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

Spring 2003 Anthony D. Joseph

Midterm Exam *Solutions*

March 13, 2003
CS162 Operating Systems

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

General Information:
This is a **closed book and notes** examination. You have two hours to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points given to the question; there are 100 points in all. 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!!

Problem	**Possible**	**Score**
1	28	
2	21	
3	12	
4	27	
5	12	
Total	**100**	

1. (28 points total) Short answer questions:
   a. (9 points) List any THREE major components of most modern operating systems (e.g., Unix , Solaris, WindowsNT, Windows2000, or WindowsXP) and briefly describe their role of each.

      i)
      3 points for each, 1 points for name and 2 points for role. Memory management, process management, file management, I/O system management, networking, scheduling, etc. No credit was given for OS functions (e.g., government).

      ii)

      iii)

   b. (9 points) Give a definition of a counting semaphore, and list and describe the valid operations.

      3 points for definition, 2 points for each operation. A counting semaphore is a synchronization data structure that can be used to control or limit the number of processes that can access to a critical region. There are three operations that are allowed on a semaphore:
      1. Setting the initial value of the semaphore (number of concurrent accesses allowed).
      2. P() decrements the semaphore's counter and either causes the process to wait until the resource is available or allocates the process the resource.
      3. V() increments the semaphore's counter, releasing a waiting process (if any is waiting).

c. (4 points) List the conditions for deadlock.
1 point for each. Limited access, circular chain of requests, no preemption, and multiple independent requests (or "hold and wait").

d. (6 points) Provide definitions for internal and external fragmentation. Draw a picture show internal fragmentation in a pure paging system (not paged segmentation or segmented paging, but only paging). Draw a picture showing external fragmentation in a pure segmentation system.

2 points for each definition, 1 point for each picture. Internal fragmentation: space inside allocated memory that is wasted, typically occurs in paging systems. External fragmentation: space outside of allocated memory that is too small to be used for another process, typically occurs in segmented systems.

2. (21 points total) Processor Scheduling. Here is a table of processes and their associated running times. *All of the processes arrive in numerical order at time 0.*

Process ID	**CPU Running Time**
Process 1	6
Process 2	1
Process 3	2
Process 4	4
Process 5	3

a. (9 points) Show the scheduling order for these processes under First-In-First-Out (FIFO), Shortest-Job First (SJF), and Round-Robin (RR) scheduling with a timeslice quantum = 1 time unit.

Time	**FIFO**	**SJF**	**RR**
0	*1*	*2*	*1*
1	*1*	*3*	*2*
2	*1*	*3*	*3*
3	*1*	*5*	*4*
4	*1*	*5*	*5*
5	*1*	*5*	*1*
6	*2*	*4*	*3*
7	*3*	*4*	*4*
8	*3*	*4*	*5*
9	*4*	*4*	*1*
10	*4*	*1*	*4*
11	*4*	*1*	*5*
12	*4*	*1*	*1*
13	*5*	*1*	*4*
14	*5*	*1*	*1*
15	*5*	*1*	*1*

b. (12 points) For each process in each schedule above, indicate the queue wait time and turnaround time (TRT).

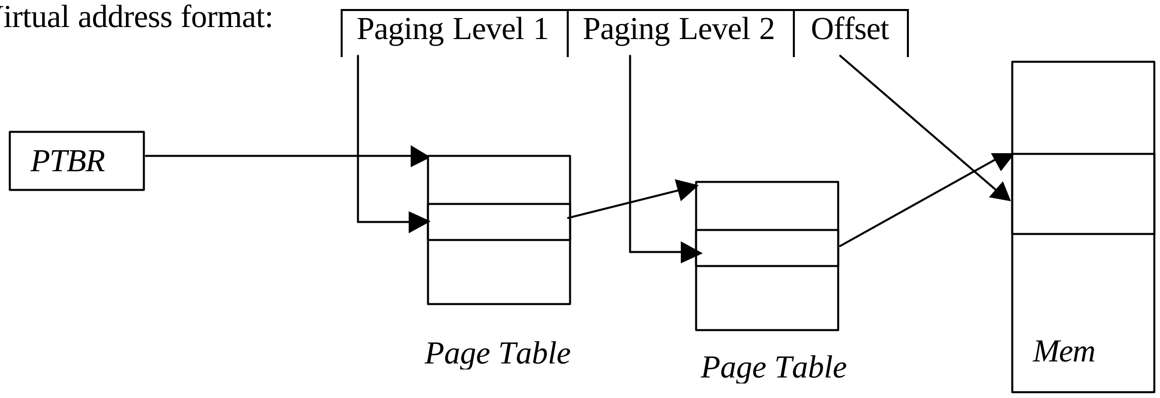
Scheduler	Process 1	Process 2	Process 3	Process 4	Process 5
FIFO queue wait	*0*	*6*	*7*	*9*	*13*
FIFO TRT	*6*	*7*	*9*	*13*	*16*
SJF queue wait	*10*	*0*	*1*	*6*	*3*
SJF TRT	*16*	*1*	*3*	*10*	*6*
RR queue wait	*10*	*1*	*5*	*10*	*9*
RR TRT	*16*	*2*	*7*	*14*	*12*

The queue wait time is the *total* time a thread spends in the wait queue.

Part a) 3 points per column, part b) 2 points per row.

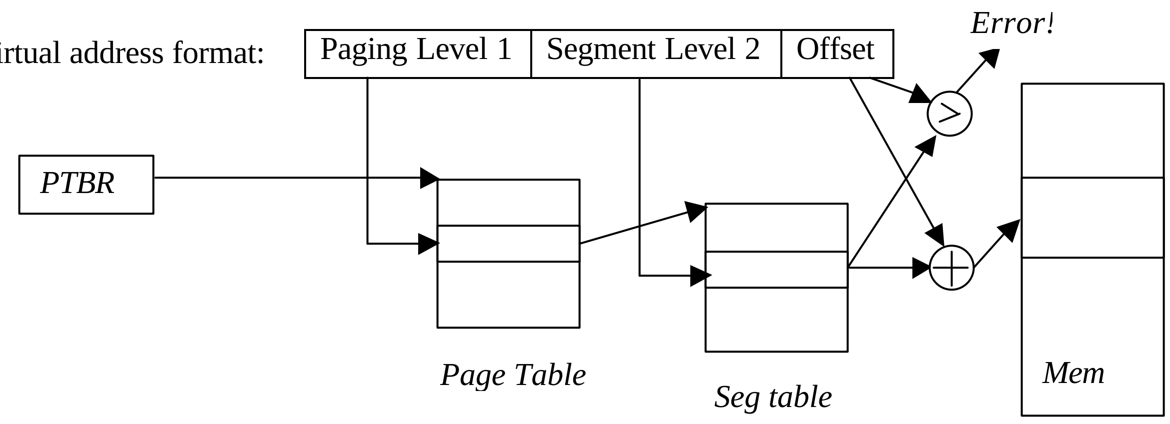
3. (12 points total) Two-level Virtual Memory. For each of the following two-level virtual memory addressing schemes, explain, *in one or two sentences*, how the scheme works.

a. Virtual address format:

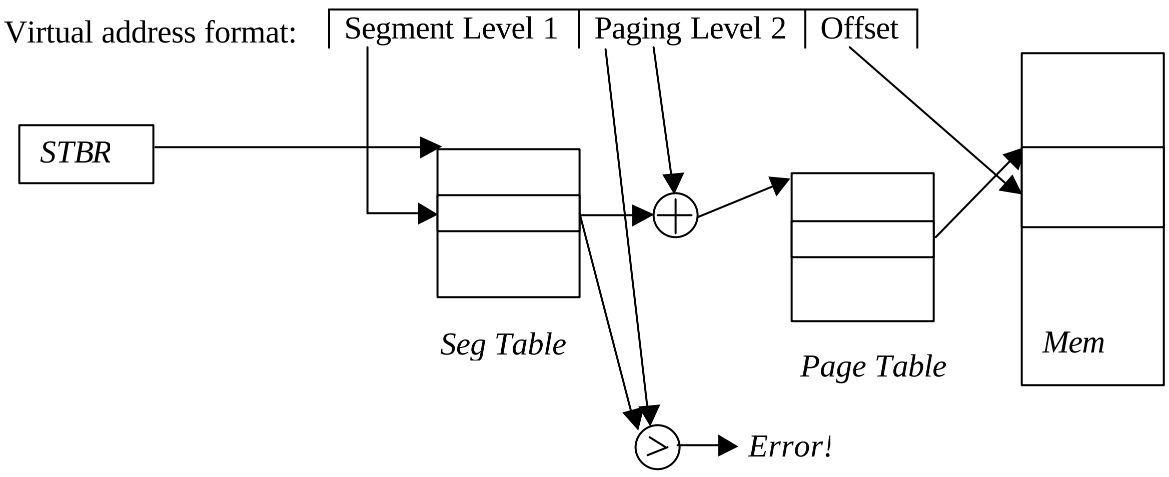


3 points for each. 1 point for basic concept, -1 for minor errors, -1 for answers that are too long. For top-level paging schemes, -1 point for no mention of Page Table Base Register.

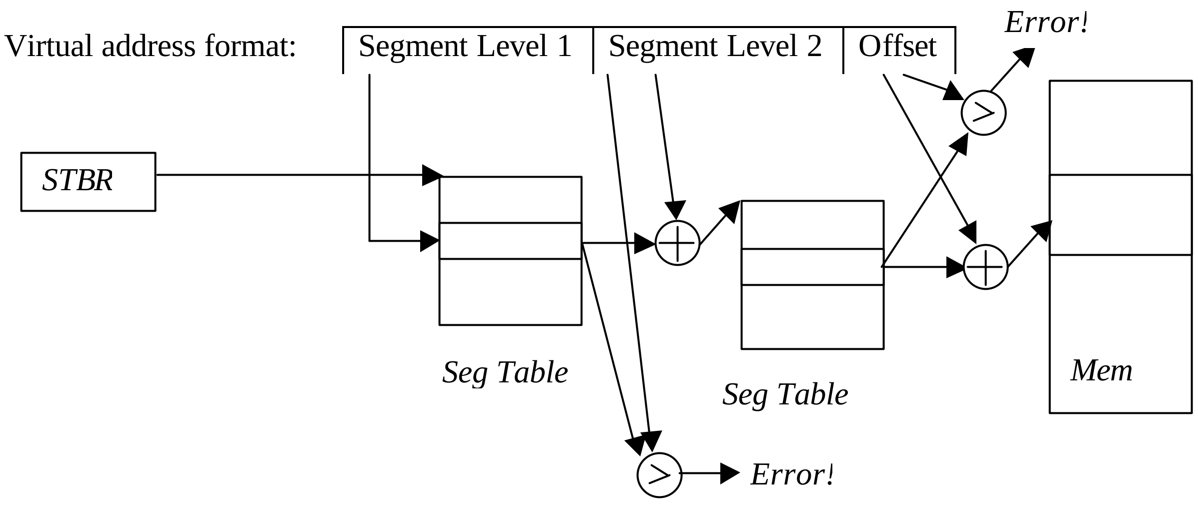
b. Virtual address format:



c. Virtual address format:



d. Virtual address format:



4. (27 points total) Two-Level Page-based Virtual Addressing. Consider a 32-bit machine with a multi-level virtual memory system with 32-bit pointers and 4096-byte pages that supports two-levels of page tables. All Page Table Entries (PTEs) are 4 bytes.

a. (6 points) Show the *complete* format of a virtual address.

Each address is broken up into three parts:

Page Level 1 – 10 bits	*Page Level 2 – 10 bits*	*Offset – 12 bits*

1 point for each field, 1 point for correct bit length of each field.

This is a multi-level scheme using two sets of page tables for ***each*** *process: an "outer" one and an "inner" one. PL 1 is an index into the outer page table, containing PTE's that contain pointers to the inner page table (which can be paged). In other words, the inner page table is broken up into pages that do not have to be stored contiguously.*

The PTE at outer [PL 1] contains a pointer to the PL 1th page in the inner page table. PL 2 is an index into the PL 1th page in the inner page table. PTE's are 4 bytes, so there are 1,024 PTE's per page. Thus, PL 2 must be exactly 10 bits. The index at PL 2 points to the page that holds virtual addresses starting with PL 1 PL 2. The particular byte is the dth byte in that page. Since pages are 4K bytes, d must be exactly 12 bits. Since d is 12 bits and PL 2 is 10 bits, PL 1 must be 10 bits.

b. (6 points) Explain the steps the hardware takes in translating a virtual address to a physical address for this scheme (do not worry about supporting paging to disk).

See Problem 3a for the picture of the steps that are taken. The PTBR is used to locate the outer page table ***in physical memory****. PL 1 is used to index into the outer page table. As above, the PTE at this index points to a page in the inner page table. PL 2 is used to index in this inner page table, and the PTE in the inner page table points to the physical page. Offset is a location on the physical page. If you didn't mention the PTBR, we subtracted one point.*

Note that each PTE should have a valid bit and several protection bits – e.g., read, write, execute, valid. The memory operation (load, store, load for execution) must agree with these bits in both sets of PTE's (inner and outer). Otherwise the hardware will generate an interrupt. If you didn't mention the role of the valid bit or the protection bits, we subtracted one point for each missing role.

If you missed a level, we subtracted one point for each level missed. If you included an incorrect step, we subtracted one point.

c. (3 points) How many memory operations are required to read or write a single 32-bit word?

Without extra hardware, performing a memory operation takes 3 actual memory operations: two page table lookups in addition to the desired memory operation. We did not award partial credit for this problem.

d. (4 points) List the fields of a Page Table Entry (PTE).

Each PTE will have a pointer to the proper page (two points) plus several bits – read, write, execute, (1 point for protection bits), and valid (one point). This information can all fit into 4 bytes, since if physical memory is 2^{32} bytes, then 20 bits will be needed to point to the proper page, leaving ample space (12 bits) for the information bits.

If you didn't mention the valid and protection bits, then you lost 2 points. If you added incorrect fields, we subtracted one point for each incorrect field.

e. (6 points) How much physical memory is needed for a process with one page of virtual memory?

Three pages are needed: one for the outer page table, one for one page of the inner page table, and one for the process' single page.

*Note that the inner page table does not need 4M (2^{10} * 4K), because the outer page table enables you to only have inner page table pages for those pages that are part of the process's virtual address space.*

For partially correct answers, we subtracted three points for the wrong total and one point for each incorrect page level. Answers that stated that only enough memory was needed for a PTE at the outer and inner levels (instead of an entire page), were penalized at least three points.

f. (2 points) What happens *in the virtual memory subsystem* on a context switch?
On a context switch, the PTBR of the new process must be loaded.
Note that it is not necessary to save the PTBR of the outgoing process as that does not change on a context switch, as it is already stored in the process' control block. We did not give partial credit for answers that only saved the PTBR.

No Credit – Problem X: (000000000000 points)

The Overwhelming Might of the US Military

The following is the transcript of an actual radio conversation that took place in October 1995, off the coast of England. The British Ministry of Defence recently released the transcript:

British: Calling unknown radar contact at position *****, please divert your course 15° to the south to avoid a collision.

Americans: Recommend you divert your course 15° to the north to avoid a collision.

British: Negative. You will have to divert your course 15° to the south to avoid a collision.

Americans: This is the Captain of a US Navy ship. I say again, divert your course north.

British: Negative, I say again. You will have to divert your course.

Americans: This is the captain of the aircraft carrier USS Lincoln, the second largest ship in the United States' Atlantic Fleet. We are accompanied by numerous support vessels. I demand that you change your course 15° north. That's 15° north, or countermeasures will be undertaken to ensure the safety of this ship.

British: This is a lighthouse. Your move …

We can only assume that the US ships adjusted their course south.

5. (12 points total) Concurrency problem: In parallel programs (one multi-threaded process), a common design methodology is to perform processing in sequential stages. All of the threads work independently during each stage, but they must synchronize at the end of each stage at a synchronization point called a *barrier*. If a thread reaches the barrier before all other threads have arrived, it waits. When all threads reach the barrier, they are notified and can begin execution on the next phase of the computation.

   There are three complications to barriers. First, there is no master thread that controls the threads, waits for each of them to reach the barrier, and then tells them to re-start. Instead, the threads must monitor themselves and determine when they should wait or proceed. Second, for many dynamic programs, the number of threads that will be created during the lifetime of the parallel program is unknown in advance, since a thread can spawn another thread, which will start in the same program stage as the thread that created it. Third, a thread may end before the barrier. In all cases, all threads must synchronize at the barrier before the processing is allowed to proceed to the next phase.

   a. (10 points) Provide the pseudo-code for a monitor class called `Barrier` that enables this style of barrier synchronization. Your solution must support creation of a new thread (an additional thread that needs to synchronize), termination of a thread (one less thread that needs to synchronize), waiting when a thread reaches the barrier early, and releasing waiting threads when the last thread reaches the barrier. *Implement your solution using monitors* (e.g., `wait()`, `signal()`, and `signalAll())`.

      Your class must implement the following three methods: `threadCreated`(), `threadEnd`(), `barrierReached`().
      Hint: this concept is very similar to Java `synchronized` objects.

```
Class Barrier () {
    ConditionVar barrier; Lock lock = FREE;
    Int numThreads = 0; Int numThreadsAtBarrier = 0;

    threadCreated() {
        lock.acquire();
        numThreads ++
        lock.release();
    }

    threadEnd() {
        lock.acquire();
        numThreads--;
        if (numThreadsAtBarrier == numThreads) {
            numThreadsAtBarrier = 0;
            barrier.signalAll();
        }
        lock.release();
    }

    barrierReached() {
        lock.acquire();
        numThreadsAtBarrier++;
        if (numThreadsAtBarrier < numThreads) {
            barrier.wait();
        } else {
            numThreadsAtBarrier = 0;
            barrier.signalAll();
        }
        lock.release();
    }
}
```

We subtracted points as follow:

- *Answers with semaphores received no credit*
- *Answers without locks lost 5 points, incorrect lock use lost 1 point for each error (max –5 points)*
- *If your threadEnd procedure didn't check the ending thread was the last one not at the barrier, you lost 2 points*
- *If you created a thread in threadCreatED, you lost 1 to 3 points*
- *If threads could slip through your barrierReached, you lost 3 to 5 points*
- *If not all threads were released after the barrier, you lost 4 points*
- *Excessive thread wakeups (e.g., in threadEnd) cost you one point*
- *Answers with indefinite waiting (never finishing the barrier) lost 5 points*
- *Answers with busy waiting lost 5 points*

b. (2 points) In your `barrierReached` method, which conditional statement (i.e., `if`, or `while`) did you use and why?
The correct choice for most solutions was an if statement. Using a while was an extra, unnecessary check for most solutions, thus we gave no credit or partial credit for answers based on using Mesa-style monitors. If you said you needed a while because new threads could be created while all threads were waiting at the barrier, we subtracted one point, as no new threads can be created by the threads (since they're all waiting at the barrier).

However some solutions that checked for all threads at the barrier in threadEnd required a while statement. We gave full credit in such cases.

We gave no credit for stating that you used a while with Hoare-style monitors, as such monitors are extremely difficult to implement in practice.