University of Rochester **Computer Systems (CSC2/456)**

Final Exam

Amy Murphy

6 May 2003

Read before beginning: Please write clearly. Illegible answers cannot be graded. Be sure to identify all of your answers in your blue book (e.g., 1a, 2b, etc). All questions are worth 10 points, numbers in parenthesis indicate relative point values.

While I have tried to make the questions as unambiguous as possible, if you need to make any assumptions in order to continue, state these as clearly as possible as part of your answer. In the name of fairness, I would like to avoid answering questions during the exam.

CSC256: Students enrolled in 256 must choose **6 of the 8** questions to answer. For up to 10 extra credit points, you may answer **one** of the other 2 questions, indicating clearly on the front of the bluebook which question is to be counted as extra credit. If you do not specify, I will assume the first 6 in your bluebook are for *normal* credit and the next one (if any) is *extra.* If you answer all 8 questions, only 7 of them will be counted toward your grade. Grades will be assigned 0-60 (with 10 possible extra credit).

CSC456: Students enrolled in 456 must choose **7 of the 8** questions to answer. For extra credit, you may answer the 8th question, but you must clearly identify which question is to be counted as *extra.* If you do not indicate any, I will assume the 8th answer (if any) is *extra.* Grades will be assigned 0-70 (with 10 possible extra credit)

1. **Short answer.** Your responses for each part should be at most 3-4 lines of text:

   (a) (3) What is the difference between a user-level instruction and a privileged instruction?

   A privileged instruction can only be invoked when the hardware is running in privileged (a.k.a. kernel) mode. Examples of privileged instructions include those that move data in and out of protected device registers.

   (b) (3) What is the difference between a network operating system and a distributed operating system?

   A distributed operating system presents a single system image to many hosts in an environment, and remote operations are hidden from the users. A network operating system, while it still allows access to remote hosts, leaves the user acutely aware of the remote-ness of any operation issued.

   (c) (4) Name two positive motivations for building distributed systems and two problems they introduce.

   Positives: speed, support inherent distribution, reliability, incremental growth, resource sharing. Negatives: overhead of communication and synchronization, consistency management, software complexity, increased security problems

2. **Scheduling.** I have just invented a new scheduling algorithm that I claim gives the highest priority to processes that have just entered the system, but is fair to all processes. The algorithm works like this: There are two queues, one for *new* processes and one for *old* processes. When a process enters the system, it is put at the end of the *new* queue. After 2 milliseconds on the *new* queue, whether a process has been scheduled or not, it is moved to the end of the *old* queue. When it is time to schedule a process, the system schedules the process at the head of one of the queues, alternating between the two queues. Each process runs to completion before the next process is scheduled. Assume that processes enter the system at random times and that most processes take much longer than 2 milliseconds to execute.

   (a) (4) Does this algorithm give the highest priority to new processes? Explain your answer.
   This algorithm does not give the highest priority to new processes. There are several reasons. First, even if executing processes took much less than 2 milliseconds to execute, the scheduler would alternate between "new" processes and "old" processes, giving equal priority to both. Second, given that executing processes take much longer than 2 milliseconds to execute, most "new" processes will not get scheduled during their 2 milliseconds on the "new" queue and will then drop to the bottom of the "old" queue without ever having been given any priority. Now they have to wait for processes older than them to execute, and processes newer than them to execute.

   (b) (3) Is this algorithm starvation free? Explain your answer.
   Yes. Because the scheduler alternates between the two queues, every job that is not executed from the "new" queue eventually ends up in the "old" queue and from there, eventually reaches the head of the queue and is executed.

   (c) (3) Discuss whether this algorithm is fair to all processes. By "fair" we mean every process has a wait time approximately equal to the average wait time, assuming all processes have close to the same execution time.
   No, it is not fair to all processes. Some lucky "new" processes will get to execute when they reach the top of the "new" queue, while some will drop to the bottom of the "old" queue and have to wait much longer to execute. These unlucky processes will have to wait for all of the processes older than them to complete, and will have to wait for many processes younger than them to complete as well.

3. **Synchronization.** Suppose a program has three threads Thread$_1$, Thread$_2$, and Thread$_3$, and a shared counter, `count`, as shown below:

```
int count = 10;
Semaphore Lock = 1; // initial value is 1
```

```
Thread1(...)
{
// do something
Lock.Wait();
count++;
Lock.Signal();
}
```

```
Thread2(...)
{
// do something
Lock.Wait();
count--;
Lock.Signal();
}
```

```
Thread3(...)
{
// do something
Lock.Wait();
printf(``$d'', count);
Lock.Signal();
}
```

   (a) (4) What are the possible outputs of this program? Hint: there is more than one answer, and you must provide them all.
   9, 10, 11

   (b) (3) Define a race condition.
   A race condition is when it is possible to observe the order in which events in different processes occur, and that order is not constrained by synchronization.

(c) (3) Does this process suffer from a race condition? Justify your answer.

A race condition does exist here because the output can be different for different orderings of the program.

Note: if your definition of a race condition addresses only the final result of the data, then the answer to this part of the answer could be "no" because the final result of the variable count is always 10.

4. **Caching.**

(a) (3) Define spatial locality and temporal locality.

Spatial locality *means the access to a data item makes near-future accesses to spatially nearby data items more likely.* Temporal locality *means the access to a data item makes another access to the same data item in the near future more likely.*

(b) (3) Disk manufacturers added a full track buffer to hard drives to improve program performance. Which of spatial or temporal locality most motivated this? Explain your answer.

Spatial locality. Access to one data block makes near-future accesses to other blocks on the same track more likely. Additionally temporal locality is already exploited by the operating system's file buffer cache

(c) (4) Hardware caches in the memory hierarchy are other performance enhancing techniques. Explain how/why spatial locality and temporal locality contribute to this performance enhancement.

The concept of caching recently accessed data items is motivated by temporal locality. The fact that a cache line (32 or 64 bytes) is usually larger than what an instruction needs is motivated by spatial locality.

5. **Paging.** Consider a reference string 1,2,3,4,2,5,6,2,3,2,1,6,7; and a system with only 4 frames, pure demand paging, and all frames initially empty.

(a) (2) How many page faults would occur with a FIFO replacement scheme? What are the identities of pages in the frames when the reference string has completed?

10 page faults, [2 3 1 7]

(b) (2) How many page faults would occur with a perfect LRU replacement scheme? What are the identities of pages in the frames when the reference string has completed?

9 page faults, [2 1 6 7]

(c) (3) Describe one possible implementation scheme for LRU.

A linked list: always evict the head page, new page join at the tail, the hit page is relocated to the tail.

Associate a timer with every page. Update it when the page is accessed. To evict a page, find the timer with the oldest value.

(d) (3) Would increasing the number of frames always decrease the number of page faults for a particular reference string for FIFO? for LRU? Why or why not? You need not provide "proofs", just an explanation.

FIFO: no. Belady's anomaly states that it is possible to have more resources, and worse access behavior.

LRU: yes. Belady's anomaly does not apply to stack based algorithms.

6. **Memory Management.** LRU can be thought of as an attempt to predict future memory access patterns based on previous access patterns (i.e. if a page has not been accessed for a while, it is not likely to be be referenced again soon). Another idea that some researchers have explored is to record the memory reference pattern from the last time the program was run and use it to predict what it will access next time.

   (a) (3) For what kinds of programs will this be most effective?

   Programs that repeat the same access pattern every time they run, i.e. deterministic programs in which the sequence of memory operations does not depend on the values of program inputs. This is most commonly true for scientific, mathematical applications.

   (b) (3) In what kinds of programs might this lead to worse performance?

   Programs that exhibit different access patterns on different inputs, or even on the same input (e.g., some non-deterministic programs).

   (c) (4) Why would this be difficult to implement in an operating system? Give at least two reasons.

   We need some mechanism to collect this information. We need to store this information, and retrieve it when a program is run again. We need to change the page replacement scheme to work with this collected information.

7. **File systems.**

   (a) (4) A file system must keep track of free blocks on disk. Name two schemes for doing this, and one advantage of each.

   Linked list of free pages supports very fast deletion: we can link a whole file onto the free list in constant time.

   Bit map facilitates spatial locality optimizations in disk layout. It is also the most effecient in terms of memory inside the OS required to find the free pages.

   The list of pages with free page references in each is also space-efficient, particularly for disks that are mostly full (the bit map takes space proportional to the size of the entire disk). Also, free page insertion and deletion operations exhibit high locality under this scheme.

   (b) (3) Consider a very large file: the maximum size file supported by a particular system. Describe the process to read the last byte of that file if it is stored in a FAT file system. Assume you already have the directory entry cached.

   From the directory entry, you will find the index of the first block. Looking up this entry in the FAT will give you the next block index. Keep following this chain until the block you want is reached.

   (c) (3) Describe the process to read the last byte of that file if it is stored in a UNIX inode-like system where the inode contains 10 direct block pointers, one single indirect pointer, one double indirect pointer, and one triple indirect pointer. Assume you already have the inode cached.

   From the inode, look up the triple indirect pointer. Read the indicated block off the disk, and look at the last pointer in this block. This points to another indirect block. Read that block, and the last pointer in this block. This points to another indirect block. Read the last pointer, and it will get you to the last block of the file.

8. **Unix File Systems.**

(a) (3) One of the changes FFS (Fast File System) made over the "classic" Unix file system was to place the inodes at multiple locations on the disk, as opposed to placing all inodes at the front. Why did this (generally) increase file system performance?

Co-locating the data blocks and their corresponding inodes in the same track reduces the need for disk seeks.

(b) (3) FFS tries to achieve the advantages of both of large and small block sizes. Briefly describe the technique by which this is achieved, and state one of its disadvantages.

FFS introduced the ability to create up to 16 block fragments per block. This allowed data that was not the size of a full block to actually share a block with information from other files. The main disadvantage is increased overhead to maintain fragment information, and complexity of implementation of the block allocation scheme: e.g., balancing the amount of data copying with the number of fragments appearing in a file. It is worth noting that the importance of this technique has, arguably, declined with increases in disk capacity. Though block sizes have increased, disk capacity has increased much more, so the internal fragmentation due to under-full blocks has declined as a percentage of total capacity.

(c) (4) What was the motivation for LFS (Log-based File System), and why does it achieve better write performance than FFS when there are many random writes of a large file?

The motivation is that when disk caches get large enough, most reads can be served from memory while only writes go to the disk. Treating random writes as log appends essentially converts random disk accesses to sequential accesses, which reduces disk seek and rotational delays.