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

# Final Exam

Amy Murphy

8 May 2001

**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. Some questions are long to read, but short to answer.

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 will avoid answering questions during the exam.

**CSC256:** Students enrolled in 256 must choose any **5 of the 7** questions to answer. For extra credit, you may answer one of the remaining 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 5 in your bluebook are for *normal* credit and the remaining 1 (if any) is *extra.* Grades will be assigned 0-50.

**CSC456:** Students enrolled in 456 must choose any **6 of the 7** questions to answer. For extra credit, you may answer a 7th question, but you must clearly identify which question is to be counted as *extra.* If you do not indicate any, I will assume the first 6 in your bluebook are for *normal* credit and the remaining 1 (if any) is *extra.* Grades will be assigned 0-60.

1. **Memory Management.** Answer the following with a few lines of text for each.

   (a) What is the difference between a physical address and a virtual address?

   *A virtual address is used by applications to logically refer to a location in a program or data. In order to access the contents of this location, it must be mapped into a physical address which identifies the actual memory location where the data resides.*

   (b) Define thrashing as it pertains to main memory management.

   *Thrashing occurs when memory pages are constantly being swapped out of main memory and to disk at a rate too rapid to accomplish any real work. In other words, the system is spending all of its time swapping pages and not doing computation.*

   (c) Summarize the tradeoffs among simple arrays, trees, and hash tables as implementations of a page table.

   *Simple arrays are quick to access whereas trees and hash tables are more complex to implement but may take less space, especially when issues of virtual memory are concerned where the address space is much larger than physical memory.*

2. **Process Management.** Some distributed operating systems and some multiprocessor systems allow running processes to be migrated from one processor to another.

   (a) What are the various components of the process's state which must be collected in order to restore the process completely?

   *You must copy all of the process control block (including the instruction pointer), all of the process's data (including stack, heap, and program text), and all of the process state relating to any open files, open network connections, etc).*

   (b) Name two nontrivial problems that have to be solved to make process migration work.

   *Heterogeneous systems make migration very difficult. Pending I/O requests must be handled correctly (Moving network connections and open files is not trivial). Any memory pages* *shared between this process and another must be managed. If the process has multiple threads, they must all be packaged together and restarted at the destination. Also: interactive and real-time response and quality of service is a real bear if migration is time-consuming.*

3. **Synchronization.** Synchronization mechanisms are typically implemented using atomic operations provided by the underling architecture; e.g., test-and-set or compare-and-swap. Consider an operation: fetch-and-increment (`fai`), which behaves as an atomic counter that is sampled, then incremented. It can also be initialized to 0. In other words, if you create an `fai` variable using `fai a;` you can issue two atomic operations, namely `int j = a++;` and `a=0;`. Your task is to use `fai` to implement a solution to the critical section problem. You need not guarantee fairness among multiple processes. Assume no `fai` variable can overflow. (Hint: this is similar to the implementations you did for critical sections in your homework using TAS and CAS.)

```
fai inside = 0; // inside is an atomic counter
int a;

repeat
    a = inside++;
until (a == 0)

/* C.S. */

inside = 0;
```

   *The intuition here is that only one process will have the chance to read the atomic counter when it is equal to zero, and you are guaranteed that some process will see the value when it is equal to zero. If there is contention to get into the critical section, then only the first process to increment* `inside` *will be granted access.*

4. **Scheduling.** Why are simple process priorities insufficient for soft-real time scheduling such as that required by multimedia systems? In addition to the scheduler being able to handle deadlines and assuming no dependencies between processes, what else must be added to guarantee that the processes on the system will meet their deadlines?

   *With only priorities, processes can still miss their deadlines. Priority inversion is also an issue in priority-based scheduling schemes. We also need an admission controller so that we do not admit too many processes and overload the system to a point where the deadlines cannot be met.*

5. **File Systems.** Imagine a file system which uses inodes to manage files on disk. Each inode consists of a file name (4 bytes), user id (2 bytes), three timestamps (4 bytes each), protection bits (2 bytes), a reference count (2 bytes), a file type (2 bytes), and the file size (4 bytes). Additionally, the inode contains 13 direct indices, 1 index to a single indirect block, 1 index to a double indirect block, and one index to a triple indirect block. Each of these indices is 4 bytes. The file system also stores the first 356 bytes of the each file in the inode.

   (a) Assume a disk sector is 512 bytes and that each indirect block fills a single sector. What is the maximum file size for this file system? Show your work *clearly.* You need **not** do the arithmetic to get full credit.

   512/4 *(number of block entries in an indirect block)*

   356 (data in inode)
   $+13 * 512$ (data directly indexed)
   $+128 * 512$ (data referenced by single indirect)
   $+128 * 128 * 512$ (data referenced by double indirect)
   $+128 * 128 * 128 * 512$ (data referenced by triple indirect)

(b) Is there any benefit to including the first 356 bytes of the file in the inode? If so, what? If not, why not?

*Yes, for small files, there will be no need to use any extra sectors to store the data.*

(c) Describe in general how a RAID disk can be used to improve the fault tolerance of the file system.

*A RAID can be used to replicate the entire data on multiple disks, providing fault tolerance through replication. Alternately, the RAID can be used to store error correcting codes which can be used to recover bits lost if one of the disks crashes.*

6. **I/O.** (Note, this is long to read, but not long to answer.) Many I/O systems have independent caches and independent buffers which are isolated from one another within the operating system. For example, a network card uses buffers to store incoming data as well as data to be transmitted. This buffer is different from the buffer used by the file system to store recently accessed files. This buffer is still different from buffers of other I/O devices such as the keyboard or the video card. Consider a web server which continuously receives TCP requests for specific web pages. At the application level, a smart server may choose to keep a buffer of frequently accessed pages (yet another buffer different from those managed within the OS!).

   When the web server receives the request for a page, it first accesses its application level cache. If the page is not there, the server does an open/read from the file system, then opens a new TCP connection, and puts the data on this outgoing socket, sending it back to the server that requested the data.

   (a) Briefly describe what copies of the data are made in the worst case if this process were to receive the request and serve a page (hint: there are many!).

   *The copies made are:*

   - incoming TCP packet copied to the incoming buffer of the network card
   - request copied to server application to read the http request
   - web page file copied into the file system cache
   - web page copied into the memory of the server application
   - web page copied into the buffer of the network card

   (b) It seems a natural place for optimization in the operating system to reduce the amount of data copying in situations such as described in the web server example where the data being accessed is read-only. Provide a sketch of how this can be accomplished.

   *One way to achieve this is to exploit virtual memory. When a buffer is needed by a device other than the one which created it, the memory of that process is mapped to the existing copy of the data. Thus, the only data changed when "copying" data in this manner is the memory reference.*

   (c) Name at least two reasons you could expect performance gains in your applications.

   *Optimization comes obviously by reducing the number of copies, saving the CPU time necessary to do the copies. Another source of optimization comes because this approach frees up the memory used by the multiple devices, allowing more memory to be used to cache more files, further increasing performance by having more pages immediately ready to be served.*

7. **Distributed Systems.**

   (a) Briefly describe two motivations for building distributed operating systems such as Amoeba rather than simply using a set of standard operating systems running in a networked environment.

   *economics, speed, reliability, flexibility (incremental, transparent growth)*

(b) Briefly describe two obstacles that must be overcome when developing distributed systems.
*sw is complex, consistency of data sharing, network not perfect*

(c) Describe one specific instance in distributed operating systems where a choice must be made between transparency and making the user aware of the distribution.
*process migration, file storage, memory allocation*