Name: **Student Id:**

CMPSCI 377: Operating Systems
Exam 2: Memory Management
April 20, 2010

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**

   - (5pts) Explain the difference between static and dynamic relocation.

   Static relocation is where all physical addresses used in a program are computed at load time. As a result, the memory allocated to a process remains in a fixed location throughout its lifetime.
   In dynamic relocation, virtual addresses are used within the process, and they are translated to physical addresses as each memory access is made. This means that the region of memory that a program occupies can change over time because the destination of each virtual address can be remapped (typically by adjusting the base and limit registers).

- (5 pts) List three advantages of static relocation over dynamic relocation. Is any special hardware necessary to support static relocation? Explain your answer.

  Benefits of static relocation include:
  - Faster since extra memory reads and adds are not required for each access
  - Cheaper since no additional hardware support is required (e.g. no base and limit registers, adder or comparator)
  - Allows for simpler memory management code within the OS
  - Can be used to support sharing easily, since the location of a process is fixed once it has been started

  No extra hardware is required for static relocation schemes.

(b) **Fragmentation**

- (4pts) What are external and internal fragmentation?

  External fragmentation refers to "holes" in memory that are formed between processes as they are allocated and removed. When there is technically sufficient free memory for a new process, but it cannot be allocated as a contiguous chunk, then the system is considered to be externally fragmented.
  Internal fragmentation refers to wasted space within the memory space of a process. For example, a process may be allocated a larger region than it actively uses, or in a paging based system, a process may not completely fill up each of its pages.

- (6pts) Consider a memory allocation scheme that uses the segmentation 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.

According to the 50/50 Rule, we expect to lose about 1/3 of the memory space to external fragmentation, thus about 21MB will be wasted in our 64MB system.

Since a system that only uses segmentation must allocates memory in contiguous chunks, we expect very little memory to be wasted due to internal fragmentation. This is because when memory is allocated contiguously, it is typically granted the precise amount of memory it needs, preventing internal fragmentation from being a major concern.

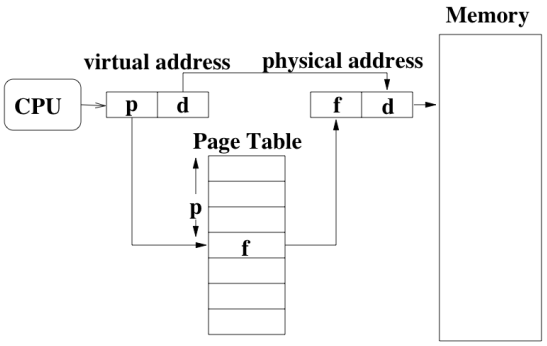
2. **Paging** (20 points)

(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?

  The page table must contain $1024/16 = 64$ entries in order to address all of the memory. The maximum size of a process will be the full size of memory – 1024 Bytes, or 64 pages.
  Note, this assumes that all of memory can be used for applications and that there are no OS overheads.

- (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: $1024 = 64 * 16$).



  - size of page $= 16 = 2^4 \rightarrow 4$ bits for d (offset)
  - number of pages $= 64 = 2^6 \rightarrow 6$ bits for p (page number)

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

A TLB is a fast fully associative memory that is found inside the MMU. The TLB is used to cache address translations from virtual to physical addresses, and is crucial to making these lookups be efficient. However, a TLB is only a performance operation, and thus it is NOT required for correctness of execution.

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 are used to layout memory according to the programer's view of a process. This means that memory is divided up into segments, one for the code, heap, stack, etc. This permits a more organized layout within memory, and provides benefits such as: (i) faster startup time since an application knows which segments to load first, (ii) simplified sharing since entire segments can be shared rather than pages, and (iii) easier process growth since segments can be grown individually.

Implementing segmentation requires base and limit registers for each segment—data that is stored within the segment table. This information is used along with an adder and a comparator to calculate physical addresses and verify that they are valid within the segment.

(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?

$ema = 3 * ma$

The *ema* will be $= 3 * ma$ because three memory accesses are required—one for the segment table, one for the page table, and one for the actual memory request.

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.

In this case we must account for the probability that either the segment table or the page table will be contained in the TLB. If an entry is in the TLB, then we only must pay the TLB lookup cost (l), otherwise we must pay for the memory access cost (ma). This gives us:

$ema = \Big(p_s * l + (1 - p_s) * ma\Big) \;+\; \Big(p_p * l + (1 - p_p) * ma\Big) \;+\; \Big(ma\Big)$
where:
$p_s =$ probability that segment entry is in TLB
$p_p =$ probability that page table entry is in TLB

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.

$ema = \Big(p_s * l + (1 - p_s) * ma\Big) \;+\; \Big(p_p * l + (1 - p_p) * ma\Big) \;+\; \Big((p_f) * ma + (1 - p_f) * da\Big)$
where:
$p_f =$ probability that the frame is in memory (not disk)
In this case, the first two terms for the segment and page table accesses are the same, but we now must account for the chance that the memory frame we are trying to access is actually stored on disk. As a result, we must include both the cost of accessing disk and accessing memory.

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.

$$ema = \Big(p_s * l + (1 - p_s) * ma\Big) \;+\; \Big((p_t)(p_p * l + (1 - p_p) * ma) + (1 - p_t) * da\Big) \;+\; \Big((p_f) * ma + (1 - p_f) * da\Big)$$

where:
$p_t =$ probability that the page table is in memory (not disk)
This equation is identical to the one above, but it accounts for the probability that the page table is in disk or memory and adjusts the cost of the second term accordingly.

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*

Asterisks indicate page faults, while bold entries are hits.

- (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)?

With three frames, LRU produces 10 page faults. Increasing to four frames, lowers the fault rate to only 8 page faults. Increasing the number of frames never reduces the performance of LRU because the pages kept in memory with a larger number of frames is always a superset of what would be kept in memory with a smaller number of frames. As a result, any page which would have been a hit with three frames will also be a hit with four frames. This is in contrast to other algorithms such as FIFO which can see reduced performance when the number of frames is increased (Belady's Anomaly).

5. **Deadlocks** 15 points

- (10 pts) Consider the following system state for four processes P_0, P_1, P_2 and P_3, and three resources A, B and C. Using Bankers algorithm, determine whether the following state is safe or unsafe. Explain your answer.

	Max			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	0	1	0	0	1			
P_1	1	7	5	1	0	0			
P_2	2	3	5	1	3	5			
P_3	0	6	5	0	6	3			
total				2	9	9	1	5	2

	Need		
	A	B	C
P_0	0	0	0
P_1	0	7	5
P_2	1	0	0
P_3	0	0	2

The state is safe because there exists at least one ordering which can meet the needs of all four processes. For example, P_0 can be run immediately since all of its needs are already met. This frees up the available resources to A=1, B=5, and C=3. At this point, any of the processes other than P_1 can be selected to be run. If P_3 is run next, then the available resources rises to A=1, B=11, and C=6. Now there are sufficient resources for either P_2 or P_1 to be run in any order. Since the only invalid sequence would involve running P_1 first, the state of the system is safe.

- (5pts) List the necessary conditions for a deadlock.

  The conditions required for deadlock to occur are:

  - Mutual exclusion: the system must be using locks to control access to resources
  - Hold and wait: a process must acquire a lock and not release it while waiting to acquire different resources
  - Circular wait: two or more processes must exist, and must be requesting locks in such a way that a circle of dependencies exists where no process can acquire the next lock because another already owns it
  - No Preemption: the system must not support preemption, as this could otherwise be used to break the deadlock