CPSC 3220 – Exam #3 April 23, 2018 Name: __**Answer Key**__

1. (3 points) A virtual memory system that uses paging is vulnerable to external fragmentation.
   Why or why not?

   No. External fragmentation is unusable space in the gaps between variable-sized memory regions; with paging, all memory is allocated in fixed- sized units.

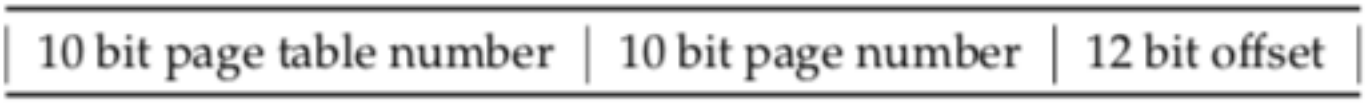
2. (3 points) For systems that use paged segmentation,
   what translation state does the kernel need to change on a process context switch?

   The contents of the segment table.

3. (3 points) Describe the advantages of an architecture that incorporates segmentation & paging over ones that are pure paging.

   Relative to pure paging (either single level or multi-level) segmentation (plus paging) supports variable sized memory regions, with a separate page table for each region.

4. Suppose a machine with 32-bit virtual addresses and 40-bit physical addresses is designed with a two-level page table, subdividing the virtual address into three pieces as follows:
   The first 10 bits are the index into the top-level page table,
   the second 10 bits are the index into the second-level page table,
   and the last 12 bits are the offset into the page.
   There are 4 protection bits per page, so each page table entry takes 4 bytes.



   a. (3 points) What is the page size in this system?

      A 12 bit page offset corresponds to a 4 KB page size.

   b. (3 points) How much memory is consumed by the first and second level page tables and wasted by internal fragmentation for a process that has 64K of memory starting at address 0?

      8 KB
      Mapping 64 KB of memory requires: 1 page frame for the top level page table (the first entry is a pointer to a page table, the rest are invalid) and 1 page frame for the second level page table (the first 16 entries are valid page frames and the remainder are invalid). Since the process size is a multiple of the page size, there is no additional internal fragmentation for representing the program. Thus the space overhead of translation is 2 pages, or 8 KB.

5. (3 points) Describe the advantages of an architecture that incorporates segmentation & paging over ones that are pure segmentation.

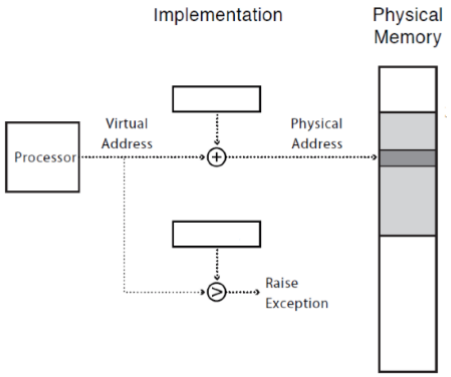
Relative to pure segmentation (either single level or multi-level): paging (plus segmentation) allows the TLB to perform efficient lookups, as the base element (the page or superpage) is fixed size.

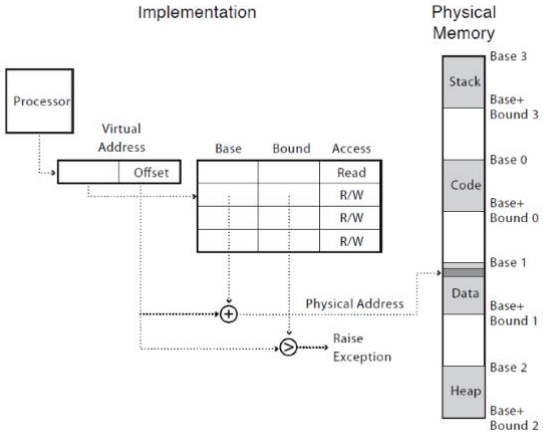
6. (3 points) Describe the difference between external fragmentation and internal fragmentation.

External fragmentation is the unused physical memory space between segments.
Internal fragmentation is the unused physical space inside a frame(page) unit.

Address Translation Concept: Match each theory with its implementation

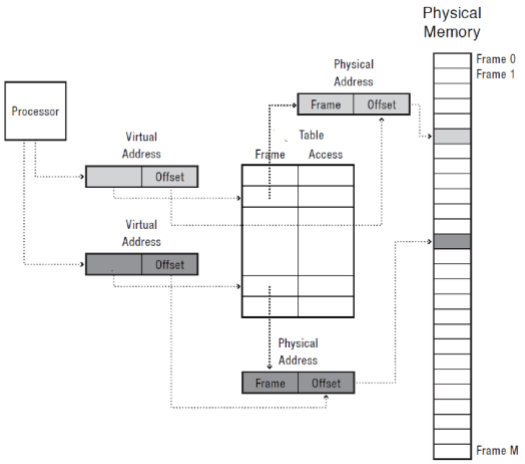
Base+Bound / Segmentation / Paged / Paged Segmentation

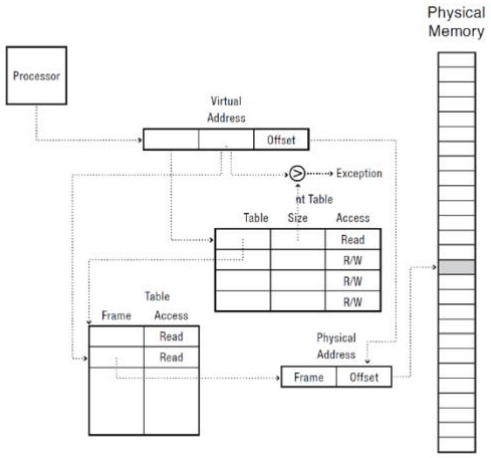




7. (3 points) **Base + Bound**

9. (3 points) **Segmentation**





8. (3 points) **Paged**

10. (3 points) **Paged Segmentation**

Paging / Segmentation Circle one or both of P or S as they apply.

6. (1 point) **P** / S fixed length allocation.

7. (1 point) P / **S** variable length allocation.

8. (1 point) **P** / S can contain internal fragmentation.

9. (1 point) P / **S** can contain external fragmentation.

10. (1 point) **P** / **S** performs a bounds check on the address.

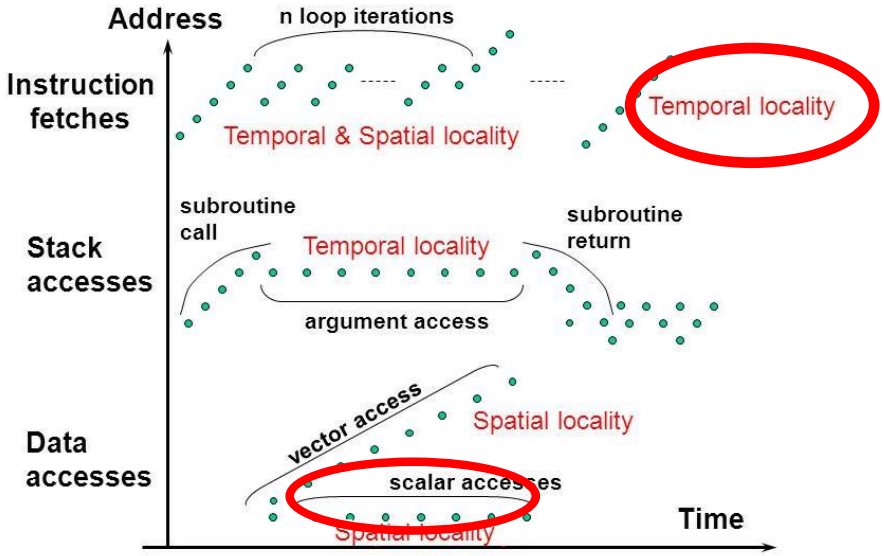
11. (3 points) Describe how Copy-On-Write (COW) works?

When a copy is requested, the file/memory region uses a pointer to reference the "old" data structures. If either the "old" file or the "new" file is written to, then the system will "create" a new memory/file and then update the correct region.

12. (3 points) Describe the difference between Spatial Locality and Temporal Locality?

Spatial Locality occurs from n memory address to n+1 memory address on the next access.
Temporal Locality occurs on 1 memory address over and over and over and ….

13. (4 points) The following diagram has two of the locality labels misplaced.
Circle the two that are misplaced.



14. (4 points) Describe the difference between write-back cache and write-through cache.

A write back cache only writes to physical memory when a "dirty" cache page is evicted.
A write through cache issues a write request to physical memory on every memory write.

15. (3 points) What happens when a working set of pages do NOT fit into physical cache?

A working set of pages is the pages that a set of pages in use by all process and if the physical cache is too small then "thrashing" will occur and the overall system performance will degrade.

16. Suppose an application is assigned 4 pages of physical memory and the memory is initially empty. It then references pages in the following sequence: ABCD EFBF EDBA CADB

a. (4 points) Show how the system would fault pages into the four frames of physical memory, using the FIFO replacement policy.

	A	B	C	D	E	F	B	F	E	D	B	A	C	A	D	B
1	A				E				+				C			
2		B				F		+							D	
3			C				B				+					+
4				D						+		A		+		

b. (4 points) Show how the system would fault pages into the four frames of physical memory, using the LRU replacement policy.

	A	B	C	D	E	F	B	F	E	D	B	A	C	A	D	B
1	**A**				**E**				**+**				**C**			
2		**B**				**F**		**+**				**A**		**+**		
3			**C**				**B**				**+**					**+**
4				**D**						**+**					**+**	

17. Most modern computer systems choose a page size of 4 KB.

a. (3 points) Give one reason why doubling the page size might increase performance.

(any one is valid)

1. If the program exhibits spatial locality (at the new page granularity), then both page n and page n+1 are needed at the same time, reducing page faults.

2. Likewise, page n and page n+1 can be evicted at the same time.

3. Increasing the page size increases the range of addresses that can be translated by a fixed number of entries in the TLB.

4. Fewer pages implies fewer use/modified bits to keep track of.

5. Fewer pages implies less space taken up by page tables, leaving more room for useful pages, decreasing page faults.

b. (3 points) Give one reason why doubling the page size might decrease performance.

(any one is valid)

1. If the program exhibits no spatial locality (at the new page granularity), there will be less room for useful pages, increasing page faults.

2. Likewise, whenever any part of i is referenced, both subpages are marked as recently used, even if the other half is not being used.

3. Internal fragmentation if application memory regions are small com- pared to the new page size.

4. Disk access time is increased slightly, as both pages need to be writ- ten/read from disk, even when only one is needed.

18. For each of the following statements, indicate whether the statement is true or false, and explain why.

a. (3 points) A direct mapped cache can sometimes have a higher hit rate than a fully associative cache (on the same reference pattern).

True, FIFO scan with a working set larger than then the cache size
A direct mapped cache limits the flexibility of the cache replacement algorithm. In each slot, only the most recently referenced cache block can be saved. With a fully associative cache, the cache replacement algorithm is free to evict any cache block, even one that is about to be used.
A FIFO scan that is slightly larger than the cache size, a direct mapped cache will have cache hits on most references, missing only where two blocks from the scan map to the same cache entry.
By contrast, if the fully associative cache uses a least recently used policy, it will cause a cache miss on every reference to a new cache block.

b. (3 points) Adding a cache never hurts performance.

False, a scan pattern that is purely random.
If the reference pattern is purely random, with no locality, or if the replacement policy is a poor fit for the reference pattern, then the cache will not help performance. Instead, it will add useless overhead for consulting the cache that could be avoided.

19. (3 points) In Linux, suppose a process successfully opens an existing file that has a single hard link to it, but while the process is reading that file, another process unlinks that file? What happens to subsequent reads by the first process? Do they succeed? Do they fail?

If the [unlinked] name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed.

20 (3 points) Consider a text editor that saves a file whenever you click a save button. Suppose that when you press the button, the editor simply
(1) animates the button "down" event (e.g., by coloring the button grey),
(2) uses the write() system call to write your text to your file, and then
(3) animates the button "up" event (e.g., by coloring the button white).
What bad thing could happen if a user edits a file, saves it, and then turns off her machine by flipping the power switch (rather than shutting the machine down cleanly)?

write() only guarantees that the updates are in the kernel's memory, it does not guarantee that the updates are flushed to stable storage. Therefore, when the user turns the machine on and looks at the file, she may see the old version of the file, a mix of the old and new versions, or the new version. The first two cases would be considered "bad" since they are probably not what the user expects after a save completes.

21. (3 points) Discussion. Some high-end disks in the 1980s had multiple disk arm assemblies per disk enclosure in order to allow them to achieve higher performance. Today, high-performance server disks have a single arm assembly per disk enclosure. Why do you think disks so seldom have multiple disk arm assemblies today?

(Any one is valid)
Increased cost to produce multiple disk arm assemblies.
Smaller track sizes create more difficulty for multi-headed reads.
One very-fast (single-arm) disk combined with one very-large, but slower, (single-arm) disk works just as well.

22. (3 points) A disk may have multiple surfaces, arms, and heads, but when you issue a read or write, only one head is active at a time. It seems like one could greatly increase disk bandwidth for large requests by reading or writing with all of the heads at the same time. Given the physical characteristics of disks, can you figure out why no one does this?

Head alignment becomes more difficult as track have become smaller.
Tracks are tiny and not perfectly aligned surface-to-surface. So, while reading track *i* on one surface, the disk head is likely not directly over track *i* on other surfaces.

23. Time for a disk access is Seek Time + Rotation Time + Transfer Time. Please define each:

a. (3 points) Seek Time

Seek Time is the time required to move the arm over the desired track and then to settle on the correct track.

b. (3 points) Rotation Time

Rotation Time is the time required for the correct sector to rotate to the read head.

c. (3 points) Transfer Time

Transfer Time is the time required for the data to move from the surface to the disk's memory buffer then to the host's memory buffers.

24. (3 points) What is the difference between SCAN and C-SCAN for disk scheduling of read/write requests on a magnetic disk?

SCAN is an Elevator-based algorithm that keeps track of a direction of the disk arm and works in both directions (servicing requests as the arm moves toward the inner tracks and then servicing requests as the arm moves towards the outer tracks).
C-SCAN is also an elevator based algorithms, but only services requests only as the arm moves from the outer tracks toward the inner tracks. When the arm reaches the last request in the center, it resets back to the outer tracks.

Size	
Usable capacity	2 TB (SLC flash)
Cache Size	64 GB (Battery-backed RAM)
Page Size	4 KB
Performance	
Bandwidth (Sequential Reads from flash)	2048 MB/s
Bandwidth (Sequential Writes to flash)	2048 MB/s
Read Latency (cache hit)	15 μs
Read Latency (cache miss)	200 μs
Write Latency	15 μs
Random Reads (sustained from flash)	100,000 per second
Random Writes (sustained to flash)	100,000 per second
Interface	8 Fibre Channel ports with 4 Gbit/s per port
Power	
Power Consumption	300 W

25. (3 points) Suppose you have a flash drive such as the one described in the Figure above and you have a workload consisting of 5,000 4 KB reads to pages **randomly** scattered across the drive. Assuming that you wait for request i to finish before you issue request i + 1, how long will these 5,000 requests take (total)?

5,000 * 200µs = 1,000,000µs = 1 s
With no concurrency, performance is limited by read latency. Since the reads are to pages scattered randomly, assume they will be cache misses, which according to the figure cost 200µs each.

26. (3 points) Suppose you have a flash drive such as the one described in the Figure above and you have a workload consisting of 5,000 4 KB reads to pages in **sequential order**. Assuming that you wait for request i to finish before you issue request i + 1, how long will these 5,000 requests take (total)?

5,000 * 15µs = 75,000µs = 0.075 s
With no concurrency, performance is limited by read latency. But the reads are to pages in sequential order, assume they will be cache hits, which according to the figure cost 15µs each.

27. (3 points) Suppose you have a flash drive such as the one described in Figure above and you have a workload consisting of 5,000 4 KB reads to pages **randomly** scattered across the drive. Assuming that you issue requests concurrently, using many threads, how long will these 5,000 requests take (total)?

0.05 seconds
With concurrency, we are limited by sustained read IOPs of 100,000 per second,
5,000 / 100,000 is 0.1 seconds
or we are limited by the sustained bandwidth of 2,048,000,000 per second,
(5,000 * 4,000 = 20,000,000); 20,000,000 / 2,048,000,000 = 0.01 seconds

Size	
Form factor	2.5 inch
Capacity	320 GB
Performance	
Spindle speed	5400 RPM
Average seek time	12.0 ms
Maximum seek time	21 ms
Track-to-track seek time	2 ms
Transfer rate (surface to buffer)	850 Mbit/s (maximum)
Transfer rate (buffer to host)	3 Gbit/s
Buffer memory	8 MB

28. (3 points) Suppose I have a disk such as the 320 GB SATA drive described in the figure above, and I have a workload consisting of 10,000 reads to sectors randomly scattered across the disk.
How long will these 10,000 requests take (total) assuming the disk services requests in FIFO order?
1 sector = 512 Bytes, 1 full rotation = 11ms, 850 Mbit/s = 106.25 MB / s

175 seconds
10,000 reads * (seek time + rotation time + transfer time)
10,000 * (12ms + 5.5ms + 0.005ms)
10,000 * 17.505ms = 175,050ms = 175 seconds
transfer time = 512 / 106,250,000 = 0.005ms

29. (3 points) Suppose I have a disk such as the 320 GB SATA drive described in figure above, and I have a workload consisting of 10,000 reads to 10,000 sequential sectors on the outer-most tracks of the disk.
How long will these 10,000 requests take (total) assuming the disk services requests in FIFO order?
1 sector = 512 Bytes, 1 full rotation = 11ms, 850 Mbit/s = 106.25 MB / s

65.5 ms
one seek + one rotation + transfer time
10,000 * 512bytes = 5,120,000 bytes
5,120,000 / 106,250,000 = 48ms
12ms + 5.5ms + 48ms = 65.5ms