CSCI 3753
Operating Systems
Spring 2005

Final Exam

April 30, 2005

SOLUTIONS

Professor Rick Han
Department of Computer Science
University of Colorado at Boulder

Instructions:

1. Put your name and student ID below
2. You should have 13 pages with 5 problems
3. It is to your advantage to partially answer a question rather than not to answer a question at all.
4. Show the details of your work and state assumptions you make. Simply writing a result or formula will not receive full credit.
5. All aspects of the Honor Code apply to this exam.

Name: ______________________________

ID: ________________________________

Problem 1: ________ / ~~64~~ 58
Problem 2: ________ / ~~51~~ 47
Problem 3: ________ / ~~21~~ 17
Problem 4: ________ / 16

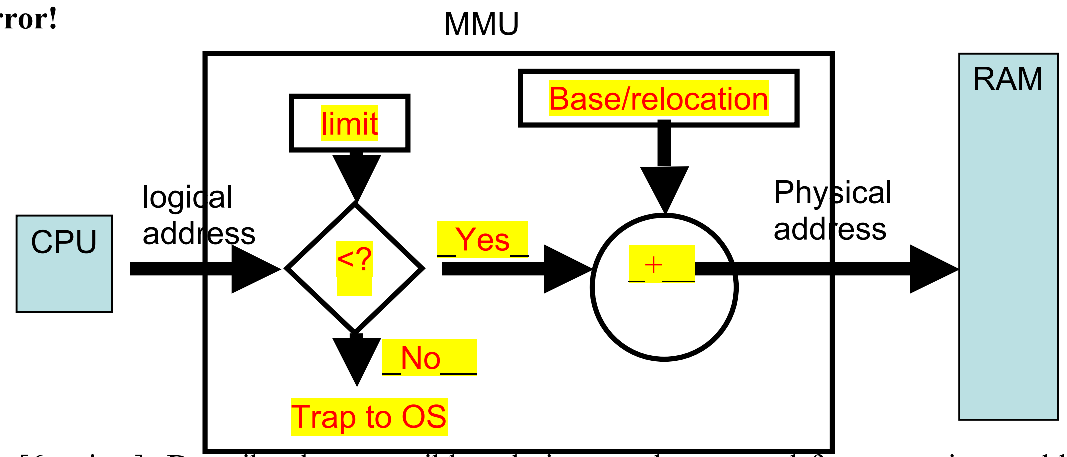
Problem 5: ________ / 12

Overall Score: ________ / ~~164~~ 150

Problem 1 (64 points): Memory Management

(a) [7 points] Label the various elements, operations, or states in the figure that describe the steps taken by the memory management unit to translate a logical address to a physical address:

Error!



(b) [6 points] Describe three possible solutions to the external fragmentation problem experienced when trying to fit processes into main memory. Explain how each solution avoids external fragmentation.

+2 each for any three of the following: paging, segmentation of a process, compaction, hierarchical paging, inverted page tables, hashed page tables, an allocation strategy to help minimize fragmentation like worst-fit, etc.

~~(c) [6 points] Assume that the page table is stored in RAM, that there is hardware support for TLB's, and that the CPU is initially given a logical address. Describe the various steps experienced in order to retrieve data from a physical memory address corresponding to the logical address.~~

deleted

(d) [6 points] Explain three advantages of on-demand paging in comparison to loading a process's entire address space into memory.

+2 for each:

- more processes in memory
- decreases swap time
- allows process's address space to exceed physical size of memory

(e) [4 points] Suppose every process's page table is very large and is stored in memory. Suppose also that most of the pages in each page table are unallocated, i.e. the address space is very sparse. Is there any benefit to using an inverted page table in this situation? Explain why or why not. Is there any benefit to using a hashed page table in this situation? Explain why or why not.

[+2] Yes, an inverted page table will reduce the amount of memory devoted to paging, because there is only one entry for each page in memory
[+2] Yes, a hashed table will reduce the size of the page table, because empty entries will not need to be allocated

(f) [4 points] Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

[+1] First-fit:
b. 212K is put in 500K partition
c. 417K is put in 600K partition
d. 112K is put in 288K partition (new partition 288K = 500K - 212K)
e. 426K must wait

[+1] Best-fit:
g. 212K is put in 300K partition
h. 417K is put in 500K partition
i. 112K is put in 200K partition
j. 426K is put in 600K partition

[+1] Worst-fit:
l. 212K is put in 600K partition
m. 417K is put in 500K partition
n. 112K is put in 388K partition
o. 426K must wait

[+1] In this example, Best-fit turns out to be the best.

(g) [6 points] Suppose a process executing an instruction causes a page fault. Describe the roughly 6 steps undertaken by the OS to service the page fault, until the process resumes executing the instruction that caused the page fault.

+1 for any of the numbered or bulleted lines,

1. MMU detects a page is not in memory (invalid bit set) which causes a *page-fault trap* to OS. +1 also if mention TLB.
2. OS saves registers and process state. Determines that the fault was due to demand paging and jumps to page fault handler
3. Page fault handler
    - If reference to page not in logical A.S. then seg fault.

        - Else if reference to page in logical A.S., but not in RAM, then load page
        - OS finds a free frame.
            1. If no free frame, OS employs page replacement policy
        - OS schedules a disk read. Other processes may run in meantime.
    4. Disk returns with interrupt when done reading desired page. OS writes desired page into free frame
    5. OS updates page table, sets valid bit of page and its physical location
    6. Restart interrupted instruction that caused the page fault

(h) [7 points] How many disk accesses are involved in a typical page replacement operation caused by a page fault? Describe three approaches for improving the performance of page replacement during a page fault. [Hint: look at the disk accesses].

+1 for noting that page replacement can cause 2 disk accesses, one to write out a replaced page, one to read into memory the requested page that triggered the page fault

any of the following is worth +2

- eliminate a disk write by using dirty/modify bit to identify unmodified frames for replacement
- remove a disk write from the critical path of a page fault by writing out modified pages earlier, so that more pages are unmodified and can be selected for replacement without requiring a disk write
- delay a disk write, read the requested page first, then write later
- remove a disk read by recording which read pages

(i) [4 points] Can page tables be used to share memory between two processes? Explain why or why not. Can inverted page tables be used to share memory between two processes? Explain why or why not.
+2, two page tables can share the same frame in memory by having the same frame # somewhere in their page table
+2, an inverted page table makes it difficult to share the same frame, because each frame in memory is owned by only 1 process

(j) [5 points] What is the cause of thrashing? How does a system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

+2 thrashing is caused by excessive page faulting, when a process or set of processes' working set exceeds the number of frames allocated to it/them in memory

+1 thrashing can be detected by looking at the CPU utilization, inspecting the disk queue, measuring the rate of page faults, etc.

+2 for any one of the following:

- employ a working set solution
- employ a page fault frequency measurement algorithm
- swap out processes, i.e. reduce the degree of multiprogramming

(k) [4 points] Explain how the Clock or Second-Chance Page Replacement Algorithm approximates an LRU page replacement algorithm.

+1 for each of the following:

- the Clock algorithm keeps a circular buffer, with a rotating pointer (clock hand)
- each page in the circular buffer has a reference bit
- when a page needs to be replaced, the hand rotates forward and inspects the reference bit of each page, If the reference bit = 1, then the page was referenced recently. The Clock algorithm gives this page a second chance, and sets the page to 0, and does not select this page for replacement.
- The first page encountered by the clock with the reference bit = 0 is selected for replacement
- This roughly resembles an LRU because ref bits = 0 correspond approximately to least-recently used pages

(l) [5 points] Suppose the working set algorithm is applied in a system with the following four types of process behavior: *sequential* execution, *random* execution, *looped* execution, and *recursive* execution. For which process type is the working set algorithm most effective at reducing excessive page faulting? Rank the process types from most effective to least effective and explain your reasoning.

There are roughly two equivalence classes in the effectiveness of the working set algorithm:

+2 Most effective: looped, recursive – working set captures this locality of reference. If the loop spans much code, then recursion might be more local. If the recursion spans much code (usually recursion is tight), then looping is more local

+2 Least effective: sequential, random – working set is probably too small to capture randomness. Sequential is interesting, because even though the accesses are correlated by increasing page #, the working set will not capture the correlation, because the working set is only useful for *reused* pages.

+1 for a reasonable explanation of any one of the above rankings

Problem 2 (51 points): File Systems

(a) [6 points] List the four key types of data structures maintained by a file system *on disk*. List at least two key data structures maintained by the OS on behalf of the file system *in memory*.

+1 each for any four of the following: directory structure, file header/FCB/inode, file data, free blocks (volume entry), boot block

+1 each for any two of the following: system-wide open file table, per-process open file table, cached directory entries, cached free block information, cached FCBs, cached delayed writes, etc.

~~(b) [4 points] Explain how deleting a symbolic link differs from deleting the file associated with the symbolic link. Explain how symbolic links are treated differently than files during searching for a filename.~~

deleted

(c) [6 points] When a process calls *open*(filename), explain the typical steps undertaken by the operating system to execute this open() call.

- when a process calls open(foo.txt) to set up access to a file, the following procedural steps are followed: [+2 for any numbered or bulleted line]
    1. the directory structure is searched for the file name foo.txt
        - if the directory entries are in memory, then the search is fast

    - otherwise, directories and directory entries have to be retrieved from disk and cached for later accesses
2. once the file name is found, the directory entry contains a pointer to the FCB on disk
    - retrieve the FCB from disk
    - copy the FCB into the system OFT. This acts as a cache for future file opens.
3. add an entry to the per-process OFT that points to the file's FCB in the system OFT
4. return a file descriptor or handle to the process that called open()

(d) [9 points] In the following, draw a line between the file allocation method and its primary disadvantage(s), if any:
+1 for each

File Allocation Method:

1. Contiguous – b, d, f (maybe h)
2. Linked List - e , g (maybe h)
3. FAT - e , g (maybe h)
4. Indexed Allocation (1 index block) - h
5. UNIX inode - none

Primary Disadvantages:

(a) TLB misses

(b) slow growth

(c) sparse address spaces

(d) external fragmentation

(e) traversal latency

(f) size of file needed a priori

(g) corruption of a pointer

(h) large files

(e) [4 points] In which of the following disk scheduling algorithms may starvation of a request occur? Recall that disk requests can arrive at any time in any order into the buffer of pending requests.

C-SCAN Starvation can occur in any of the following. +1 for noting that for each algorithm.
LOOK

FIFO

SSTF

(f) [6 points] List two free space management techniques for file systems and give one advantage and one disadvantage of each technique.

+1 bit vector/bit map
- +1 for advantage: simple, easy to search
- +1 for disadvantage: large, empty entries waste space

+1linked list
- +1 for advantage: no empty entries, exactly
- +1 for disadvantage: slow to traverse

+1 for Grouping -linked list, except store n-1 pointers to free blocks in each list block, while the last block points to the next list block containing more free pointers
- +1, advantage: allows faster allocation of larger numbers of free blocks all at once
- +1, disadvantage: traverse linked list

+1 for Counting - grouped linked list, except add a field to each pointer entry that indicates the number of free blocks immediately after the block pointed to
- +1, advantage: even faster allocation of large #'s of free blocks
- +1, disadvantage: traverse linked list

(g) [6 points] After a crash, explain how a log-based file system recovers, including the types of operations that must be performed on the log. Does the log need to be replayed in its entirety? If the system failed during a write to the log, then is the file system unrecoverable?
+2, after a failure, the LFS will redo() transactions that are committed but not yet completed, i.e. that have both a <start> and <commit>

+2, the LFS will undo() transactions that have a <start> but not a <commit>
+1 depending on how you define a log, the log i) does not need to be replayed in its entirety – only from the last checkpoint, or ii) does need to be replayed in its entirety starting with the oldest committed but not completed transaction
+1, no, the system is still recoverable after failing during a write to the log. The transaction will simply be uncommitted and will be undone during the recovery process.

(h) [4 points] Suppose the technique of *write-after* logging executes writes to the file system on disk prior to recording write operations on the log. If a file system crashes during a transaction like file create(), is this logging approach robust to failures? Why or why not?
+1, this write-after approach is not robust to failures. [+2] Example: in the middle of a create(), I may be writing the FCB when the failure happens. The FCB may only be partially written. Moreover, the file data has not yet been written. So the file is in an inconsistent state. [+1] The log should be written to *before* the actual operation.

(i) [6 points] Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for FIFO, SSTF, and C-LOOK disk-scheduling algorithms?
+2 points, The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081.
+2 points, The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774. The total seek distance is 1745.
+2 points, The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130. The total seek distance is 3363.

For reference, SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86. The total seek distance is 9769.
For reference, the LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek distance is 3319.
For reference, the C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 86, 130. The total seek distance is 9813.

Problem 3 (21 points): Security

Notation for the following problems: Let E(K,M) be equivalent to message M being encrypted by some key K. Alice and Bob both possess the same symmetric key K. Alice also generates her own private key $PRIV_{ALICE}$ and its associated public key PUB_{ALICE}. Similarly, Bob generates his private key $PRIV_{BOB}$ and public key PUB_{BOB}.

(a) [4 points] Suppose the following public key cryptography protocol ***A*** is used to protect login authentication. Alice wishes to login to Bob's server. Bob advertises his public key PUB_{BOB} to Alice. Alice uses the public key to encrypt her password, sending E(PUB_{BOB}, Alice's password) to Bob. Does this public key-based login protocol securely authenticate Alice to Bob? Explain why or why not.

[+2] This is not secure, and [+2] is subject to a *replay* attack. Trudy the eavesdropper can simply replay the public-key encrypted password and login as Alice, so the authentication is not secure.

~~(b) [4 points] Suppose the following security protocol ***B*** is used to protect login authentication. Alice wishes to login to Bob's server. Bob advertises his symmetric key to Alice. Alice then encrypts her password first with her public key PUB_{ALICE} and then a second time with the symmetric key K. Does this double encryption login protocol securely authenticate Alice to Bob? Explain why or why not.~~

deleted

(c) [4 points] Suppose the following security protocol ***C*** is used to protect login authentication. Alice wishes to login to Bob's server. Bob advertises his public key to Alice. Alice then chooses a *different* symmetric key K2 and sends the encrypted key K2 to Bob, i.e. E(PUB_{BOB}, K2). Alice then sends her password encrypted with K2, i.e. E(K2, Alice's password). Does this hybrid public/symmetric key-based login protocol securely authenticate Alice to Bob? Explain why or why not.

A variety of reasonable correct answers:

- [+2 explanation] Assuming that K2 is chosen differently each time Alice logs in, then this is basically the SSL protocol, so [+2] yes the authentication is secure. It is not subject to a replay attack.
- [+2] Not secure, because [+2] Assuming that the K2 is chosen only once, rather than each time, then this protocol is subject to a replay attack.
- [+2] not secure, because [+2] Because there is no certification of Bob's public key, this protocol is subject to a man-in-the-middle attack (we didn't really cover certification)

(d) [6 points] Explain how Unix implements authorization, i.e. checking of permissions for various file operations like read, write, execute, etc. [Hint: your answer should start with the concept of an access matrix]

+1, UNIX employs a combination of ACL and capability lists.
+1 An ACL is a column of the access matrix.
+1, A capability list is a row.
+1, in UNIX, the ACL is stored with the file attributes in the inode

+1, the file header is then loaded into memory in the system OFT (and per-process OFT)
+1, over time, the set of open files conceptually constitutes a partial capability list for the user/process

(e) [3 points] Unlike the Internet network stack, the OSI classification of networks layers actually defines 7 layers, including one where security is positioned. SSL is one example of such a layer. If SSL were to be added to the 5-layer network stack, between which two layers would SSL most likely be placed? Justify your answer. [Hint: answer Problem 4(a) first, then come back to this one]

[+1] SSL is an end-to-end protocol, and would have to be built at least on top of the IP network layer. Typically, secure login and secure Web access requires reliability , so SSL would have to be built on top of the transport layer's reliability. Applications need to call SSL to set up secure connections, so SSL would reside between the application and transport layers [+2]

Problem 4 (16 points): Networking

(a) [7 points] List the five layers of the network stack and explain the purpose of each layer.

+1, Application – application process, e.g. Web server, email client, etc.
+1, Transport – ensures reliability between two endpoints, if desired
+1, Network – routes packets over multiple hops
+1, Data Link/MAC – sends packets one hop, arbitrates collisions on broadcast media
+1, Physical – sends bits one hop

+1 for any two correct explanations, +2 for all five correct explanations

(b) [6 points] The Sun Networked File System (NFS) is a distributed file system that employs RPCs. Summarize how NFS would access a file located on a remote host, assuming that the file is located in a directory that has already been mounted.

+2, for any three of the following

- NFS employs a client-server architecture
- NFS client sends a vnode operation (e.g. read/write) on a file or directory to an NFS server daemon (nfsd) process executing on a remote server/host
- NFS server translates the operation to the local file manager's language
- depending on the operation (e.g. an open), a *file handle* may be returned to the NFS client which is used in future references to the file - this handle is a "pointer" to the file
- there may be multiple roundtrips needed, e.g. to find a remote file /usr/local/bin/emacs, NFS will first request the file handle from the NFS server for the mount point (let's say /usr/local was NFS-mounted), then will request the file handle for the bin directory, then the file handle for the emacs file
- NFS is stateless for fast crash recovery

(c) [3 points] Describe at least three different techniques used by NFS to improve its performance.

NFS improves performance through: +1 for any three of the following lines

- client caching
    i. client cache can service read requests immediately

ii. NFS client can pre-fetch file data using *read-ahead*
iii. NFS client can delay writes and return immediately, i.e. asynchronous writes or *write-behind*

- server caching.
    i. Cache directory entries in RAM
    ii. Cache FCBs in RAM
    iii. Cache file data blocks in RAM

Problem 5 (12 points): Processes, Threads, and Device Management

Short Answer:

(a) [2 points] How is an interrupt detected by the OS? How does the OS jump to the interrupt handler code?
+1, CPU polls interrupt pins in each instruction cycle
+1, CPU then indexes into the interrupt vector/jump table, determines which device called the interrupt, and jumps to the device handler.

(b) [2 points] In a user-level implementation of threads, if one thread in a process blocks, do the remaining user threads in a process continue to execute? Justify your answer.
[+1] No, the remaining user threads will be blocked. [+1] This is because the threads do not have their own thread of execution recognized by the kernel. The kernel only recognizes the process's sequence of execution.

(c) [2 points] Name two state values that are saved during a context switch.
+1 for any two of the following: PC, instruction register, integer registers, frame pointer, stack pointer, PTBR, etc.

(d) [2 points] What is the point of the Banker's Algorithm and what does it seek to find?
+1 the Banker's Algorithm's goal is deadlock avoidance
+1, B.A. seeks to find a safe sequence, to determine whether the system is in a safe state

(e) [2 points] Give a reason why the OS benefits by having hardware CPU support for a separate user mode and a kernel/supervisor mode.

- +2 OS needs to be protected from user programs that could otherwise arbitrarily execute risky instructions that could corrupt the OS. [or +1] Supervisor mode can execute all machine instructions, and can reference all memory locations. [and +1] User mode can only execute a subset of instructions, and can only reference a subset of memory locations

(f) [2 points] In the Dining Philosophers Problem, give one solution for preventing deadlock.

+2 for any one of the following: break circular wait, odd/even philosophers, only pick up two forks or none, order all resources, break hold and wait, allow at most four philosophers at the table when there are 5 resources, etc.

Have a good summer!