CHAPTER 9: VIRTUAL MEMORY

- Background
- Demand Paging
- Performance of Demand Paging
- Page Replacement
- Page-Replacement Algorithms
- Allocation of Frames
- Thrashing
- Other Considerations
- Demand Segmentation

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Only *part* of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Need to allow pages to be *swapped* in and out.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Valid–Invalid bit

- With each page table entry a valid–invalid bit is associated (1 $\Rightarrow$ in-memory, 0 $\Rightarrow$ not-in-memory)

- Initially valid–invalid bit is set to 0 on all entries.

- Example of a page table snapshot.

frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 $\Rightarrow$ page fault.

Page Fault

1. If there is ever a reference to a page, first reference will trap to OS ⇒ *page fault.*

2. OS looks at another table to decide:

   a) Invalid reference ⇒ abort.

   b) Just not in memory.

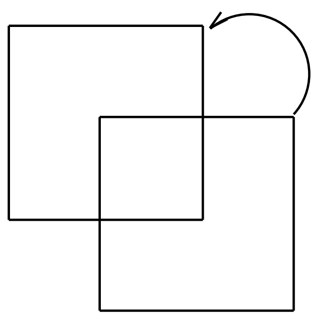
3. Get empty frame.

4. Swap page into frame.

5. Reset tables, validation bit = 1.

6. Restart instruction:

   - block move



   - auto increment/decrement location

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.

  $\Rightarrow$ algorithm

  $\Rightarrow$ performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times.

Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$

  if $p = 0$, no page faults

  if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

  EAT = $(1 - p) \times$ memory access

  $+ p$ (page fault overhead

  + [swap page out]

  + swap page in

  + restart overhead)

- Example:
  - memory access time = 1 microsecond
  - 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
  - Swap Page Time = 10 msec = 10,000 msec
  - EAT = $(1 - p) \times 1 + p$ (15000)

    $= 1 + 15000P$ (in msec)

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.

- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Page-Replacement Algorithms

- Want lowest *page-fault rate.*

- Evaluate algorithm by running it on a particular string of memory references (*reference string*) and computing the number of page faults on that string.

- In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frames (3 pages can be in memory at a time per process)

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

9 page faults

- 4 frames

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

10 page faults

FIFO Replacement – Belady’s Anomaly

more frames $\not\Rightarrow$ less page faults

Optimal Algorithm

- Replace the page that will not be used for the longest period of time.

- 4 frames example

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

1	4
2	
3	
4	5

6 page faults

- How do you know this?

- Used for measuring how well your algorithm performs.

Least Recently Used (LRU) Algorithm

1		5
2		
3	5	4
4	3	

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine which are to change
- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0.
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit.
  - Clock replacement.
  - If page to be replaced (in clock order) has reference bit = 1, then:

    a) set reference bit 0.

    b) leave page in memory.

    c) replace next page (in clock order), subject to same rules.

- Counting Algorithms – keep a counter of the number of references that have been made to each page.

  - LFU Algorithm: replaces page with smallest count.

  - MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

- Page-Buffering Algorithm – desired page is read into a free frame from the *pool* before the victim is written out.

Allocation of Frames

- Each process needs minimum number of pages.

  Example: IBM 370 – 6 pages to handle
  SS MOVE instruction:

  a) Instruction is 6 bytes, might span 2 pages.

  b) 2 pages to handle **from.**

  c) 2 pages to handle **to.**

- Two major allocation schemes:

  - fixed allocation

  - priority allocation

- Fixed allocation
  - Equal allocation

    If 100 frames and 5 processes, give each 20 pages.

  - Proportional allocation

    Allocate according to the size of process.

    - s_i = size of process p_i
    - $S = \Sigma s_i$
    - m = total number of frames
    - a_i = allocation for $p_i = \dfrac{s_i}{S} \times m$

Example : $m = 64$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

- Priority allocation

  - Use a proportional allocation scheme using priorities rather than size.

  - If process P_i generates a page fault,

    - select for replacement one of its frames.

    - select for replacement a frame from a process with lower priority number.

Global versus 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

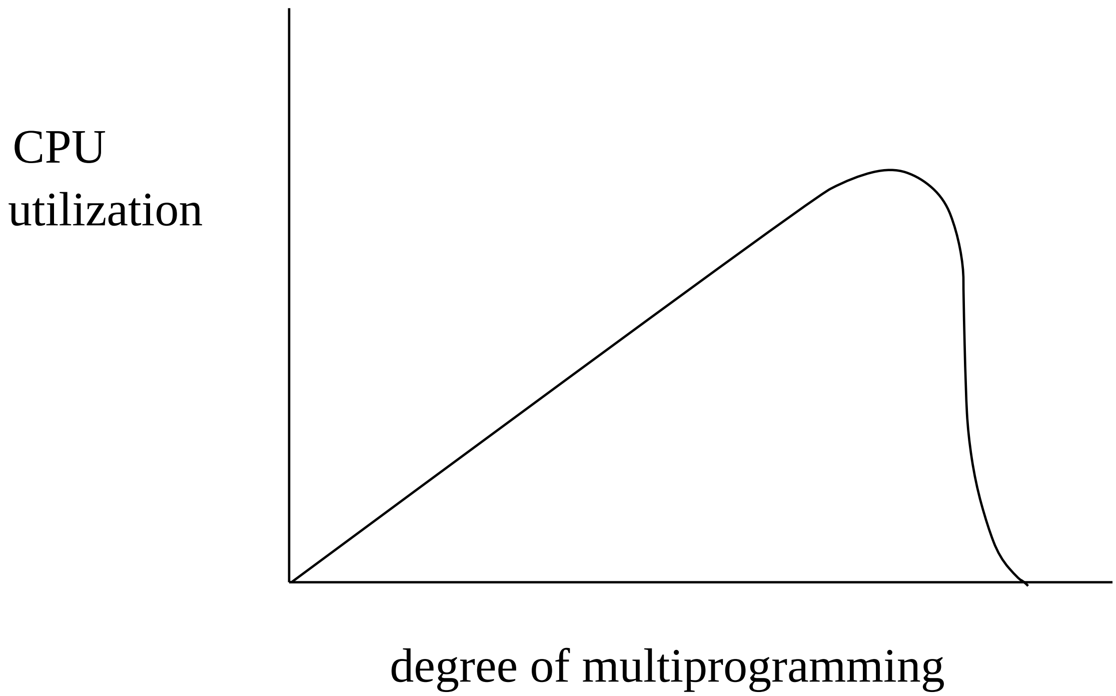
- If a process does not have ‘‘enough’’ pages, the page-fault rate is very high:

  $\Rightarrow$ low CPU utilization.

  $\Rightarrow$ operating system thinks that it needs to increase the degree of multiprogramming.

  $\Rightarrow$ another process added to the system.

- Thrashing $\equiv$ a process is busy swapping pages in and out.



- Why does paging work?

  Locality model

  - Process migrates from one locality to another.
  - Localities may overlap.

- Why does thrashing occur?

  Σ size of locality > total memory size

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references

  Example: 10,000 instruction

- WSS_i – working set of process P_i =

  total number of pages referenced in the most recent Δ (varies in time)

  If Δ too small will not encompass entire locality.

  If Δ too large will encompass several localities.

  If $\Delta = \infty \Rightarrow$ will encompass entire program.

- $D = \Sigma\, WSS_i \equiv$ total demand frames
- If $D > m \Rightarrow$ thrashing.
- Policy if $D > m$, then suspend one of the processes.

How do you keep track of the working set?

- Approximate with:

  interval timer + a reference bit

- Example:

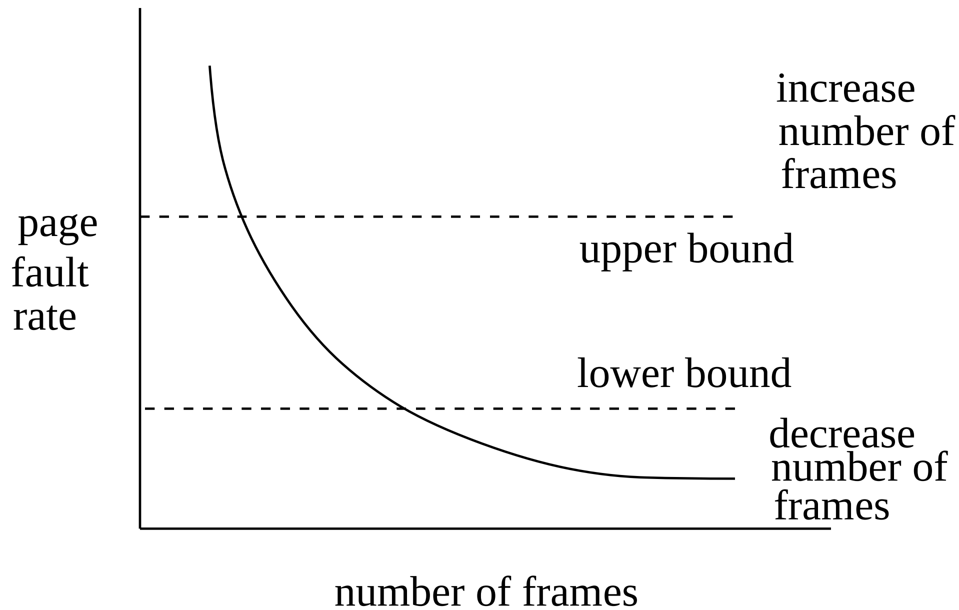
  $\Delta = 10{,}000$

  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.

  Not completely accurate (why?)

  Improve = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme



- Establish ‘‘acceptable’’ page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.

Other Considerations

1. Prepaging
2. Page size selection
   - fragmentation
   - table size
   - I/O overhead
   - locality
3. Program structure
   - Array A[1024,1024] **of** integer
   - Each row is stored in one page
   - One frame
   - Program 1

```
for j := 1 to 1024 do
    for i := 1 to 1024 do
        A[i, j] := 0;
```

     1024×1024 page faults

   - Program 2

```
for i := 1 to 1024 do
    for j := 1 to 1024 do
        A[i, j] := 0;
```

     1024 page faults
4. I/O interlock and addressing

Demand Segmentation – used when insufficient hardware to implement demand paging.

- OS/2 allocates memory in segments, which it keeps track of through *segment descriptors*.
- Segment descriptor contains a valid bit to indicate whether the segment is currently in memory.
  - If segment is in main memory, access continues,
  - If not in memory, segment fault.