CS202 (003): Operating Systems
Virtual Memory IV

Instructor: Jocelyn Chen

Most of the materials covered in this slide come from the lecture notes of Mike Walfish's CS202

Understanding “page-not-present in memory”

How to decide which entry to throw away if we get a cache miss?

Replacement policy

FIFO	MIN (optimal)	LRU
throw out the oldest	throw away the entry that won't be used for the longest time	throw out the least recently used

Replacement policy

FIFO	MIN (optimal)	LRU
throw out the oldest	throw away the entry that won't be used for the longest time	throw out the least recently used

How do we evaluate these algorithms?

Input: Reference string (sequence of page accesses)
Cache size (i.e. physical memory)

Output: # of cache evictions (i.e. number of swaps)

Replacement policy

FIFO

throw out the oldest

A	B	C	A	B	D	A	D	B	C	B

A	A	A	A	A	D	D	D	D	C	C
-	B	B	B	B	B	A	A	A	A	A
-	-	C	C	C	C	C	C	B	B	B

Number of Hits: 4

Page Faults: 7

Hit Rate: 36.36%

MIN (optimal)

throw away the entry that won't be used for the longest time

A	B	C	A	B	D	A	D	B	C	B

A	A	A	A	A	A	A	A	A	A	A
-	B	B	B	B	B	B	B	B	B	B
-	-	C	C	C	D	D	D	D	C	C

Number of Hits: 6

Page Faults: 5

Hit Rate: 54.55%

LRU

throw out the least recently used

A	B	C	A	B	D	A	D	B	C	B

A	A	A	B	C	A	B	B	A	D	D
-	B	B	C	A	B	D	A	D	B	C
-	-	C	A	B	D	A	D	B	C	B

Number of Hits: 6

Page Faults: 5

Hit Rate: 54.55%

Replacement policy

FIFO

throw out the oldest

A	B	C	D	A	B	C	D	A	B	C	D
A	A	A	D	D	D	C	C	C	B	B	B
-	B	B	B	A	A	A	D	D	D	C	C
-	-	C	C	C	B	B	B	A	A	A	D

Number of Hits: 0

Page Faults: 12

Hit Rate: 0.0%

MIN (optimal)

throw away the entry that won't be used for the longest time

A	B	C	D	A	B	C	D	A	B	C	D
A	A	A	A	A	A	A	A	A	B	B	B
-	B	B	B	B	B	C	C	C	C	C	C
-	-	C	D	D	D	D	D	D	D	D	D

Number of Hits: 6

Page Faults: 6

Hit Rate: 50.0%

LRU

throw out the least recently used

A	B	C	D	A	B	C	D	A	B	C	D
A	A	A	B	C	D	A	B	C	D	A	B
-	B	B	C	D	A	B	C	D	A	B	C
-	-	C	D	A	B	C	D	A	B	C	D

Number of Hits: 0

Page Faults: 12

Hit Rate: 0.0%

Replacement policy (adding new memory)

FIFO

throw out the oldest

A	B	C	D	A	B	E	A	B	C	D	E
A	A	A	D	D	D	E	E	E	E	E	E
-	B	B	B	A	A	A	A	A	C	C	C
-	-	C	C	C	B	B	B	B	B	D	D

Number of Hits: 3

Page Faults: 9

Hit Rate: 25.0%

MIN (optimal)

throw away the entry that won't be used for the longest time

A	B	C	D	A	B	E	A	B	C	D	E
A	A	A	A	A	A	A	A	A	A	A	A
-	B	B	B	B	B	B	B	B	C	D	D
-	-	C	D	D	D	E	E	E	E	E	E

Number of Hits: 5

Page Faults: 7

Hit Rate: 41.67%

LRU

throw out the least recently used

A	B	C	D	A	B	E	A	B	C	D	E
A	A	A	B	C	D	A	B	E	A	B	C
-	B	B	C	D	A	B	E	A	B	C	D
-	-	C	D	A	B	E	A	B	C	D	E

Number of Hits: 2

Page Faults: 10

Hit Rate: 16.67%

Replacement policy

FIFO

throw out the oldest

MIN (optimal)

throw away the entry that won't be used for the longest time

LRU

throw out the least recently used

Pretty decent!

It approximates OPT when: principle of temporal locality holds strongly

Implementing LRU

In OS, it doubles the memory traffic
(since after every reference, have to move some structure to the head of some list)

In hardware, it's **a lot of work** to timestamp each reference and keep the list ordered

Implementing LRU in OS/hardware is a lot of pain!

Approximating LRU

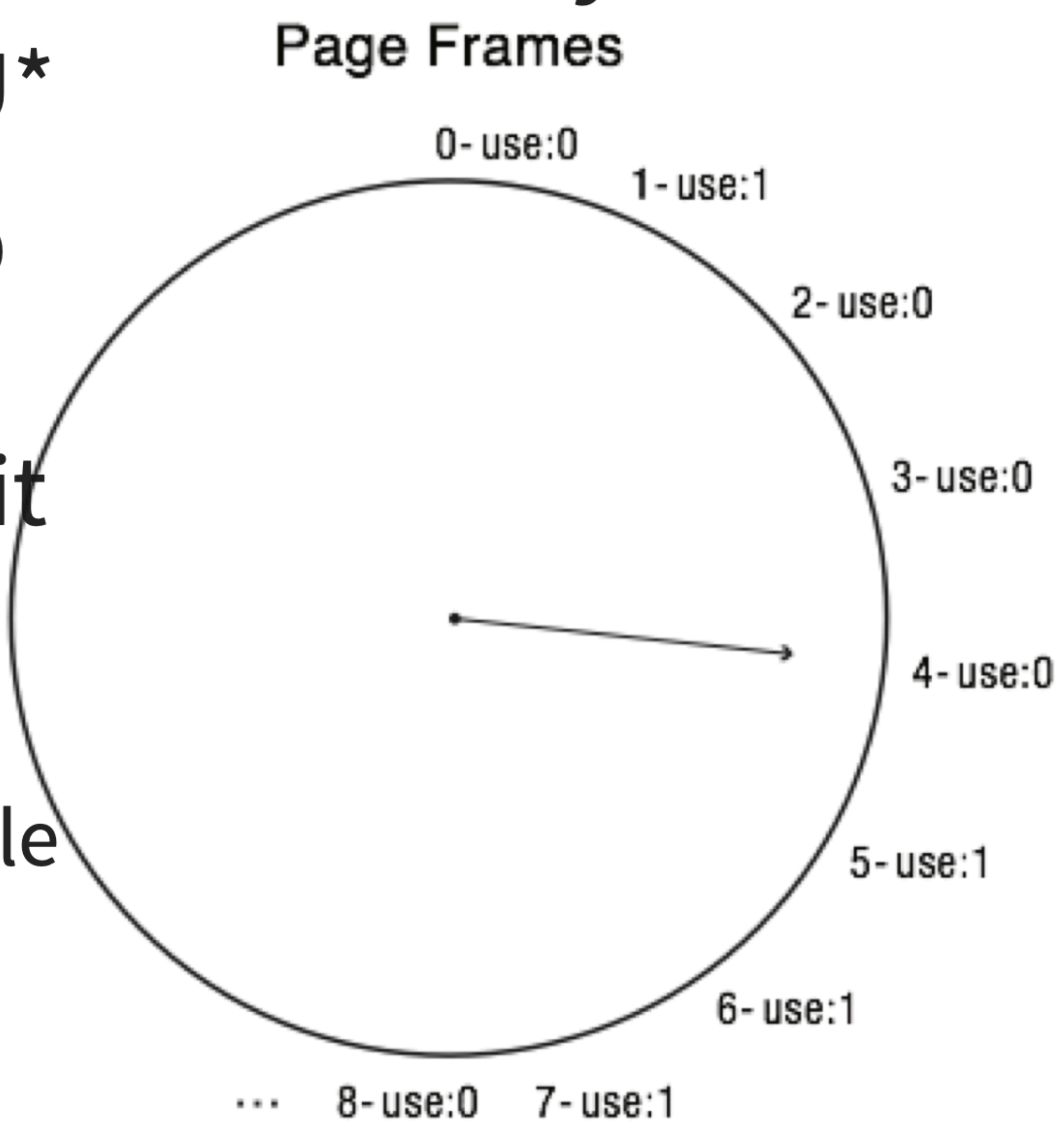
Clock Algorithm: Not Recently Used

"Second-chance" algorithm

Approximating LRU*

Periodically, sweep through all pages

- Used? Clear use bit
- Unused? reclaim
  - update core map
  - invalidate page table
  - write back if dirty
  - TLB shootdown
  - add to free list



*(*yes, LRU was already an approximation…)*

https://www.cs.cornell.edu/courses/cs4410/2018sp/schedule/slides/08-vm.pdf

Generalizing CLOCK: Nth Chance

- With each page, OS maintains a counter to indicate the number of sweeps that page has gone through.
- On page fault, OS checks accessed bit:
  - If 1, then clear it, and also clear the counter.
  - If 0, then increment the counter; if count == N, replace page.

Large N implies better approximation to LRU:
e.g., N = 1000 is a very good LRU approximation.
However, a large N implies more work by the OS before a page can be replaced.

Decent approximations to LRU, assuming that past is a good predictor of the future

N = 1 implies the default clock algorithm.

https://www.cse.iitd.ernet.in/~sbansal/os/lec/l30.html#:~:text=Nth%20chance%3A%20The%20clock%20algorithm,N%20chances%20before%20evicting%20it.

Still remember the PTE?

Core i7 Level 4 Page Table Entries

63	62–52	51–12	11–9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address	Unused	G		D	A	CD	WT	U/S	R/W	P=1

63–1	0
Available for OS (for example, if page location on disk)	P=0

Each entry references a 4K child page. Significant fields:

P: **Virtual** page is present in memory (1) or not (0) — It's set **only if** page is in memory

R/W: Read-only or read-write access permission for this page — Program can read page, but not modify it

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

A: Reference bit (set by MMU on reads and writes, cleared by software) — Set when page referenced; cleared by an algorithm like CLOCK

D: Dirty bit (set by MMU on writes, cleared by software) — Set when page modified; cleared when page written to disk

Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

XD: Disable or enable instruction fetches from this page.

Thrashing

Process requires more memory than the system has

Each time a page is brought in, another page, whose contents will soon be referenced, is thrown out

A program touches 50 pages (each equally likely) but only have 40 physical page frames

If we have enough physical pages, 100ns/ref
If we have too few physical pages, assuming every 5th reference leads to a page fault, then:
4 ref * 100 ns + 1 page fault * 10ms for disk I/O
This lead to 5 refs per (10ms + 400ns) ~ 2ms/ref = **20,000x slowdown!**

Thrashing

Process requires more memory than the system has

Each time a page is brought in, another page, whose contents will soon be referenced, is thrown out

What we want: virtual memory the size of disk with access time the speed of physical memory

What we have: memory with access time roughly at the same magnitude as disk access

Note: this issue is not limited to page access, but we are discussing this issue in the context of page access

Thrashing - what are the causes?

What we want: virtual memory the size of disk with access time the speed of physical memory

What we have: memory with access time roughly at the same magnitude as disk access

process don't reuse memory (no temporal locality)
OR
process reuses memory but the memory that is absorbing most of the accesses doesn't fit

Each processes fit the memory individually, but too much to fit for all processes in the system!

Thrashing - What do we do?

Each processes fit the memory individually, but too much to fit for all processes in the system!

Working Set

The pages a process has touched over some trailing window of time

Only run a set of processes s.t. the union of their working sets fit in memory

Page fault frequency

Track the metric (# page faults/instructions executed)

If that thing rises above a threshold, and there is not enough memory on the system, swap out the process

How to review for midterms?

- The scope for midterm: **everything we covered so far**
- Everything means: lectures (up to today's lectures), handouts, readings, homework, labs (1-3)
- All homework solutions have released
- Make sure you **understand** everything we covered, exams will test your understanding.
- The past exam questions are on the websites with solutions
- **Cheatsheet**: You may refer to ONE two-sided letter-sized sheet that is written or typed by yourself.