Second Exam name: _____

Question 1: Translation Look-aside Buffer

(a) Describe the TLB. Include its location, why it is located there, its contents, and how it is used.

A TLB is a cache of popular virtual-to-physical address translations. A TLB lives in hardware inside the processor chip. On a virtual memory access, the hardware looks at the TLB first. If there is a cache miss, a page table is accessed directly or through a page table directory. A TLB is implemented as a hash table for fast access. Only pages belonging to running processes have TLB entries.

(b) I get this after executing `x86info -c`:

```
Data TLB: 4KB or 4MB pages, fully associative, 32 entries
```

What does this mean?

The TLB is for pages associated with data, not instructions, and has room for 32 page table entries for page sizes of either 4KB or 4MB. A virtual address may hash to any of the 32 entries.

(c) The effectiveness of TLBs depends on locality, locality, and ___locality_____

Question 2: Page Tables

In the following assume a 32 bit virtual address space and page size equal to 4096 bytes

(a) How many page table entries are required, per process, maximum?

$2^{32}/2^{12} = 2^{20} \approx 1$ million

(b) How much total memory is required per table table, maximum (assume single level page tables)?

$2^{20} * 4 = 4$MB

(c) How much total memory is required for page tables if 50 processes are running (assume single level page tables)?

$4 * 50 = 200$MB

(d) Now use a 2-level page table where the upper 10 bits of a virtual address index the first table and the next 10 bits index the second table. Suppose page table memory is allocated in pages. Suppose 50 processes are running and each has a working set of 10 pages and suppose addresses of pages in a working set have the same upper 10 bits. How many pages would have to be used for the page tables?

Since the upper 10 bits on all virtual addresses is the same, there is only one valid directory table (first table) entry per process and this points to a single page table section that is indexed from the middle 10 bits of the virtual address. Hence, 2 pages per process are needed for a total of 100 pages.

(e) At most how many pages would have to be used for an inverted page table?

\# PTE $= 2^{32}/2^{12} = 2^{20}$
total bytes for PTEs $= 2^2 * 2^{20} = 2^{22}$
total pages needed for PTEs $= 2^{22}/2^{12} = 2^{10} = 1024$

(f) Suppose a quarter of that number of pages is used for the inverted page table, collisions are accomodated by linked lists (bucket hash), accessing a linked list element takes 60nsec, and computing a hash takes 10nsec. Suppose only pages in working sets produce hits in the inverted page table. For 50 processes, each with 10 page working sets, how much time on the average does it take to translate a virtual address to a physical address?

number of hash table entries $= 2^8 * 2^{12}/2^5 = 2^{15}$.
probability PT address has no collision in table $= (1 - 2^{-15})^{499} \approx .985$
probability PT address has a collision $= 1 - (1 - 2^{-15})^{499} \approx .015$
probability PT address is 2nd in bucket chain = tiny - forget it
Avg lookup time $= 10 + .015 * 60/2... \approx 10.45$nsec

Question 3: Segment Allocation Algorithms

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, worst-fit, and next-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm(s) make(s) the most efficient use of memory?

Process:	212KB	417KB	112KB	426KB
First fit:	500KB	600KB	500KB	stuck*
Best fit:	300KB	500KB	200KB	600KB
Worst fit:	600KB	500KB	300KB	stuck*
Next fit:	600KB	500KB	600KB	stuck*

Best fit wins

* - waits for either the 500KB or 600KB chunk to free up

Question 4: Buddy Allocation

(a) What is so great about the buddy memory allocator?

It's a low overhead mechanism for allocating pages. External fragmentation is low: small allocation holes are not persistent. Compaction in the classical sense is not necessary. Small data structures are used to keep track of which buddies exist

(b) Describe how the buddy allocator works

Two halves of the address space are segments that are buddies. Two halves of each half of the address space are segments that are buddies. Two halves of each quarter of the address space are segments that are buddies and so on until adjacent pages (or unit segments) are buddies. On a request for space from a process the allocator finds a segment of size that is a power of 2 that satisfies the request. This may be done by repetitive splitting from a large segment until a further split would result in a segment that is too small. Where to look for a segment, if there are many, depends on many factors, for example the need to retain some large segments. When memory is returned, if the buddy of the memory is free the returned segment and the buddy are combined to form a larger segment. If the larger segment has a free buddy, the two are again combined, and so on until there is no free buddy.

Question 4: Memory Mapped Files

(a) Describe memory mapped files, their use, and their operation

With memory mapped files data intended for disk or another IO device is read into memory using demand paging instead of directly to the disk or device using system calls. Also, data received from a device is taken by the processor from memory instead of system calls.

(b) What is so good about memory mapped files?

Reads and write are treated as ordinary memory reads and writes. Speeds up access to the resource: a system call takes orders of magnitude longer than a memory access because, for example, seek time and latency may be charged for each system call. Allows several processes to share access to the resource. The mapping can be to the kernel's page cache and therefore not take away from user pages.

(c) What is not so good about memory mapped files?

A 5KB resource maps to two 4KB pages, so there is some waste. When a block of data is loaded in page cache, but is not yet mapped into the processs virtual memory space, page faults may occur. A file larger than the addressable space can have only portions mapped at a time, complicating reads.

Question 5: Demand Paging

(a) Describe page replacement and frame allocation

If a page must be swapped out to make room for a new frame, a victim frame must be chosen and swapped out. A page replacement algorithm is responsible for choosing. Every process should be granted some minimum number of page frames or else it will be competing with itself or other processes for free pages. The number of pages to grant is determined by a frame allocation algorithm. The algorithm tries to prevent thrashing in making its allocations - that is, it may vary a process's working set.

(b) Decribe the second chance algorithm - is this used for frame allocation or page replacement?

All pages are in a circular queue. If the cursor points to a page with reference bit set then reset it (giving it a second chance) and move the cursor to the next page to test again. If all reference bits are set the one the cursor pointed to originally is the victim. This is easily implemented in software.

It is used for page replacement.

(c) When does thrashing occur?

When the total minimum number of working set pages of running processes is greater than the number of available pages.