Name: _____

Operating Systems
CSCI-UA.0202 Fall 2013

Final Exam
ANSWERS

Please write your name on this sheet. Answer question 1 on this sheet and put all other answers in the blue book.

1. **True/False**. Circle the appropriate choice on this sheet.

   (a) **T F** DMA is a mechanism for allowing an I/O device to transfer data to and from memory without involving the CPU in the transfer.
   Answer: T

   (b) **T F** Memory mapped I/O determines how the pages of an I/O-bound process are mapped to page frames.
   Answer: F

   (c) **T F** The root directory of a partition in a Unix system is named "/".
   Answer: T

   (d) **T F** There is only one MBR (master boot record) on a disk drive, but there could be several boot sectors.
   Answer: T

   (e) **T F** A context switch from one process to another can be accomplished without executing OS code in kernel mode.
   Answer: F

   (f) **T F** An advantage of implementing threads in user space is that they don't incur the overhead of having the OS schedule their execution.
   Answer: T

   (g) **T F** Deadlock can never occur if no process is allowed to hold a resource while requesting another resource.
   Answer: T

   (h) **T F** In round robin scheduling, it is advantageous to give each I/O bound process a longer quantum than each CPU-bound process (since this has the effect of giving the I/O bound process a higher priority).
   Answer: F

   (i) **T F** For machines with 32-bit addresses (i.e. a 4GB address space), since 4GB physical memories are common and cheap, virtual memory is really no longer needed.
   Answer: F

   (j) **T F** A TLB miss could occur even though the requested page was in memory.
   Answer: T

2. Virtual Memory

   (a) Suppose you had a computer that supported virtual memory and had 32-bit virtual addresses and 4KB (2^{12} byte) pages. If a process actually uses 1024 (2^{10}) pages of its

virtual address space, how much space would be occupied by the page table for that process if a single-level page table was used? Assume each page table entry occupies 4 bytes.

There are $2^{32}/2^{12} = 2^{20}$ pages in the address space. Therefore, a single level page table would have 2^{20} entries, where each entry is 4 bytes. Therefore, the page table would require $4 * 2^{20}$ bytes = 4 MB. This is independent of how many pages are actually being used.

(b) Suppose, in the same machine as above, a two-level page table was used, such that the first level page table was four times the size of each second level page table. Do you have enough information to know exactly how much space is occupied by the two-level page table for that process? Explain.

No. How the used pages are distributed across the address space of the process will determine how many second-level page tables are needed.

(c) If your answer to part (c) was "yes", compute the space occupied by the two-level page table for that process. Otherwise, compute the minimum amount of space that could possibly be required for the two-level page table for that process and compute the maximum amount of space that could possibly be occupied by the two-level page table for that process. Be sure to show your work.

First, we need to compute the sizes of the first and second level page tables. Since there are 2^{20} pages and the first-level page table is four times the size of each second level page table, the first level page table must have 2^{11} entries and each second level page table must have 2^9 entries (since their product must equal 2^{20}). Further, since each page table entry occupies 4 bytes (2^2 bytes), the first level page table occupies $2^{11} * 2^2 = 2^{13}$ bytes (8KB) and each second level table occupies $2^9 * 2^2 = 2^{11}$ bytes (2KB).

Since 2^{10} second-level page table entries are needed, in total, to support the 2^{10} pages in use, there needs to be, at a minimum $2^{10}/2^9 = 2$ second-level page tables. These two second-level page tables, along with the first level page table, would occupy a total of $(2 * 2^{11}) + 2^{13}$ bytes = 4KB + 8KB = 12KB.

At worst, each of the 2^{10} pages used by the process could be mapped by a different second-level page table (e.g. if the pages in use were spaced equally across the entire address space). In that case, 2^{10} second-level page tables would be required in addition to the first level page table, so the total space occupied by the page tables would be $(2^{10} * 2^{11}) + 2^{13}$ bytes = 2MB + 8KB.

(d) In an virtual memory system that supports the NRU (Not Recently Used) page replacement algorithm, how are the R and M bits used to determine which page to evict from RAM, if necessary? What hardware support is needed to maintain the R and M bits?

For a page in memory, the R bit for a page is set upon every access to that page. The M bit for a page is set when that page is written to. The R bits for all the pages in memory are occasionally cleared. When memory is full and a new page needs to be brought into memory, the NRU algorithm chooses a page to evict from memory in the following order of priority: 1) R=0, M=0; 2) R=0, M=1; 3) R=1; M=0; 4) R=1; M=1. The hardware must support the setting of the R and M bits of each page(since you don't want to incur the cost of having the OS do it). Depending on how often the R bits are cleared, you may want to have the hardware do this as well.

(e) Suppose you had a (very) simple virtual memory system with a physical memory that consists of only 4 page frames. Suppose also that R bits are supported and are cleared after every 6 memory accesses.

Assuming that memory is initially unoccupied, give a sequence of page accesses that, if generated by the processor, would cause a page to be evicted from memory – but would cause a *different* page to be chosen for eviction by each of following page replacement algorithms:

- NRU
- Second Chance
- LRU (least recently used, assume it is fully supported by the hardware).

Your answer should be of the form "0,2,9,1,0...", in this case indicating that page 0 was accessed first, page 2 was accessed next, etc. Your sequence should be fairly short. You should also indicate which page would be chose for eviction by each of the above page replacement algorithms.

One possible answer is the following sequence of page accesses: 3, 0, 1, 2, 3, 0, 1, 4.

Which page to evict when page 4 needs to be brought into memory depends on the page replacement algorithm as follows:

- **NRU: The request to page 4 would require the eviction of one of the four pages (0,1,2,3) already in memory. Since the R bits are reset after 6 memory accesses, only the R bit for page 1 would be set. Therefore, the OS would either choose one of the other pages randomly or might choose the lowest numbered page to evict (0).**
- **Second chance: Second chance would pick the page to evict that was first loaded into memory (like FIFO), unless the R-bit for that page was set. Since the R bits are cleared after 6 accesses, and only page 1's R bit would be set, page 3 would be chosen to evict.**
- **LRU: Since page 2 is the least recently accessed, LRU would choose page 2 to evict.**

3. Files

(a) Suppose a computer has a file system for a 128GB (2^{37} byte) disk, where each disk block is 8KB (2^{13} bytes). If the OS for this computer uses a FAT, what is the smallest amount of memory that could possibly be used for the FAT (assuming the entire FAT is in memory)? Explain.

There has to be a FAT entry for each disk block. Since the disk is 2^{37} bytes and a disk block is 2^{13} bytes, the number of disk blocks (and thus the number of FAT entries) is $2^{37}/2^{13} = 2^{24}$. Since there are 2^{24} entries, a block number (disk address) requires a minimum of $\log(2^{24})$ bits = 24 bits = 3 bytes. In this case, the minimum amount of space occupied by the FAT is the number of entries (2^{24}) times the 3 bytes per entry, namely $2^{24} * 3$ = 16MB * 3 = 48MB.

(b) Suppose that on a different computer, the OS uses i-nodes and each disk block is 4KB (2^{12} bytes). Assume that an i-node contains 12 direct block numbers (disk addresses) and the block numbers for one indirect block, one double indirect block, and one triple

indirect block. Assume also that a block number is 4 bytes. What is the largest possible file on that computer (assuming the disk is large enough).

Since a disk block is 4KB (2^{12} bytes) and a block number is 4 (2^2) bytes, there are 2^{10} = 1024 entries per indirect block. Therefore, the maxium number of blocks of a file that could be referenced by the i-node is $12 + 2^{10} + (2^{10})^2 + (2^{10})^3 = 12 + 2^{10} + 2^{20} + 2^{30}$**. Thus, the maxium size of the file would be** $(12 + 2^{10} + 2^{20} + 2^{30}) * 2^{12} = (12 * 2^{12}) + 2^{22} + 2^{32} + 2^{42}$ **= 48KB + 4MB + 4GB + 4TB.**

(c) On the computer from part (b) above, suppose you wanted to create a new file of the maximum size (that you computed for the answer to (b)). How many free blocks on the disk would there need to be in order to create that file?

As discussed above, you would need $12 + 2^{10} + 2^{20} + 2^{30}$ **blocks just to contain the data in the file itself. However, you would also need free blocks to use for your single indirect, double indirect, and triple indirect blocks. The total of those blocks is** $1+(1+2^{10})+(1+2^{10}+2^{20}) = 3+(2*2^{10})+2^{20}$**. Adding together the file blocks and the indirect blocks gives** $(12+2^{10}+2^{20}+2^{30})+(3+(2*2^{10})+2^{20}) = 15 + (3 * 2^{10}) + (2 * 2^{20}) + 2^{30}$**.**

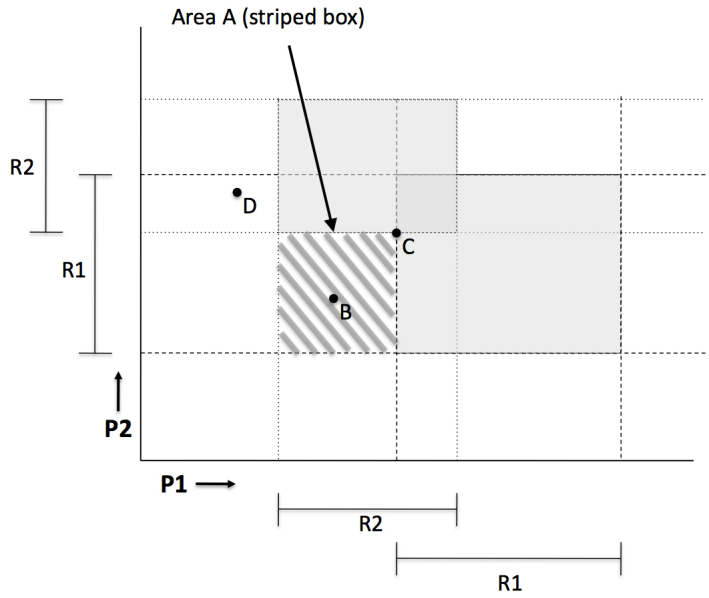
(d) As discussed in class, a hard link is an entry in a directory that refers to the same file as another directory entry (in the same directory or not). If you were to try to implement hard links in an OS that uses a FAT, what would be contained in the directory entry for a hard link?

Using a FAT, each entry in a directory contains the first block number for that file. This is because the blocks of a file is determined by its first block, i.e. the entry point into the FAT. Thus, a hard link might be implemented in a FAT system by having a directory entry contain the same first block number as another directory entry.

(e) If you didn't change the structure of a FAT (as found, for example, in DOS), what would be the problem with your hard link implementation?

The problem is that a FAT entry, unlike an i-node, doesn't keep track of how many directory entries refer to that entry. Thus, if a user deletes a file – even if another hard link to that file exists – the block of that file will be put back on the list of free blocks (and eventually get overwritten).

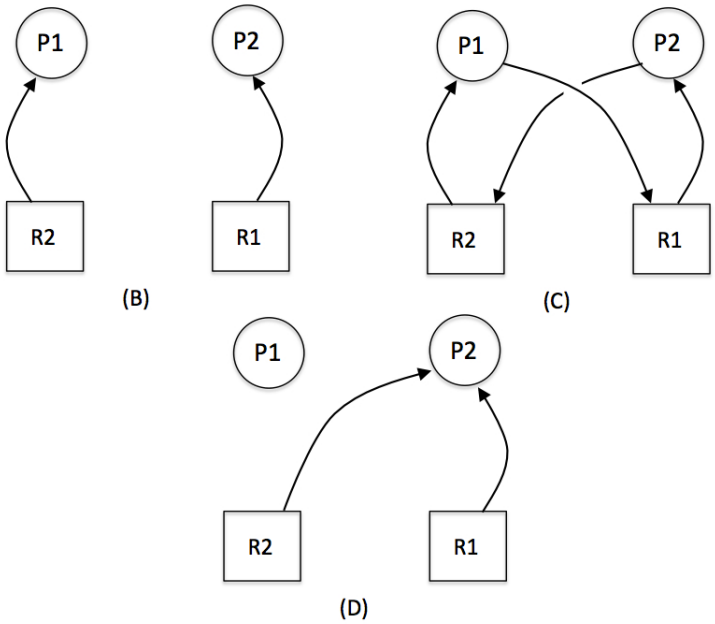
4. Deadlock Consider the following resource trajectory graph for a system with two processes (P1 and P2) and two resources (R1 and R2).



(a) What does the area identified as Area A indicate?

A region where the processes are not deadlocked, but will inevitably be deadlocked (when they hit point C).

(b) Draw three resource allocation diagrams (the ones with circles, squares and arrows), one for each of the points indicated above as B, C, and D.



(c) If you drew the same resource allocation graphs for both B and C, explain why. If you

drew different resource allocation graphs for points B and C, explain why. What does that tell you about the usefulness of using resource allocation graphs to avoid deadlock?

They're different because B is not actually a deadlocked state (no cycle), whereas C is a deadlocked state. The problem with using resource allocation graphs to avoid deadlock is that, as for point B, they can't be used to detect when deadlock is possible, or even inevitable. They can only distinguish between deadlock and non-deadlock.

(d) In what way is the banker's algorithm "pessimistic"?

It makes a worst case assumption about how processes will request and release resources. It still ensures that, even in the worst case, there is still a path that allows all the processes to eventually have their resource requests satisfied and thus terminate.

(e) Write, in C, three very simple, non-looping programs that use shared binary semaphores, such that deadlock among all three could occur if nothing was done to prevent it. Assume the programs don't use any other resource that could cause deadlock. Give a scenario of execution in which your three programs actually deadlock.

Here is just one of many possible answers. Assuming there are three binary semaphores, `S1`, `S2`, and `S3`, the programs are,

```
void prog1()
{
  down(S1);
  down(S2);
  down(S3);
  critical_section();
  up(S3);
  up(S2);
  up(S1);
}
```

```
void prog2()
{
  down(S2);
  down(S3);
  down(S1);
  critical_section();
  up(S1);
  up(S3);
  up(S2);
}
```

```
void prog3()
{
  down(S3);
  down(S1);
  down(S2);
  critical_section();
  up(S2);
  up(S1);
  up(S3);
}
```

A deadlock scenario is when `prog1` executes `down(S1)`, followed by a context switch which allows `prog2` to execute `down(S2)`, followed by a context switch that allows `prog3` to execute `down(S3)`. After that, each program will block when executing its next statement (another `down(...)` operation) – which leads to deadlock.

(f) Assume that each program has to declare, when it starts, which semaphores it will be using. In what way would the bankers algorithm, assuming the OS used it, prevent the deadlock in the scenario you gave in your previous answer?

Since each of the above programs will have declared that it will be using all three semaphores, once `prog1` has succesfully executed `down(S1)`, any successful `down` operation by the other programs would result in an unsafe state, therefore the OS would cause a `down` operation by `prog2` or `prog3` to block. In fact, `prog2` and `prog3` would block until `prog1` has finished, at which time whichever program is unblocked first would run to completion before the last program could unblock.

5. I/O

(a) Describe the elevator disk scheduling algorithm.

Given a set of requests for disk blocks, the controller services those requests by having the disk head move in a single direction, satisfying the requests for blocks on cylinders encountered along the way (i.e choosing the next block requiring the shortest seek in the current direction from the current head position). When the block request on the last requested cylinder in the current direction has been satisfied, the controller reverses the current direction of the head.

(b) What is the advantage of the elevator algorithm over the shortest-seek-first algorithm?

In the presence of a stream of incoming block requests, the shortest seek first algorithm, which always chooses the block request to satisfy that requires the least movement of the head, could lead to starvation of requests for blocks that are far from the current head position.

(c) When a user uses the mouse to "drag" an object across the screen, what is the sequence of events that the mouse controller will generate to communicate with the CPU? What data is communicated with each event?

The mouse controller will generate an event (e.g. an interrupt) indicating a mouse-button-pressed event, specifying which mouse button was pressed. Then it will generate a series of events indicating that the mouse has moved, specifying each movement by Δx and Δy displacement values. Finally, it will generate a mouse-button-released event, specifying the mouse button that was released.

6. Extra Credit: Describe how a modern optical mouse controller determines the direction and distance that the mouse has been moved (I discussed this in detail in class).

The controler will capture a rapid sequence of images of the surface under the mouse. The surface is lit by an LED shining light downwards at an angle, so that irregularities in the surface cause shadows. Each successive image is compared to the previous image, by overlaying images at successive relative offsets (in both the X and Y directions). The offset that minimizes the differences (e.g. by performing a least-squares computation) between the successive images indicates the direction and magnitude of the movement, which will be reported by the mouse controller.