CS-537: Midterm Exam (Fall 2004)

Midterm Harder

Please Read All Questions Carefully!

There are eight (8) total numbered pages.

Please put your student ID (but NOT YOUR NAME) on every page.

Name and Student ID: ____

Grading Page

	Points	Total Possible
Part I: Short Answers		$(12 \times 5) \rightarrow 60$
Part II: Long Answers		$(2 \times 20) \rightarrow 40$
Total		100

Part I: Short Questions

The following questions require short answers. **Each of 12 is worth 5 points (60 total)**.

1. In a traditional single-threaded process, we have both a **stack** and a **heap**.

    - What is the stack used for?

    - What is the heap used for?

    - If, for some reason, you could only have one of these two, which would you pick and why?

2. Assume you have a free list that consists of the following free chunks, in this order from the head of the list: 10 bytes, 20 bytes, 40 bytes, and 10 bytes. Then assume you get the following allocation requests: allocate 10, allocate 15, allocate 10, allocate 35.

    - Using **first fit allocation**, will all requests succeed?

    - Using **best fit allocation**, will all requests succeed?

    - In general, which is better, first fit or best fit?

3. With **dynamic relocation**, the hardware has a **base** register and a **bounds** register, which it uses to support multiprogramming.

    - Imagine if you just had a **base** register; what functionality do you lose with the loss of the bounds register?

    - Imagine if you just had a **bounds** register; what functionality do you lose with the loss of the base register?

    - In a multiprogrammed system, if you could only have one such register, which would you choose, **base** or **bounds**? Why?

4. This question is about **external fragmentation**.

    - Define it.

    - Give an example of where it occurs.

5. Name and describe **two** advantages that **segmentation** has over simple **dynamic relocation**:

    - Advantage #1:

    - Advantage #2:

6. Envision a system that uses **pure paging** (i.e., no segmentation) and a hardware **TLB**. Also assume the the TLB is **software managed**, i.e., any updates to the TLB are handled by the operating system.

    - What happens on a **TLB hit**?

    - What happens on a **TLB miss**?

    - What happens on a **page fault**?

7. This question is about the contents of a typical TLB.

   - Sometimes a TLB will contain two entries that have the same **physical address** – when?

   - Sometimes a TLB will contain two entries that have the same **virtual address** – when?

   - Do either or both of these cases require extra hardware support from the system to work properly?

8. In this question, we discuss the **clock** replacement strategy.

   - Describe how clock works. What hardware support is needed? What software structures must be kept?

   - Can clock ever behave exactly like “perfect” LRU? (describe)

9. This question is about physical addressability in a system that uses **paging**. Let’s say we have a **20-bit virtual address**, with a **4 KB page size**.

   - Let’s assume that the system we’re running upon has a maximum of 1 GB of physical memory. How big is each page table? (assume 2 extra bits of information are needed beyond the usual stuff).

   - Let’s assume a different system we’re running upon has a maximum of 64 KB of memory. How big is each page table? (again assume the 2 extra bits).

   - Which of the preceding two cases is worse, having more physical memory than your process can address, or less? Why?

10. In this question, we explore page cache replacement strategies.

    Assume you have the following page reference stream: A, B, C, D, A, B, E, A, B, C, D, E.

    - Assuming a page cache of size **3 pages** and a **FIFO** replacement policy, how many misses will there be?
    - Assuming a page cache of size **4 pages** and a **FIFO** replacement policy, how many misses will there be?
    - Does the comparison between the 3-page and 4-page caches surprise you in any way? Why?

11. **Thrashing** occurs when more memory is being actively utilized than the system contains. When talking about thrashing, one often refers the **working set** of a process.

    - Define the "working set" of a process.
    - If a system is thrashing, how can we try to reduce thrashing **within the OS**? (i.e., how would we change the OS?)
    - If a system is thrashing, how can we try to reduce thrashing **with hardware (of some kind)**? (i.e., how would we change the hardware?)

12. Assume you have a **physical address** P. Let's say this is in a system that has a typical **linear page table** structure.

    - How would you find out which virtual address(es) are mapped to P?
    - What kind of data structures might you add to speed up this process?

Part II: Longer Questions

The second half of the exam consists of two longer questions, **each worth 20 points (total 40)**.

1. **Improving your memory.**

   In this question, we consider a new hardware system that has two types of memory. **Primary memory** is the type of memory attached to the system in the typical manner (e.g, on a memory bus). **Secondary memory** is additional memory that you add for more capacity, but it's added onto the I/O bus. The implication of this structure is that **secondary memory accesses are slower than primary memory accesses** (but still of course are much faster than disk accesses).

   Note that secondary memory is addressed just like primary memory. Specifically, the hardware looks like it just has one big physical memory. However, the lower part of this "physical" memory is the faster primary memory, and the upper part is slower secondary memory. The OS can also easily find out where the boundary is, and thus can be intelligent in how it uses primary versus secondary memory.

   We now discuss how OS design must change to accommodate the new hardware.

   (a) Let's say you decide to use a combination of segmentation and paging in the OS for this system. Describe how a combined segmentation and paging approach works, and how this hybrid approach improves on strict paging and strict segmentation.

   (b) In this system, you of course will have page tables. Should you change how the page tables are structured in this new system? (if so, how, if not, why not)

   (c) You also decide to use LRU replacement for the page cache. Describe how LRU replacement works.

   (d) Should you change how your basic LRU algorithm works in this new system? (if so, how, if not, why not)

2. **Size does matter.**

   In this question, we try to understand the issues that surround the choice of **page size** in a system. In particular, we will discuss some new systems that have support for **two** pages sizes, one **"small"** sized (say 4 KB), and one that is **"big"** (say 1 MB).

   (a) If we just have a single page size, and it is quite **big**, what are the possible negative consequences? In contrast, what negative consequences arise if our page size is **too small**?

   (b) Using **big pages** can improve performance. Which **hardware resources** of the system are better utilized with big pages?

   (c) To support multiple page sizes, some aspects of the **page table** must **change**. Describe the most important changes needed, and how you would implement them. Assume a simple **linear page table**.

   (d) Now imagine a scheme that tries to make use of big pages where possible. Specifically, the OS first hands out small pages when a process asks for more memory. Then, the OS periodically tries to **convert** batches of these small pages into big pages. Describe how the OS would do this – what must be true to convert small pages into a big page? What (software/hardware) structures must be updated?