9 – Virtual Memory

9 – Virtual Memory

- Benefits of the virtual memory
- Demand paging
- Page replacement algorithms
- Working-set model

Background

- In many cases, an entire program is not needed.
  - Some code, e.g., error handling, is rarely used.
  - Arrays, lists, tables may allocate some memory but not be fully used.
  - etc.
- So, something should be done to allow programs to run although they are not entirely loaded to memory.
  - That's something is the virtual memory.

(cont'd.)

- Separate logical address space from physical address space
  - We have seen this already.
  - But, the point is that we do not need to load the whole program, so logical address space can be much larger than physical address space.
- A common technique to implement is *demand paging*.

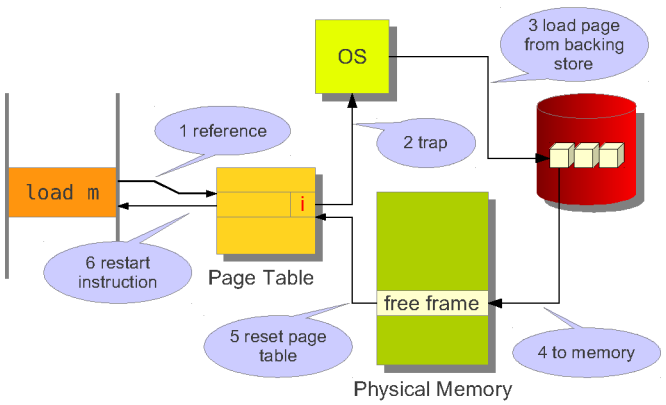
Demand Paging

- Similar to swapping but only page.
  - Remember ? Swapping switches the whole process(es).
  - Imply paging system.
- Bring page into memory only when it is needed – *lazy swapper*.
  - Less memory needed.
  - Less I/O because less pages read.
  - Less loading latency because pages are smaller.
  - Increase the degree of multiprogramming.
- Need the *valid-invalid bit* to distinguish pages in memory and those are on the disk.

Page Faults

- Occur when a process access pages that was not in the memory.
  - The paging hardware generates a page-fault trap to the operating system.
- So, the operating system must handle the page fault. How ?

(cont'd.)



Performance of Demand Paging

Let α be probability of the page fault $(0 \leq \alpha \leq 1)$,
m be memory access time,
p be page fault time

$$\text{EAT} = (1 - \alpha)m + \alpha p$$

where p = page fault overhead
+ swap-out time
+ swap-in time
+ restart overhead

(cont'd.)

Example
m = 200 ns., p = 8 ms.
EAT = $(1 - \alpha)200 + 8{,}000{,}000\alpha$ ns.
= $200 + 7{,}999{,}800\alpha$ ns.

If page fault occurs once every 1000 accesses,
$\alpha = 0.001$

EAT = $200 + 7999.8 = 8199.8$ ns.

So, demand paging is about 41 times slower.

Basic Page Replacement

- Find the location of the desired page on disk.
- Find a free frame.
  - If there is a free frame, use it.
  - If there is no free frame, use a page replacement algorithm to select a frame to be replaced.
- Bring the desired page into the free frame, update the page and frame tables.
- Restart the process.

Page Replacement Algorithms

- FIFO
- Optimal
- Least Recently Used
- Second Chance
- Counter

FIFO Algorithm

- Simplest one
- Example: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - 3 frames

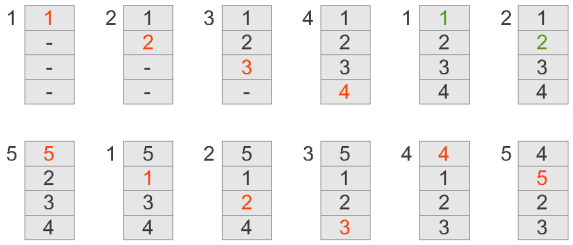
1	2	3	4	1	2
1	1	1	4	4	4
-	2	2	2	1	1
-	-	3	3	3	2

5	1	2	3	4	5
5	5	5	5	5	5
1	1	1	3	3	3
2	2	2	2	4	4

  - 9 page faults
- More frames should be less page faults.

(cont'd.)

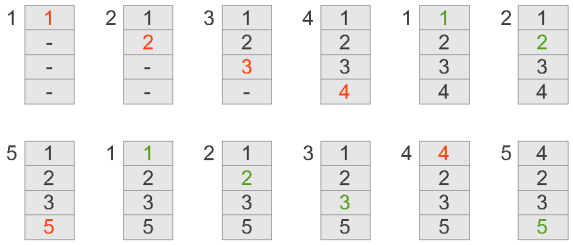
- Let's see
  - 4 frames



  - 10 page faults
- *Belady's anomaly*: more frame, more page faults

The Optimal Algorithm

- Ideal, the best one
- Replace page that **will** not be used for a long period of time, e.g.,
  - 4 frames:



(cont'd.)

- Impossible to implement, but this is the baseline to measure other algorithms.

Least-Recently-Used (LRU) Algorithm

- Replace the least-recently-used frame
  - Imply a timestamp for every page/frame
  - Every time a page is referenced, update the timestamp.
  - The algorithm looks at the timestamp, the oldest one will be replaced

(cont'd.)

1			2			3			4			1			2		
1	1		1	1		1	1		1	1		1	5		1	5	
-	-		2	2		2	2		2	2		2	2		2	6	
-	-		-	-		3	3		3	3		3	3		3	3	
-	-		-	-		-	-		4	4		4	4		4	4	

5			1			2			3			4			5		
1	5		1	8		1	8		1	8		1	8		5	12	
2	6		2	6		2	9		2	9		2	9		2	9	
5	7		5	7		5	7		5	7		4	11		4	11	
4	4		4	4		4	4		3	10		3	10		3	10	

  - 8 page faults
- It can also be implemented using stack
  - Referenced, move to the top.
  - Page fault, replace the bottom.

(cont'd.)

1		2		3		4		1		2	
1		2		3		4		1		2	
-		1		2		3		4		1	
-		-		1		2		3		4	
-		-		-		1		2		3	

5		1		2		3		4		5	
5		1		2		3		4		5	
2		5		1		2		3		4	
1		2		5		1		2		3	
4		4		4		5		1		2	

  - With double-linked list, update is bound at 6 pointers.

LRU Approximation Algorithm

- Based on LRU
- Introduce reference bits
  - Associated to each page.
  - Initially, reference bit = 0
  - Referenced, set bit = 1
  - Page fault, if exists, replace one that bit = 0
    - Any arbitrary order

(cont'd.)

- Additional-Reference-Bit Algorithm
  - A set of bits, e.g., 1 byte
  - The operating system shifts bits every regular interval.
    - Shift reference bit to high-order bit
    - SHR the others.
  - So, it keeps page reference history.
    - `11000100` has been used more recently than has one with `01110111`.
    - The lowest number is the LRU.
      - It may not be unique, so use a FIFO selection among them.

(cont'd.)

4	1	00010000
	2	00100000
	3	01000000
	4	10000000

1	1	10001000
	2	00010000
	3	00100000
	4	01000000

2	1	01000100
	2	10001000
	3	00010000
	4	00100000

5	1	00100010
	2	01000100
	5	10000000
	4	00010000

1	1	10010001
	2	00100010
	5	01000000
	4	00001000

2	1	01001000
	2	10010001
	5	00100000
	4	00000100

3	1	00100100
	2	01001000
	5	00010000
	3	10000000

4	1	00010010
	2	00100100
	4	10000000
	3	01000000

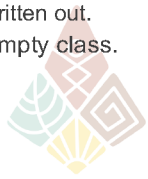
5	5	10000000
	2	00010010
	4	01000000
	3	00100000

Second-Chance Algorithm

- A kind of FIFO algorithm + reference bit
- When a page has been selected to be replaced, the operating system checks the reference bit.
  - If the reference bit = 0, replace it.
  - If the reference bit = 1,
    - Set reference bit to 0.
    - Leave the page in memory.
    - Replace next page, subject to the same rule.
- With circular queue, the algorithm would finally replace some page.

(cont'd.)

- Enhanced Second-Chance Algorithm
  - Reference bit and modify bit
  - Four classes
    1. (0, 0) neither used nor modified, the best to replace.
    2. (0, 1) the page need to be written out before replacement, so it should not be replaced.
    3. (1, 0) may be used again soon.
    4. (1, 1) may be used again soon, have to written out.
  - Replace the first page in the lowest non-empty class.
  - Macintosh uses this technique.



The Counter Algorithms

- Every page has a counter.
- Increase the counter every time a page is referenced.
- Least-Frequently-Used (LFU) algorithm replaces the page with smallest counter.
  - It does not work well if the page is used heavily initially, then is never used again.
    - SHR at regular intervals.
- Most-Frequently-Used (MFU) algorithm replaces the page with largest counter.
  - The smallest counter has just been brought in.

Allocation of Frames

- Problem: there are m frames, how do we distribute those frame among n active processes ?
- Each process needs the minimum number of pages.
- Two main schemes
  - Fixed allocation
  - Priority allocation

Fixed Allocation

- Equal allocation
  - Equal frames for each process
  - Does not make much sense since different processes require different numbers of pages
- Proportional allocation
  - The number of available frames is m, each process p_i allocate approximately

$$a_i = \frac{s_i}{\sum s_i} m$$

where s_i is the size of the virtual memory for process p_i.

Priority Allocation

- Similar to proportional allocation, but by priority instead of the size of VM.
- If process p_i generates a page fault
  - Replace one of its frames
  - Replace a frame from a process with lower priority

Global vs. Local Allocation

- Global replacement
  - Process selects a replacement frame from the set of all frames.
  - One process can take a frame from another.
- Local replacement
  - Each process selects from only its own set of allocated frames.

Thrashing

- For any process that does not have enough frames, it is technically possible to reduce the number of allocated frame to the minimum.
  - But, a large number of frame may be actively used.
  - The page-fault rate will be very high.
    - Low CPU utilization
    - The operating system detects the low utilization, it adds more process to get higher degree of multiprogramming.
- This high activity of paging is called *thrashing*.
  - This kind of processes spends more time paging than executing.

Locality Model

- To prevent trashing, a process should get frames it needs. There are many techniques, e.g., locality model.
- Locality model states that, as a process executes, it moves from locality to locality
  - Locality is a set of pages that are actively used together.
  - If we can allocate enough frames for current locality, it would not page fault again until it changes localities.
    - And if it cannot allocate enough frames for the locality, it will trash.

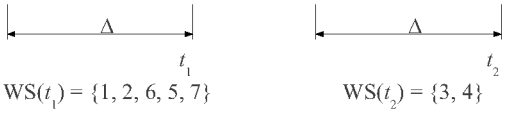
Working-Set Model

- Based on the assumption of locality
- A working set is an approximation of the program's locality.
- A parameter Δ defines the *working-set window*.
  - A fixed number of page references.
- The set of pages in the most recent Δ page references is the *working set*.
  - If a page is in active use, it will be in the working set.
  - If it is no longer being used, it will be dropped from the working set Δ time units after its last reference.

(cont'd.)

Example $\Delta = 10$

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 ...



- If Δ is too small, it will not cover the entire locality.
- If Δ is too large, it may overlap several localities.
- $D = \sum WSS_i$ = total demand frames
- If $D > m$, then thrashing
  - Suspend one of the processes to make frames available

In the Real World ...

- Windows can expand the page file.
  - Windows 3.x – `386SPART.PAR`, `WIN386.SWP`
  - Windows NT/2K/XP – `pagefile.sys`
  - Up to 3x physical memory
  - Expanding is bad since it leads to more external fragmentations, and cause performance drop.
  - So, do not let it resize the page file.
- Linux uses the whole partition(s).
  - `mkswap`, `swapon`, `swapoff`
  - Put swap partition at very beginning of the disk may increase performance.
  - There is a hack to use video memory as a storage.