, 2, 2, 1, 2, 4, 5, 2

Using the working-set model, with working-set window, Δ=6, what are the largest and smallest working sets encountered, and which pages are in those sets? Only begin looking for the smallest set after a full window of 6 pages has been processed. (4 points) [Our exam will not include the working-set model, but it will be a candidate for the final!]

d. Consider a system which uses a demand paging memory scheme. Explain why small page sizes necessitate keeping the page table in memory rather than in hardware registers. (5 points)

e. Which memory management scheme is more likely to suffer from internal fragmentation, segmentation or paging? Which is more likely to suffer from external fragmentation? Explain briefly. (4 points)

5. Process Synchronization (32 points)

a. *The Dining Savages.* A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot. The behavior of the savages and cook is defined by the following processes:

$Savage_i$: while (1) { get serving from pot; eat; }
$Cook$: while (1) { sleep; put M servings in pot; }

Develop psuedocode for the actions of the savages and the cook. Use semaphores for synchronization. You may assume the semaphores are fair. Your solution should avoid deadlock and awaken the cook only when the pot is empty. (15 points)

b. Can semaphores which are implemented without specific operating system support function correctly and still avoid all busy waiting? Why or why not? (4 points)

c. Why can multiple readers be allowed concurrent access to the shared database in the Readers-Writers problem? (3 points)

e. The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2];  /* initialized to false */
int turn;
```

The structure of process P_i (`i` == 0 or 1), with P_j (`j` == 1 or 0) being the other process is given by:

```
while(1) {
  flag[i] = true;
  while (flag[j]) {
    if (turn == j) {
      flag[i] = false;
      while (turn == j);
      flag[i] = true;
    }
  }
  /* Critical Section */
  turn = j;
  flag[i] = false;
  /* Non-critical Section */
}
```

Explain how this algorithm solves the critical-section problem. (10 points)