Segmentation and Paging Algorithms

44-550: Operating Systems

Last Time: Paging

Advantages

- No external fragmentation
- Simple memory management algorithm
- Swapping is easy
- Transparent to programmer (system allocates memory)

Disadvantages

- Internal fragmentation on last page of process memory
- Page tables may consume more memory
- Doesn't really offer memory protection
  - Can only flag whether or not a frame is valid or invalid

Segmentation

- Variable length modules of a program corresponding to logical units (addresses)
- Represent modular structure of a program
- Examples include:
  - Main method
  - Other methods
  - Data
- We *prefer* to load segments into contiguous blocks of memory
- Very common way to achieve memory protection
- Segments allocated in various sizes depending on the process' need for address space
- Note: At this point, if you google paging vs segmentation, the results you get will be hairy; Much of the difference depends on what level we're talking "segmentation". Additionally, some Windows documentation refers to swapping as paging...

Segmentation

- The OS keeps track of segments for a process inclduding starting address and length.
- Linux avoids low level segmentation as much as possible:
  - Kernel Code
  - Kernel Data
  - Userspace Code
  - Userspace Data
- But wait... what's a segmentation fault?
  - Program tried to access an invalid or illegal memory address (like NULL) or a value larger than a valid pointer for that process
  - Not really related to the segmented memory model.
  - Different from a *Page fault*
- Operating systems can overlay segmentation on top of paging: a process has segments, and those segments directly map to one or more pages

Virtual Memory

- Memory space of a process is divided into blocks of pages or segments
  - Doesn't matter what, for our purposes
- Not all blocks are required to execute a process
- Keep entire program address space on disk
  - BUT, we usually have a lot less physical RAM than we do disk space.
- The virtual memory manager swaps out pages/segments of an executing process as it's needed
- The OS must provide a way to translate between virtual and physical addresses
- Virtual address space is MUCH larger than physical address space
  - Otherwise, a 30GB game (not unheard of) wouldn't fit in 4-8GB of ram
- An efficient way to load blocks as the program executes is needed
  - Either swap space or page file

Process Locality and Locality of Reference

- The set of pages a process operates on are called a locality
- Locality of Reference is the concept that data accesses are predictable; can be exploited to have an intelligent virtual memory manager
- Temporal Locality
  - Data recently used is likely to be reused
- Spatial Locality
  - Data element X was used, so data close to it in physical memory may be used soon
- Exploited to avoid *page faults* as much as possible
  - If required page is not in memory, must go to swap space to fetch it
  - This is a slow process, since we are accessing a slower kind of memory (HDD/SDD)

Address Translation

1. Check for valid page number
2. Check for valid physical address
   - If page is not in memory, OS must trigger an interrupt to acces disk (page fault)
3. OS must write data pages back to disk if there are modifications to physical memory (must keep the page/swap file up to date)
   - Hooray consistency!

- Managing the page table can be costly; you may only need to load part of it into memory
- Page size is important too
  - Too big: taking time to load spurious data
  - Too small: Lots of page faults

Paging with Virtual Memory

- Large processes may require multiple pages
- Page faults occur when the process references a page not in memory
  - This is a very costly operation

Paging Policies

- Fetch policy: when to swap a page into memory
- Replacement policy: If there are no empty frames, which page gets replaced?
- Placement policy: where to place the page in memory
- Number of frames to allocate to a process

Fetch Policy

- Demand paging: page isn't loaded until referenced in memory
  - Page faults occur when an instruction or an operand to an instruction is needed
- Prepaging: Load page before it is referenced by the program
  - Must be careful about which pages to load (what happens if they don't get used soon?)

Replacement Policy

- When a page fault occurs:
  - Process generating fault is suspended
  - OS locates referenced page on disk using page tables
- if there is no free frame, a page must be selected to be moved back to the HDD
- Load the referenced page into the selected frame and update page/frame tables
- Resume interrupted process
- If frame sent back to memory has been modified, need to update the page in memory

Frame Allocation

- How many frames should a process be allowed?
- Too few frames means lots of page faults
  - Causes *thrashing*: worse than deadlock because nobody makes progress, and the system can be brought down
  - Also: excessive thrashing could (possibly) reduce the life of SSDs (though not noticeably)
- Can use equal or proportional allocation (proportional to size)
- Replacement can use either local or global allocation
  - local: select only from process' frames
  - global: select from other process frames or empty ones

Page Faults and Performance

- Time to service page fault is important (lots of overhead)
  - Time interval to service page fault interrupt
  - Time to swap out replaced page to disk
  - Tiem to swap in (load) referenced page from disk
  - Delay in queueing for HDDs
  - Delay in schedculing process with referenced page
- We can investigate how much this hurts by guessing at the page fault rate ($0 \leq p \leq 1.0$):
  - If $p = 0$, no page faults
  - If $p = 1$, every reference is a fault

Effective Access Time (EAT)

Effective Access Time

If we assume that checking to see if the page we want is already loaded is instantaneous (0 time): $EAT = (1 - p)(MAT) + p(FST)$

MAT: memory access time; FST: Fault Service Time

Tells us on average how long memory accesses take

Example: $MAT = 200ns$, $FST = 8ms = 8000000ns$. Then:

$$\begin{aligned} EAT(p) &= 200(1-p) + 8000000p \\ &= 200 - 200p + 8000000p \\ &= 200 + 7999800p \end{aligned}$$

p	$EAT(p)$
0	.2us
$\frac{1}{1000}$	8.2us(40x slower)
1	8ms(40000x slower)

Paging Algorithm

- Assume a process has a stream of pages it will reference (per process)
- Assume also that each action is represented by one page (for simplicity)
  - Example: $<$ 3, 6, 1, 3, 3, 2, 7, 8, 9, 10, 11$>$
- We have 3 static policies we can use:
  - FIFO
  - Optimal replacement
  - LRU

FIFO

- As name suggests, the first page to get replaced will be the first to be swapped out
- Our example will use a 3 page "segment"
  - All this means is that we have 3 pages that can be loaded at a time
- Note: This can exhibit poor performance even if the number of frames is increased

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:														
Page 2:														
Page 3:														
Hit/Fault														

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4													
Page 2:														
Page 3:														
Hit/Fault	F													

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4												
Page 2:		7												
Page 3:														
Hit/Fault	F	F												

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4											
Page 2:		7	7											
Page 3:			3											
Hit/Fault	F	F	F											

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0										
Page 2:		7	7	7										
Page 3:			3	3										
Hit/Fault	F	F	F	F										

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0									
Page 2:		7	7	7	1									
Page 3:			3	3	3									
Hit/Fault	F	F	F	F	F									

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0								
Page 2:		7	7	7	1	1								
Page 3:			3	3	3	7								
Hit/Fault	F	F	F	F	F	F								

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3							
Page 2:		7	7	7	1	1	1							
Page 3:			3	3	3	7	7							
Hit/Fault	F	F	F	F	F	F	F							

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3						
Page 2:		7	7	7	1	1	1	8						
Page 3:			3	3	3	7	7	7						
Hit/Fault	F	F	F	F	F	F	F	F						

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3					
Page 2:		7	7	7	1	1	1	8	8					
Page 3:			3	3	3	7	7	7	5					
Hit/Fault	F	F	F	F	F	F	F	F	F					

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3	4				
Page 2:		7	7	7	1	1	1	8	8	8				
Page 3:			3	3	3	7	7	7	5	5				
Hit/Fault	F	F	F	F	F	F	F	F	F	F				

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3	4	4			
Page 2:		7	7	7	1	1	1	8	8	8	8			
Page 3:			3	3	3	7	7	7	5	5	**5**			
Hit/Fault	F	F	F	F	F	F	F	F	F	F	H			

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3	4	4	4		
Page 2:		7	7	7	1	1	1	8	8	8	8	3		
Page 3:			3	3	3	7	7	7	5	5	**5**	5		
Hit/Fault	F	F	F	F	F	F	F	F	F	F	H	F		

FIFO Example

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3	4	4	4	**4**	
Page 2:		7	7	7	1	1	1	8	8	8	8	3	3	
Page 3:			3	3	3	7	7	7	5	5	5	5	5	
Hit/Fault	F	F	F	F	F	F	F	F	F	F	H	F	H	

Needed Page:	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Page 1:	4	4	4	0	0	0	3	3	3	4	4	4	**4**	4
Page 2:		7	7	7	1	1	1	8	8	8	8	3	3	3
Page 3:			3	3	3	7	7	7	5	5	5	5	5	7
Hit/Fault	F	F	F	F	F	F	F	F	F	F	H	F	H	F

$$p = \frac{12}{14} = .85714$$

$$EAT(.857) = 6.85ms$$

‘ Ouch

Other Paging Algorithms

- Optimal:
  - Replace page in memory that will not be used for the longest period
  - Requires the entire page reference stream in advance (what's with all the psychic algorithms?)
- Least Recently Used:
  - Replace the page in memory that has not been used for the longest period
  - Exploits Locality of Reference (temporal)
- In class examples: next time