Name: **Student Id:**

CMPSCI 377: Operating Systems

Exam 2: Memory Management and File Systems

November 18, 1999

General instructions:

- This examination booklet has 9 pages.
- Do not forget to put down your name and student number on the exam books.
- The exam is closed book and closed notes.
- Explain your answers clearly and be concise. Do not write long essays.
- You have 90 minutes to complete the exam. Be a smart test taker, if you get stuck on one problem go on to the next. Don't waste your time giving details that the question does not request.
- Show your work. Partial credit is possible, but only if you show intermediate steps.
- Good luck.

---

1. **Contiguous Memory Allocation** (20 points)

   (a) **Relocation**

   - (4pts) Explain the difference between static and dynamic relocation.
     Static relocation: at load time, the OS adjusts the addresses in a process to reflect its position in memory. Once a process is assigned a place in memory and starts executing it, the OS cannot move it.
     Dynamic relocation: The OS uses a relocation register to store the starting location (base) of the process. The hardware adds the relocation register (base) to virtual address to compute a physical address. Thus, virtual to physical address translation is done at execution time. Processes can be moved at any time by merely changing the base address in the relocation register.
   - (6 pts) List the advantages and disadvantages of dynamic relocation.
     Advantages:
     – OS can easily move a process during execution.
     – OS can allow a process to grow over time.
     – Simple, fast hardware: two special registers, an add, and a compare
     Disadvantages:
     – Slows down hardware due to the add on every memory reference.

- Can't share memory (such as program text) between processes.
- Process is still limited to physical memory size.
- Degree of multiprogramming is very limited since all memory of all active processes must fit in memory
- Complicates memory management.

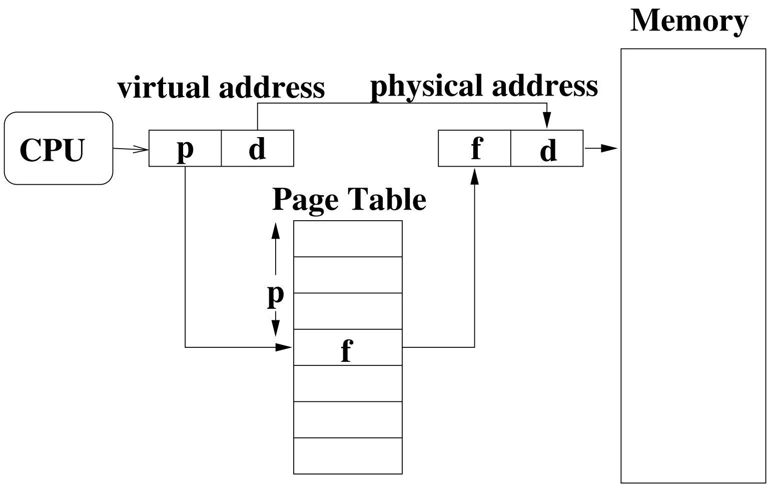
(b) **Fragmentation**

- (4pts) What are external and internal fragmentation?
  External Fragmentation:
  Frequent loading and unloading programs causes free space to be broken into little pieces. External fragmentation exists when there is enough memory to fit a process in memory, but the space is not contiguous
  Internal fragmentation:
  Internal fragmentation is caused when a process is allocated more space that it requested (or needs), effectively wasting space within the process. This typically happens when the process requests a certain amount of memory and the operating system allocates a larger chunk because it is more expensive to keep track of very small pieces of memory.
- (6pts) Consider a contiguous memory allocation scheme that uses the first-fit policy. Assume that the size of physical memory is 64MB. On an average, how much memory do you expect to lose to: (i) external fragmentation, and (ii) internal fragmentation? Explain your answer.
  50-percent rule: For every 2N allocated blocks, N blocks are lost due to fragmentation (i.e., 1/3 of memory space is wasted) Hence, 1/3 of 64MB (i.e., 21.3MB) is wasted to external fragmentation.
  In theory, contiguous memory allocation does not suffer from *any* internal fragmentation, as a process can be allocated exactly what it needs. In practice, if the OS allocates more memory than requested to avoid tracking small holes, then a small fraction may be lost to internal fragmentation.

2. **Paging** (20 points)

(a) Consider a memory management scheme that uses pure paging (i.e., the entire process is in memory at all times). Let the size of physical memory be 1024 bytes, and let us assume that the size of a memory frame (and a page) is 16 bytes.

- (4 pts) What is the maximum size of a process in such a system? How large should the page table be to accommodate this process?
  In the entire process is in memory at all times, then the maximum size of the process can be at most the size of physical memory (i.e., 1024 bytes). (ignoring the space taken up by the OS and OS data structures such as PCBs, page tables etc).
  Since page size is 16 bytes, we can have at most 64 pages for a 1024 byte process. So the page table should have 64 entries.
- (10 pts) Draw a picture showing how a virtual address generated by the CPU is translated to a physical address. Indicate clearly how many bits are used for the page number and how many for the offset. Assume byte level addressing. (Note: $1024 = 64 * 16$).
  Since we have 64 pages in the page table, we need 6 bits for the page number. Each page is 16 bytes, so we need 4 bits for the offset (assuming byte-level addressing). Hence, p=6 bits, d=4bits.



(b) (6 pts) What is a *translation look-aside buffer (TLB)*? Do you need a TLB for correct execution of programs?

TLB: a fast fully associative memory that stores page numbers (key) and the frame (value) in which they are stored. TLB acts as a cache for the page table and stores the most recently accessed page table entries to speed-up future lookups.

TLB is not needed for correct execution of programs; it merely speeds up execution of programs.

3. **Segmentation and Segmented Paging** (30 points)

(a) (10 pts) *Segmentation:* What is the purpose of using segments within a program? What hardware features do you need to implement a pure segmentation scheme?

Segments take the user's view of the program and gives it to the OS. User views the program in logical segments, e.g., code, global variables, stack, heap (dynamic data structures), not a single linear array of bytes. The compiler generates references that identify the segment and the offset in the segment, e.g., a code segment with offset = 399 Thus processes thus use virtual addresses that are segments and segment offsets.

Hardware support: segment table with multiple base/limit registers.

(b) (20 pts) *Performance of Segmented Paging.* This question asks you to compute the effective memory access time (*ema*) in symbolic terms for different hardware implementations of segmented paging with and without virtual memory. Let *ma* be the time to read or write a value in memory. Define other terms as you need them. First assume that the process, the segment table and the page table fit and stay in memory while the program executes, and there is no TLB.

i. (4 pts) What is the *ema* for this system?
$ema = 3 * ma$, memory reference plus page table look up plus segment table lookup.

ii. (4 pts) Now assume there is a TLB. What is the *ema* for this system? Include *l*, the time for the TLB look up in your equation.

$$ema = x * (2 * ma + l) + (1 - x) * 3 * ma$$

where x be the probability of a TLB hit. $x * (2ma + l)$ is the time with a TLB hit, and $(1 - x) * 3 * ma$ is the miss.

iii. (6 pts) Now consider a virtual memory system with a TLB where the process can grow larger than the memory, and the pages may not always be in memory. What is the *ema* now? Assume that the page table and the segment table are always in memory.

$$ema = x * (2 * ma + l) + (1 - x)(y * 3 * ma + (1 - y)(pf + 3 * ma))$$

where x be the probability of a TLB hit, y the probability the referenced page is in memory, and pf the page fault time. The TLB miss time is now the sum of probability of a page hit in memory ($y * 3 * ma$), and the page miss ($pf + ma$) which includes the memory access as well as the page fault time. The page fault time includes the time to read the page in, and write the replaced page on to disk if it was modified

iv. (6 pts) Now what is the *ema* for the same system except that the page table entry itself may not be in memory? Assume that the segment table is always in memory.

$$b = y * 3 * ma + (1 - y)(pf + 3 * ma)$$

$$ema = x * (2 * ma + l) + (1 - x)(z * b + (1 - z)(pf + b))$$

where z is the probability of no page fault of the page table look up. $z * b$ is thus the hit. For the miss, we add the page fault time for the look up, the memory access time for the page look up is already included in b.

4. **Page Replacement Algorithms** (15 points)

Consider a demand paged virtual memory scheme that used the LRU page replacement algorithm. Assume that a certain process has five pages A,B,C,D and E and is allocated three page frames.

- (6pts) Indicate the contents of each page frame for the reference string: $ABCDABEABCDE$.

	A	B	C	D	A	B	E	A	B	C	D	E
frame 1	A	A	A	D	D	D	E	E	E	C	C	C
frame 2		B	B	B	A	A	A	A	A	A	D	D
frame 3			C	C	C	B	B	B	B	B	B	E

- (6pts) Indicate the contents of each page frame if four page frames are allocated the the process.

	A	B	C	D	A	B	E	A	B	C	D	E
frame 1	A	A	A	A	A	A	A	A	A	A	A	E
frame 2		B	B	B	B	B	B	B	B	B	B	B
frame 3			C	C	C	C	E	E	E	E	D	D
frame 4				D	D	D	D	D	D	C	C	C

- (3pts) How many page faults do you see in each case? Why does the number of page faults show this behavior (i.e., an increase or a decrease)?

  We see 10 and 8 page faults when three and four frames are allocated to the process, respectively.

  The number of page faults decreases, since for LRU, the set of pages in memory would be the n most recently referenced pages. If the number of frames in increased, these n pages will still be the most recently referenced and so will still be in memory (i.e., the pages in memory is always a subset of the pages in the previous case). Thus, LRU, unlike FIFO, does not suffer from Belady's anomaly.

5. **File Systems** 15 points

- (5 pts) What must the operating system do when a program executes the *open* system call?
  - Check if the file is already open by another process. If not,
    * Find the file.
    * Copy the file descriptor into the system-wide open file table.
  - Check the protection of the file against the requested mode. If not ok, abort
  - Increment the open count.
  - Create an entry in the process's file table pointing to the entry in the system-wide file table. Initialize the current file pointer to the start of the file.
- (5 pts) Consider a multi-level directory structure that allows soft links. What happens if you exectute the following commands:

```
ln -s reader.cc reader2.cc
rm reader.cc
```

  where the `ln -s A B` command creates a soft link from file B to file A and the `rm` command deletes a file.

  The first command creates a soft link from reader2.cc to reader.cc. The second command deletes the original file (reader.cc), causing reader2.cc to point to a file that no longer exists. This dangling pointer causes an error whenever reader2.cc is accessed (similar to creating a shortcut in Windows and deleting the original object).
- (5 pts) List the advantages and disadvantages of a multi-level index file descriptor.

  Advantages:
  - Supports incremental file growth
  - Small files can be accessed efficiently, large files pay a penalty due to indirect pointers. Since most files are small, this strategy optimizes for the common case.

  Disadvantages:
  - Indirect access is inefficient for random access to very large files.
  - Lots of seeks because data is not contiguous