Virtual memory - Swapping

Johan Montelius

KTH

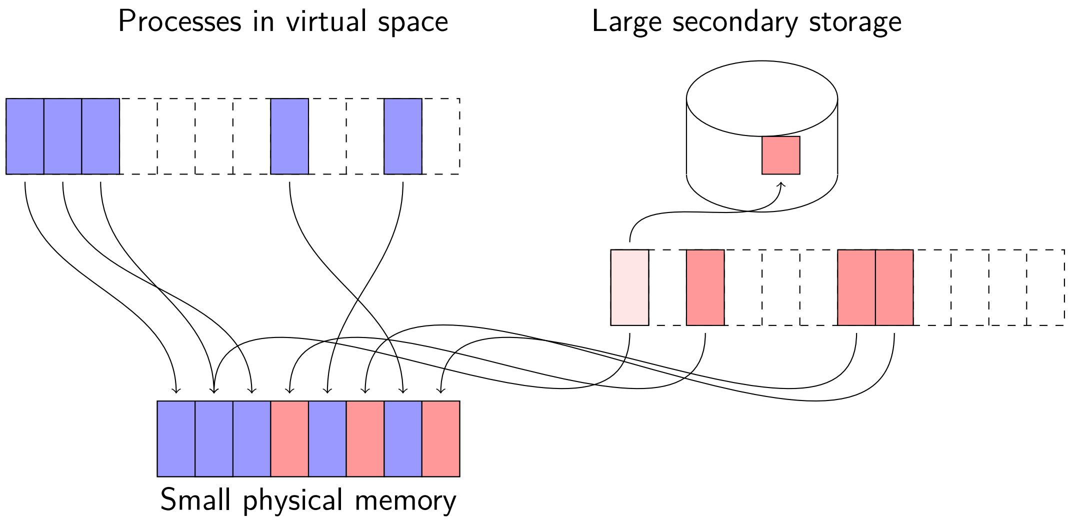
2020

Virtual memory

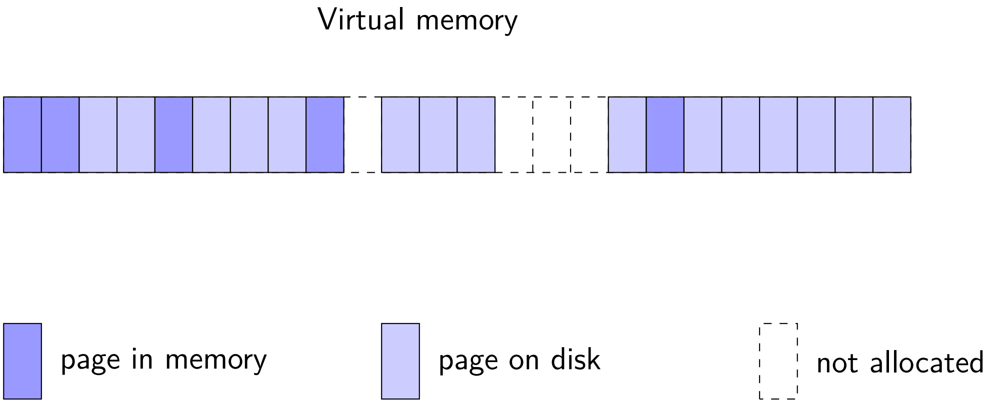
- **1:** Allowing two or more processes to use main memory, given them an illusion of private memory.
- **2:** Provide the illusion of a much larger address space than provided by the main memory.

Pages can be temporarily stored in secondary memory i.e. on disk.

Processes


Processes in virtual space
Large secondary storage
Small physical memory

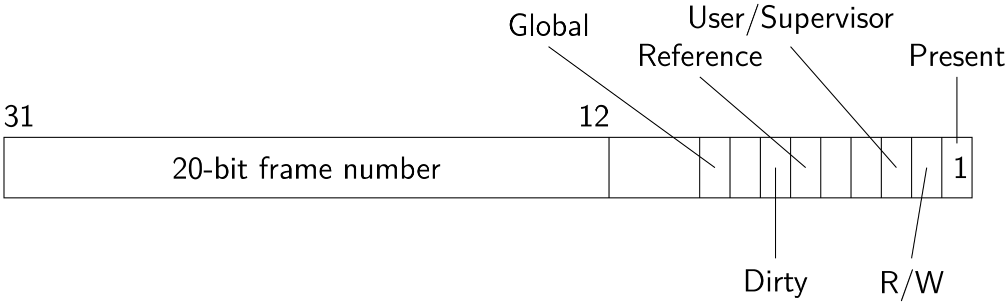
Large virtual memory



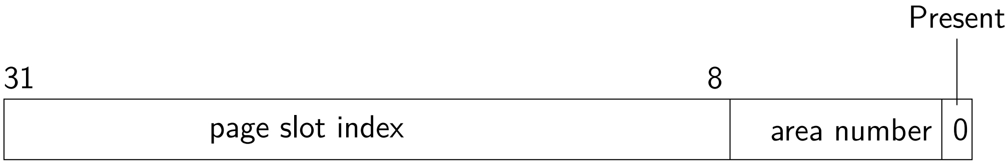
The problem of Swapping

- Memory management must detect that a page is currently not in memory.
- If it is not in memory, how do we find it?
- If the memory is full, which page do we throw out?
- When we throw out a page, do we have to copy it to disk?
- Who should do all this, hardware or operating system?

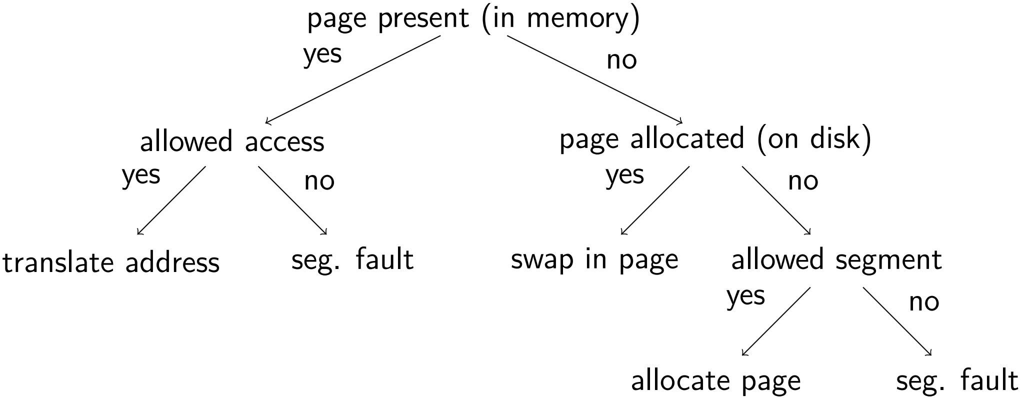
The page table entry (PTE)



The page table entry (PTE)

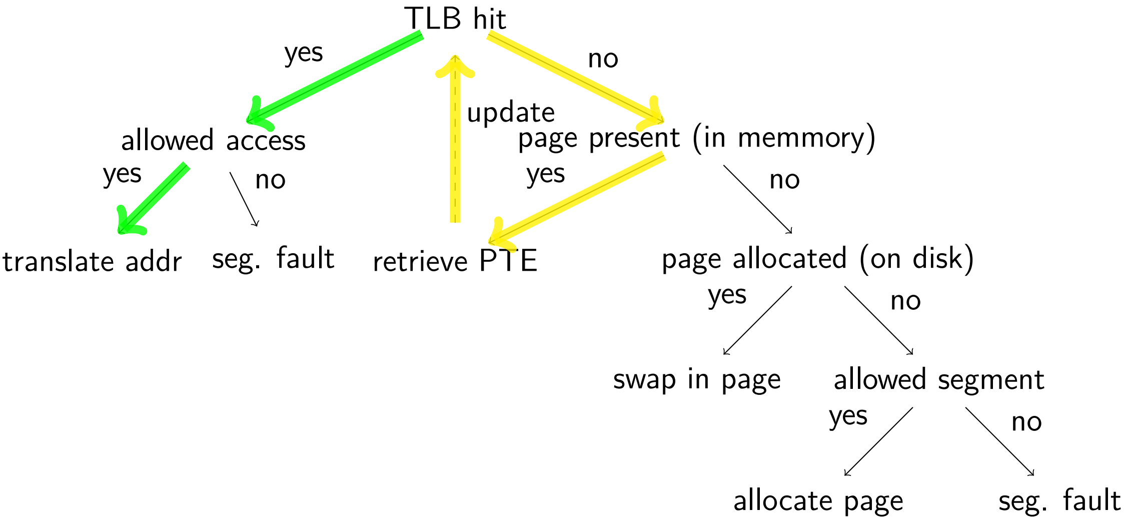


Page faults



page present (in memory)
yes
no
allowed access
yes
no
translate address
seg. fault
page allocated (on disk)
yes
no
swap in page
allowed segment
yes
no
allocate page
seg. fault

Remember the TLB



The operating system

The operating system keeps track of all processes and maintains:

a memory structure

- Which segments are allowed.
- Read/write access.
- User/Supervisor mode.
- Copy on write.

a page table

- Which pages are allocated?
- Physical frames of pages or
- .. location on secondary storage.
- Access rights.
- Modified, accessed, cacheable...

The cost of page faults

The cost of memory access:

- TLB hit, address found in cache: $\sim 1ns$
- TLB hit, address in memory: $\sim 10ns$
- TLB miss, page in memory: up to $100ns$
- TLB miss, page on disk: up to $10ms$

Retrieving from disk is a factor 100.000 times more expensive than finding things in memory.

What can we do while we're waiting?

Replacement policy

The problem with caching - which item do we throw out when the cache is filled?

Why try to be smart - pick a page by random.

Are pages referenced randomly?

Locality of references

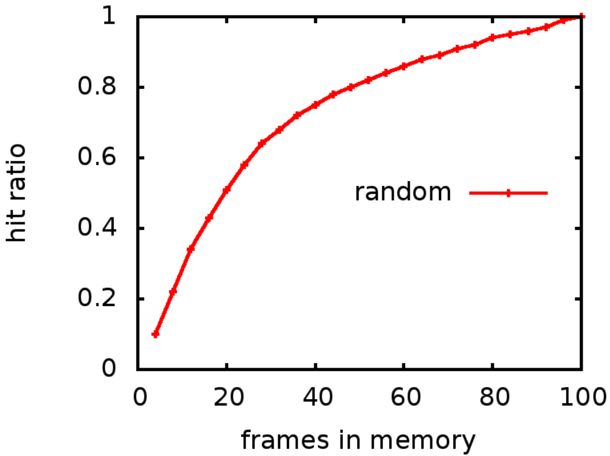
Memory references are not random in space nor time.

- **Temporal locality**: an address that has been referenced is likely to be referenced soon again.
- **Spatial locality**: an address that is close to something that has been referenced is likely to be referenced.

In these benchmarks we have simulated locality by assuming that 20% of the pages are access 80% of the time.

The random policy

When the memory is full select a frame by random and move it to disk.

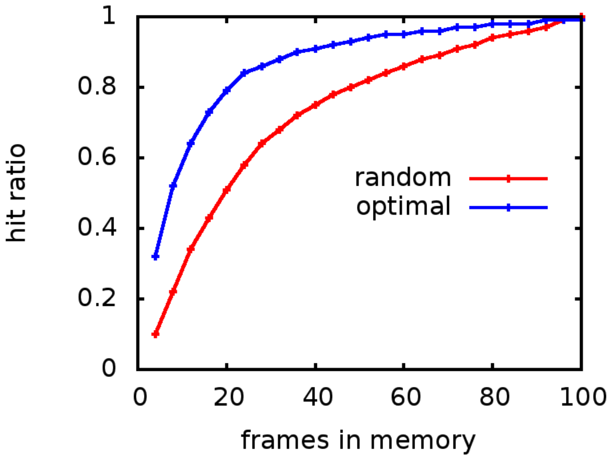


We can do better!

- When you need to throw out a page, select the one that will be used *the furthest in the future*.
- page references: 0,1,2,3,0,2,3,1,2,0,3,0

access	hit/miss	evict	memory
0	miss	-	0
1	miss	-	0,1
2	miss	-	0,1,2
3	miss	1	0,3,2
0	hit		0,3,2
2	hit		0,3,2
3	hit		0,3,2
1	miss	3	0,1,2
2	hit		0,1,2
0	hit		0,1,2
3	miss	1	0,3 2
0	hit		0,1 2

Optimal replacement policy



Case closed .... ehhh?

Optimal replacement policy

Important to know the best possible solution (even if it's not obtainable).

We might not have access to the future - but the past might give us a good approximation.

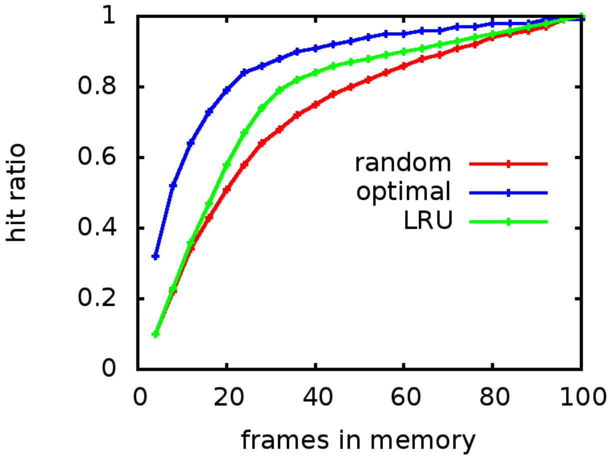
Least Recently Used (LRU)

- A page that has not been referenced for long is not likely to be referenced in the near future.
- page references: 0,1,2,3,0,2,3,1,2,0,3,0

access	hit/miss	evict	queue
0	miss	-	0
1	miss	-	0,1
2	miss	-	0,1,2
3	miss	0	1,2,3
0	miss	1	2,3,0
2	hit		3,0,2
3	hit		0,2,3
1	miss	0	2,3,1
2	hit		3,1,2
0	miss	3	1,2,0
3	miss	1	2,0,3
0	hit		2,0,3

Result: two more misses compared to the optimal.

Least Recently Used



Implement Least Recently Used

Keep track of a queue of pages (as many as we have frames).

In each page reference, move page to the end of the list.

When evicting a page, select the first page in the list.

Is this expensive?

- Manchester University, 1962
- 48-bit word, 16 K word memory, 96 K word “drum”
- 24-bit address space
- paged virtual memory
- 512 word pages
- approximated Least Recently Used replacement policy

a much cheaper solution - FIFO

The problem with LRU is that we need to update the lists in each page reference.

It is much cheaper if we only update the list when we have a page fault.

Idéa: It's better to keep a page that was recently brought in compared to one that has been around for a while.

FIFO - first-in, first-out

- Keep allocated pages in a queue - add in one end, reclaim in the other.
- page references: 0,1,2,3,0,2,3,1,2,0,3,0

access	hit/miss	evict	fifo
0	miss	-	0
1	miss	-	0,1
2	miss	-	0,1,2
3	miss	0	1,2,3
0	miss	1	2,3,0
2	hit		2,3,0
3	hit		2,3,0
1	miss	2	3,0,1
2	miss	3	0,1,2
0	hit		0,1,2
3	miss	0	1,2,3
0	miss	1	2,3,0

Result: only 3 hits :-(

Let's try again

- Let's try with more pages 0-4
- page references:
  0,1,2,3,0,1,4,0,1,2,3,4

access	hit/miss	evict	fifo
0	miss		0
1	miss		0,1
2	miss		0,1,2
3	miss	0	1,2,3
0	miss	1	2,3,0
1	miss	2	3,0,1
4	miss	3	0,1,4
0	hit		0,1,4
1	hit		0,1,4
2	miss	0	1,4,2
3	miss	1	4,2,3
4	hit		4,2,3

3 hits out of 12 page references - hmmm

Belady's anomaly

- Let's try with more frames, four instead of three!
- page references: 0,1,2,3,0,1,4,0,1,2,3,4

access	hit/miss	evict	fifo
0	miss		0
1	miss		0,1
2	miss		0,1,2
3	miss		0,1,2,3
0	hit		0,1,2,3
1	hit		0,1,2,3
4	miss	0	1,2,3,4
0	miss	1	2,3,4,0
1	miss	2	3,4,0,1
2	miss	3	4,0,1,2
3	miss	4	0,1,2,3
4	miss	0	1,2,3,4

WTF!

approximating LRU

Assume a *reference bit* in the page table entry, initially set to zero.

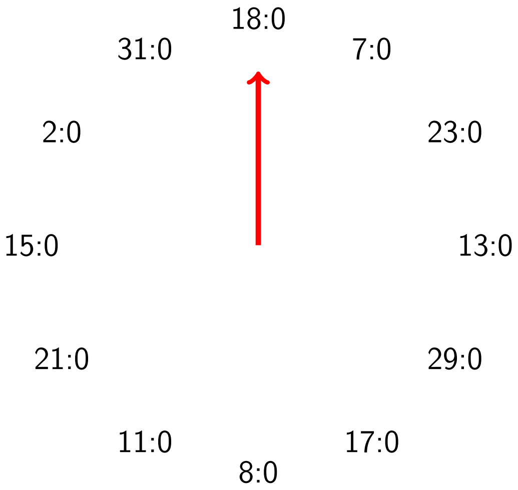
When the page is referenced, the bit is set to one - by the hardware.

When selecting a page for eviction - select a page with the reference bit set to zero.

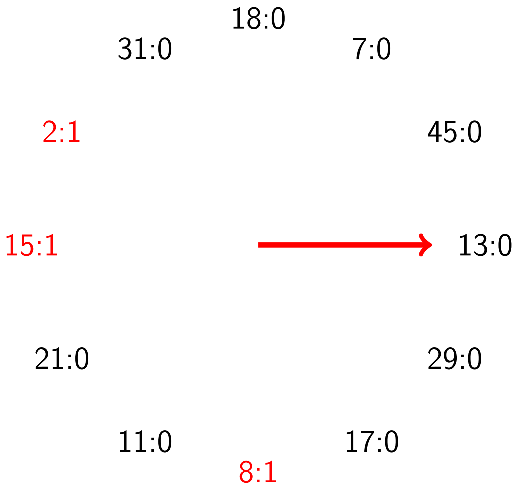
A page with a reference bit set to one - **is given a second chance.**

When should a reference bit be cleared?

the clock algorithm

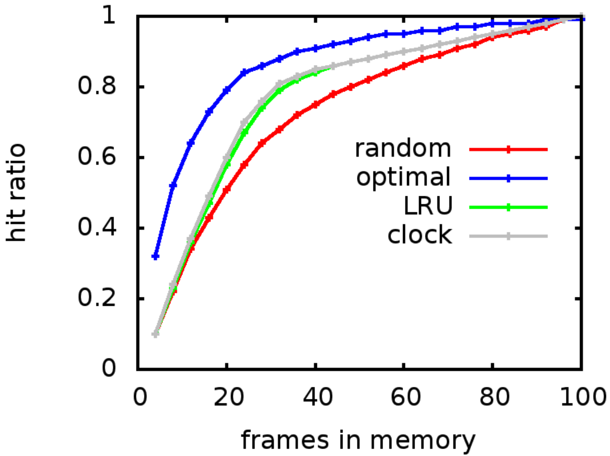


the clock algorithm



access page 7
access page 15
access page 18
access page 8
access page 2
access page 45
move forward, reset reference bit
remove page
allocate new page
move forward

the clock algorithm



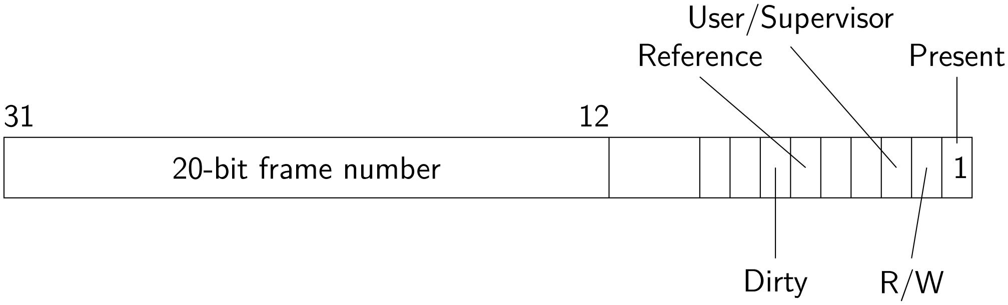
dirty pages

You have two frames to choose from, holding:

- a page that has been recently used but not modified and,
- a page that has not been used for a long time but has been modified.

Which one should we reclaim of we need a free frame?

The page table entry (PTE)



in Linux

Implementation of Page Frame Reclaiming Algorithm in Linux:

- Global i.e. all processes share all frames.
- Two sets: the *active list* and the *inactive list*.
- Each set implements an approximation of LRU similar to the clock algorithm.
- Inactive pages are moved from the active to the inactive set and vice verse.
- A kernel thread tries to maintain a set of free frames i.e. moving pages from the inactive set to disk before it is needed.
- Operations are batched to improve disk locality and reduce locking.

The problem of Swapping

- Memory management must detect that a page is currently not in memory.
- If it is not in memory, how do we find it?
- If the memory is full, which page to we throw out?
- When we throw out a page, do we have to copy it to disk?
- Who should do all this, hardware or operating system?