UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS 162 Alan Jay Smith
Fall, 1998

Midterm 1, October 5, 1998
Part I

You have until the time announced for this exam. The exam is closed book. All answers should be written on the exam paper. Anything that we can't read or understand won't get credit. Any question for which you give no answer at all will receive 25% partial credit. Please answer in standard English; illiterate or illegible answers to essay questions will lose credit. Please watch the front board for corrections and other information. This exam has 7 questions on 6 pages and is in two parts.

Name (last, first, middle):____

Student ID #___________________________

Class Account:_______________________________________

1. What is the difference between an open and a closed (queueing) system? In studying scheduling algorithms, why does it matter which one we use? Does the scheduling algorithm affect the throughput in an open system? Explain. (12)

An open system is one in which the arrival rate is not related to the number of customers in the system.

```
   world  -----> system  -----> done
```

A closed system is one in which the total number of customers in the "system"+"the world" is constant. Thus, typically, the arrival rate drops as the number of customers queued or in service increases.

```
                 <-----world--------<
                 |                  |
                 |                  |
                 >-----system------>^
```

The reason to use the open system is that it is easier to analyze and to say definite things about. The reason to use a closed system is that it is more realistic - almost any real system is closed. The behavior of the system under different queueing disciplines may not be the same (at least in certain respects), so in some cases it is important to use a realistic model.

In an open system, the throughput is invariant with the scheduling algorithm (as long as rho (=arrival rate/service rate) is <1. In such a case, all arriving jobs get processed and leave, so the throughput is exactly equal to the arrival rate. Since in (almost) any real system the throughput does vary with the scheduling algorithm, we can see that it does matter whether we use an open or closed system.

==

2. Assume that a cafeteria has a person making sandwiches. In most cafeterias, FIFO scheduling is used. I.e. the person at the head of the line gives his/her order, and the sandwich maker makes the sandwich, gives it to that person, and then takes the next order. Please compare and explain the relative desirability of using SET, FIFO and RR scheduling for the sandwich making process. (12)

The point of this question was to realize that sandwich scheduling is NOT the same as CPU scheduling. A lot of people just recited what they learned about CPU scheduling (in some cases by making the unreasonable assumption that sandwich making times are highly skewed). The relative desirability of scheduling algorithms depends on the job processing time distribution (highly skewed for the CPU, very small skew for sandwich maker in a cafeteria) and the overhead of task switching (low for CPU, high for sandwiches).

In a cafeteria (not a take-out sandwich shop in the business district), almost all customers order one sandwich, and most sandwiches take a similar amount of time to make. I.e. the sandwhich-making-time distribution is centralized, not skewed. Thus FIFO should give the lowest flow time. RR will have a much higher flow time (remember the example I gave in class?), and also will incur high overhead, as the sandwich maker switches between partially finished sanwiches. SET involves always working on the least finished sanwich. For a centralized distribution (i.e. the expected time to completion is decreasing), this is the worst algorithm - i.e. gives the highest flow time, and it also has high task switch overhead.

Name (last, first, middle):______________________________________

3. In class, four methods were given for minimizing the amount of space allocated to the page table. Please list and explain each. Indicate any advantages and disadvantages. (20)

Method 1 - reduce VA space

We can reduce the size of the virtual address space available to a process. For example, in a 32-bit system, the OS might choose to allocate page tables only for the first 128MB of the VA. If the page size is 4KB, this means we will only have 128MB/4KB = 32K entries instead of 4GB/4KB = 1M entries in the page table. Under this scheme, it would be illegal for a process to reference a virtual address above 128MB.

Advantages: No hardware support is needed and easy to implement. If the page fault handler sees a reference above 128MB, just say "illegal reference" and kill the process.

Disadvantages: Some processes may need more than 128MB of VA space. A likely problem is that the process doesn't need more than 128MB of real memory, but it uses non-contiguous blocks in the 32-bit address space. For example, it might want to allocate its stack starting from the top of the VA space.

Method 2 - multilevel page tables.

Let's take a step back and think about why a single level page table can get so large. Say we have a 32-bit address space, 4KB page size.

```
           page number         page offset
        + --- 20 bits --- + --- 12 bits --- +
```

Suppose a process needs to use only 16MB of memory, why do we say 4GB/4KB = 1M page table entries are required? In this single level scheme, we are using the first 20 bits of VA as an offset in REAL memory to locate a PTE. Even though most of the 1M pages will not be used by the process, we need to allocate space to hold 1M PTE's just so that those PTE's the process does use can be found.

In contrast, a two-level paging scheme might look like this.

```
          page directory      page number      page offset
        + --- 10 bits --- + --- 10 bits --- + --- 12 bits --- +
```

The idea here is to add a level of indirection in our search for PTE's. We recognize that many blocks of PTE's will never be used, so let's group these PTE's into 1024 groups with 1024 PTE's each. We use the first 10 bits of VA to index into a directory that points to information about each of the 1024 PTE groups. If a process never requested portions of its VA corresponding to a PTE group, the OS doesn't need to allocate memory for that group.

Thus, we use the first 10 bits to lookup a page directory, which returns a pointer to a 1024 entry page table (PTE group), and we use the next 10 bits to lookup that page table. Assuming no internal fragmentation (both within pages and PTE groups), only 1024 + 16MB / 4KB = 5K entries are required for this process.

Advantages: Full address space available to processes. Can extend to arbitrary levels of indirection to handle even larger address spaces.

Disadvantages: We can suffer from fragmentation in PTE groups. What happens if a process needs exactly one page in every PTE group? Even though only 1024 pages are used, 1024 + 1M PTE's would be allocated. Also, this method requires specific hardware support (to avoid trapping to OS at EVERY memory reference) and may incur a performance penalty due to indirection.

Method 3 - Put User Page Tables in OS Virtual Memory

We can use virtual memory for the operating system as well as for the user. We can allocate single level user page tables in the operating system's virtual memory. Of course, those page tables are huge, but if a given user process is only using a small number of pages, only those pages of the user's page table which contain active PTEs actually have to be in memory. The remainder are allocated in the OS virtual memory, but don't have to be physically resident in main memory.

Of course, the problem now is that the operating system uses virtual memory, and we can't put the OS page tables in the OS's own virtual memory and page them - we get into a loop. So we put the OS page tables in real memory. Note that we thus have 2 level addressing again - the OS page table and the user page table.

Advantages: This is pretty much the same as scheme 2.

Disadvantages: Pretty much the same as scheme 2, plus the problem of finding enough real memory to contiguously allocate the OS page table.

Method 4 - Inverted Page Tables

Page tables map every virtual address to a physical address. Page tables can become wastefully large because some virtual addresses never get used. So why don't we use a hash table? The Hash table only needs O(# of page frames in real memory) size - e.g. perhaps twice as many entries as there are page frames in real memory. The obvious problem here is that lookups now involve doing a hash table lookup in hardware. (It's easy in software; it isn't so easy in hardware.) We can share one table among all processes (and index it by VA and PID).

Advantages: Page table size is proportional to the amount of real memory. Only one table is needed for all processes.

Disadvantages: Slow lookup time. Sharing pages among processes is difficult, since we need to figure out how to map different VA+PID pairs to the same physical address. Complex implementation.

Please refer to Chapter 8 of Silberschatz and Galvin for detailed explanations of methods 2 and 4. I accepted segmentation + paging in addition to multilevel paging. If you said increase page size, you received some credit as well.

4. The set in the TLB is usually selected using the low order vir-

tual address bits. Why? (10)

The question should have been phrased, "the set in the TLB is usually selected using the low order PAGE NUMBER bits. Why?" Everyone made this interpretation anyway.

The short answer is: locality of reference, in both space and time. If page x was referenced, we expect nearby pages to be referenced, and also expect page x to be referenced again in the near future. To maximize hit rate, we would like to keep the most recently used entries in the TLB. How can we do that without full associativity? Using low order bits to index the set means during a cache miss, we evict an entry with the same low order bits but different high order bits, in other words, an entry that is spatially distant. Spatial distance implies temporally distance. Thus, the best way to mimick LRU eviction is to index sets with low order bits.

Another way to say the same thing is as follows. Locality of reference means the low order bits in a page number vary much more frequently than the high order bits. If high order bits are used for set selection, more temporally adjacent references would map into the same set, resulting in more cache misses.

Another view: most processes allocate a few contiguous areas of memory. If we used high order bits, those would all map into a small number of TLB entries.

5. Why is the text_and_set instruction preferred for synchronization to compare_and_swap? (10)

Compare_and_swap involves reading two arbitrary values and writing two arbitrary values. Test_and_set reads one arbitrary value, writes one arbitrary value (returning a value involves writing to a register!), and writes a fixed value of 1. This is simpler to implement efficiently in hardware because it involves one less read, and we do not need additional datapath to carry data to the fixed value write. For a 32-bit machine, we need to use a 64-bit datapath to perform a swap in one cycle, but only 32-bit datapath to do test_and_set.

I accepted most answers that addressed the additional complexity of compare_and_swap. Note however, that these instructions are implemented in hardware, and reasoning about how it would be implemented in software (how many instructions it takes, or the use of temporary registers) is not exactly correct.

6. For each of FIFO, SRPT, and RR (Q=.25), and for the following set of arrival and service times, please show a time line for which process is executing, and compute the mean flow time. Show your computations. (We might given partial credit, if you made a simple and obvious error; we're not going to try to decode your calcula- tions if they aren't obvious.) (20)

	arrival	service
A	0	1.75
B	.4	.9
C	1.4	1.1

FIFO: (6 points)

```
   -----------------------------------------------------------
   |           A            |       B        |      C        |
   -----------------------------------------------------------
   0                       1.75             2.65            3.75

                (1.75 - 0) + (2.65 - .4) + (3.75 - 1.4)
   mean flow = --------------------------------------- = 2.12
                                  3
```

SRPT: (7 points)

```
   -----------------------------------------------------------
   |   A   |     B      | A |        C       |        A      |
   -----------------------------------------------------------
   0      0.4          1.3 1.4              2.5             3.75

                (3.75 - 0) + (1.3 - .4) + (2.5 - 1.4)
   mean flow = ------------------------------------- = 1.92
                                 3
```

RR: (7 points)

```
   ------------------------------------------------------------
   |  A  |  A  |  B  |  A  |  B  |  A  |  B  |  C  |  A  |  B  |
   ------------------------------------------------------------
   0    .25    .5   .75   1.0   1.25  1.5   1.75  2.0   2.25  2.4

   -------------------------------------
   |  C  |  A  |  C  |  A  |  C  |  C  |
   -------------------------------------
  2.4   2.65  2.90  3.15  3.4   3.65  3.75

                (3.4 - 0) + (2.4 - .4) + (3.75 - 1.4)
   mean flow = ------------------------------------- = 2.58
```

Also accepted for RR was a variation where the new process goes on the front of the queue, not the back. The mean flow time in that case is 2.67.

7. For the following two cases, please either show a complete safe sequence or show that there isn't one. (16)

Process	has-X	has-Y	max needs-X	max needs-Y
A	30	40	45	330
B	20	90	80	120
C	50	30	90	70
D	70	100	130	250

a. available: X: 70 Y: 70

(8 points) There are three possible solutions: B, C, D, A; B, D, C, A; and C, B, D, A. The available resources after each process is run is as follows.

B, C, D, A:

X	Y
90	160
140	190
210	290
240	330

B, D, C, A:

X	Y
90	160
160	260
210	290
240	330

C, B, D, A:

X	Y
120	100
140	190
210	290
240	330

b. available: X: 70 Y: 65

(8 points) There is no safe sequence because there is only 325 Y in the system and process A needs 330 Y to run.