Pages Replacement Algorithms

OPERATING SYSTEMS COURSE
THE HEBREW UNIVERSITY
SPRING 2022

Replacement Algorithms

Replacement Algorithms (1)

- Loading and evicting a page takes time
- We want to lower the "page miss rate"
- Typically, all frames are filled with pages
- Which pages should be evicted from their frame?
- Page miss →
  need to bring new page into some frame →
  need to evict a page from a frame
- Evicted page / frame is called **victim**

Replacement Algorithms (2)

- How do we choose *victim*?
  - We can just choose at random…
  - But better not to choose a page that's used often – we'll probably need to bring it back in soon
  - When using a Hierarchical Page Table, better pick an actual page instead of a non-empty table page – why?
  - We'll ignore the implementation for now (Single \ Hierarchical \ Inverted Table) and discuss only replacing the actual pages, i.e. not the table pages.

Replacement Algorithms (3)

Replacement Algorithms (3)

- Many policies are possible
  - Optimal
  - FIFO (first-in-first-out)
  - Second Chance FIFO
  - NRU (not recently used)
  - LRU (least recently used)
  - Pseudo-LRU
  - LFU (least frequently used)
  - Random
  - Etc

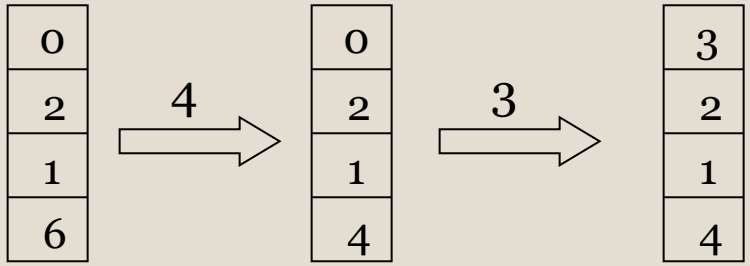
Optimal Page Replacement

Optimal Page Replacement

- Often called Bélády's Optimal Page Replacement Policy
- Basic idea - replace the page that will not be used for the longest time
- This gives the lowest possible fault rate
- Impossible to implement
- Does provide a good measure for other techniques

Optimal Page Replacement

- Consider the following reference string:
  0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1



- Fault Rate = 6 / 12 = 0.50
- With the above reference string, this is the best we can hope to do

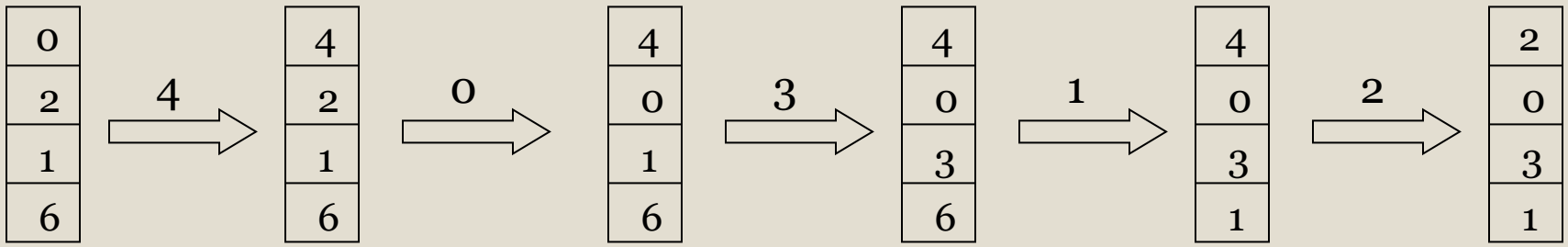
FIFO – First In, First Out

FIFO – First In First Out

- The oldest page in the RAM is the one selected for replacement
- Very simple to implement

FIFO – First In First Out

- Consider the following reference string:
  0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1



- Fault Rate = 9 / 12 = 0.75

FIFO Issues

Poor replacement policy -

- Does not consider block usage
- Evicts the oldest page in the RAM
  - Usually a heavily used variable should be around for a long time
  - FIFO replaces the oldest page - perhaps the one with the heavily used variable

Second Chance FIFO

Second Chance FIFO

Second Chance FIFO

- Each page has a reference bit R
  - When page is accessed R=1.

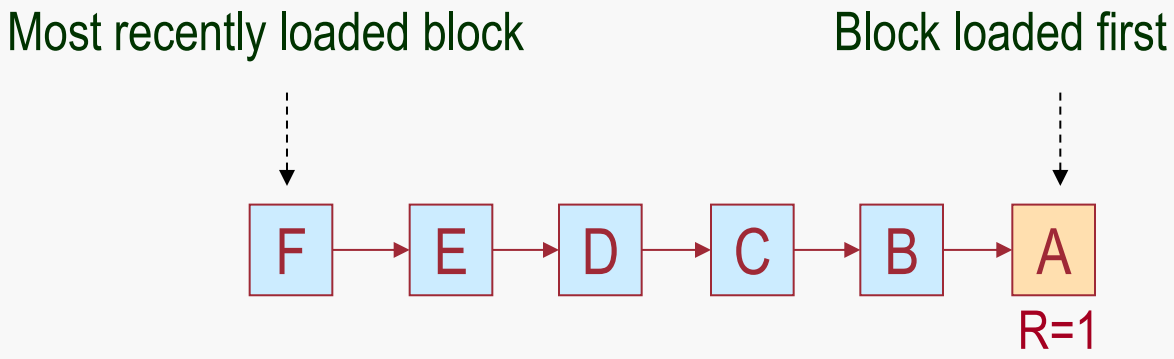
Second Chance FIFO

- Each page has a reference bit R
  - When page is accessed R=1.
- Works like FIFO, but with second chance: If the page to be replaced has R=1, then
  - Leave page in memory; set R=0
  - Move page to the end of the queue
  - Apply same rules to the next block

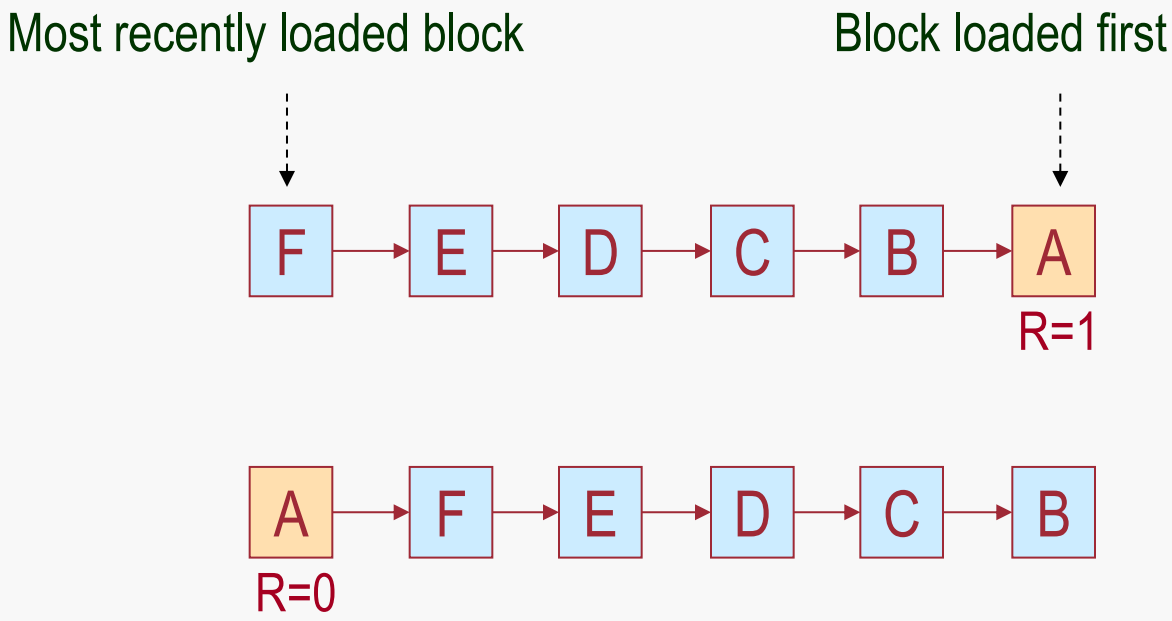
Second Chance FIFO

- Each page has a reference bit R
  - When page is accessed R=1.
- Works like FIFO, but with second chance: If the page to be replaced has R=1, then
  - Leave page in memory; set R=0
  - Move page to the end of the queue
  - Apply same rules to the next block
- Inefficient because it is constantly moving blocks around on its list.

Second Chance FIFO



Second Chance FIFO



A is treated like a
newly loaded block

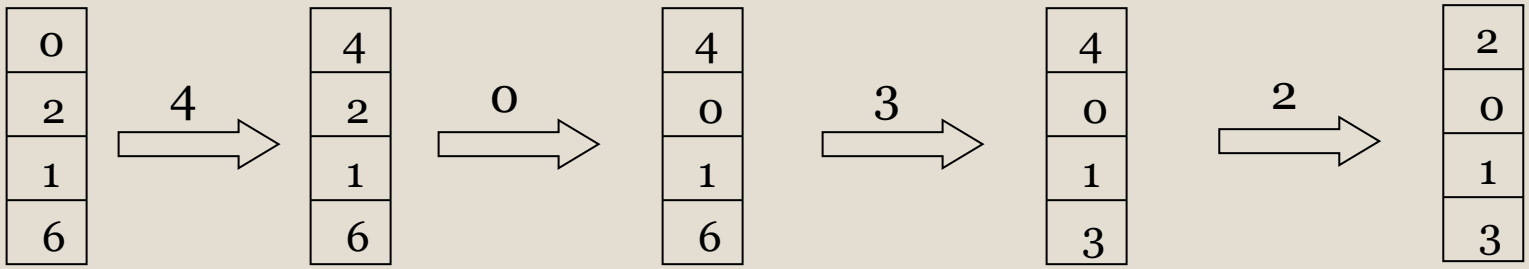
LRU – Least Recently Used

Least Recently Used (LRU)

- Basic idea
  - replace the page that has not been accessed for the longest time
- Optimal policy looking back in time
  - as opposed to forward in time
  - fortunately, programs tend to follow similar behavior

Least Recently Used (LRU)

- Consider the following reference string:
  0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1



- Fault Rate = 8 / 12 = 0.67

LRU Issues

- How to keep track of last page access?
  - requires special hardware support
- 2 major solutions
  - counters
    - hardware clock “ticks” on every memory reference
    - the page referenced is marked with this “time”
    - the page with the smallest “time” value is replaced
  - stacks
    - keep a stack of references
    - on every reference to a page, move it to top of stack
    - page at bottom of stack is next one to be replaced

LRU Issues

- Both techniques require additional hardware
  - remember, memory reference are very common
  - impractical to invoke hardware on every memory reference
- LRU is not used very often, but it is similar to many other algorithms –
  - Pseudo-LRU - approximates LRU
  - NRU - looks at the usage in the previous time step only
  - Many other variants and hybrids

Replacement Algorithms - Summary

- Our physical memory is bounded
- We need to evict pages so we could load new ones
- Ideal Case - evict the ones that won't be used for the longest time
- But we can't know the future – we have to use heuristics
- Usually, we rely on temporal and special locality, but it's just a heuristic
  - Plus, it has many real-world failure cases

Questions from Exams

Address Translation – Question A

Consider a machine with:
Physical memory of 8 GB
Page size of 8 KB
Page table entry size of 4 bytes
Every Page table fits into a single page

How many levels of page tables would be required to map a 46-bit virtual address?

Address Translation – Answer A

Physical memory of 8 GB
Page size of 8 KB
Page table entry size of 4 bytes
Every Page table fits into a single page
How many levels of page tables would be required to map a 46-bit virtual address?

Since each PTE is 4 bytes and each page contains 8KB, then a one-page page table would point to 2048 or 2^{11} pages, addressing a total of $2^{11} * 2^{13} = 2^{24}$ bytes. Continuing this process:

Depth	*Address Space*
1	2^{24} *bytes*
2	2^{35} *bytes*
3	2^{46} *bytes*

Address Translation – Question B

List the fields of a Page Table Entry (PTE) in your scheme

Address Translation – Question B

List the fields of a Page Table Entry (PTE) in your scheme

Each PTE will have a pointer to the proper page, plus several bits – read, write, execute, and valid. This information can all fit into 4 bytes, since 20 bits are needed to point to the proper page, leaving ample space (12 bits) for the information bits.

Address Translation – Question C

Without a cache, how many memory operations are required to read or write a single 32-bit word?

Address Translation – Question C

Without a cache, how many memory operations are required to read or write a single 32-bit word?

Without extra hardware, performing a memory operation takes 4 actual memory operations: 3 page table lookups in addition to the actual memory operation.

Address Translation – Question D

How much physical memory is needed for a process with three pages of virtual memory (for example, one code, one data, and one stack page)?

Address Translation – Answer D

- We need:
  - three frames for the process' three pages
  - one frame for one page of the third level page table
  - one frame for one page of the second-level page table
  - one frame for the first-level page table
- In total, we need six frames, which are 48KB
- The fact that most of the table isn't used is crucial – each table points to 16MB (2^11 entreis of size 8 KB)

Pages Replacement – Question A

התבונן בקטע קוד הבא שמאפס מערך של מספרים מסוג integer (כל integer הינו בגודל 4 בתים).

```
for (int i=0; i<2^29; i++) {
  numbers[i] = 0;
}
```

הנחות:

- למחשב זיכרון פיזי בגודל 2^{32} בתים המחולק למסגרות בגודל 2^{12} בתים. שימו לב: כל מסגרת מכילה עד 2^{10} איברים מסוג integer.
- איברי המערך מוקצים בצורה רציפה בתוך כל דף ומתחילים מתחילת הדף הראשון.
- הקוד כולו נכנס בדף אחד.
- שיטת החלפת דפים הינה demand paging ו- i נמצא ברגיסטר.
- בהתחלה הזיכרון מכיל רק את טבלאות הדפים והן נשארות תמיד בזיכרון (התעלמו מ-page faults על טבלאות הדפים).

א. (6 נק') נניח שהחלפת דפים מתבצעת במדיניות LRU. כמה page faults יהיו במהלך האלגוריתם אם מוקצים לתהליך 2^{12} מסגרות בזיכרון (לא כולל טבלאות דפים)? נמק.

Pages Replacement – Answer A

א. (6 נק') נניח שהחלפת דפים מתבצעת במדיניות LRU. כמה page faults יהיו במהלך האלגוריתם אם מוקצים לתהליך 2^{12} מסגרות בזיכרון (לא כולל טבלאות דפים)? נמק.

מערך numbers הוא בגודל 2^{29} איברים מסוג integers כלומר בגודל 2^{19} דפים. בתחילת הרצת התוכנית אף דף אינו בזכרון ולכן יש להביא את כל הדפים הנ"ל לזכרון. כמו כן, צריך להביא לזכרון את דף ה-code (ע"פ הנתון הקוד הוא בדף בודד). הבאת כל דף גוררת page fault ולכן סה"כ מספר ה page-faults הוא $2^{19}+1$.

מדיניות ה-LRU משנה רק לגבי הקוד (ברגע הבאת דף חדש מהמערך, לא נשתמש יותר במילא בכל הדפים הקודמים). מכיוון שאנחנו משתמשים בקוד כל הזמן הוא ישאר בזכרון כל הזמן ולא יגרור page faults נוספים.

Pages Replacement – Question B

ב. (6 נק') נניח כעת שהחלפת דפים מתבצעת במדיניות Second Chance
FIFO. כמה page faults יהיו במהלך האלגוריתם אם מוקצים לתהליך 2^{12}
מסגרות בזיכרון (לא כולל טבלאות דפים)? נמק.

Pages Replacement – Answer B

ב. (6 נק') נניח כעת שהחלפת דפים מתבצעת במדיניות Second Chance FIFO. כמה page faults יהיו במהלך האלגוריתם אם מוקצים לתהליך 2^{12} מסגרות בזיכרון (לא כולל טבלאות דפים)? נמק.

כל ה-page faults מסעיף א' יקרו גם במקרה זה.
יתרה מזאת, לאחר האיטרציה ה-2^{12}-1 יתמלא הזכרון המוקצה לתהליך באופן הבא: דף אחד לקוד, ו-2^{12}-1 דפים למערך. מכיוון שלא פנינו או ניסינו לפנות דפים עד כה, ה-reference bit של כל הדפים הוא 1. ולכן האלגוריתם יתן הזדמנות שניה <u>לכל הדפים</u>. כלומר, למעשה, יאפס את ה-reference bit של כל הדפים, ולבסוף יוציא את הדף הראשון (הקוד) מהתור. מכיוון שצריך להשתמש בקוד, יתווסף עוד page fault להכנסה מחודשת של הקוד.
תופעה זו תקרה כל 2^{12}-1 איטרציות ולכן סה"כ יתווספו $\lceil 2^{19}/(2^{12}-1) \rceil = 2^7$ page faults, וסה"כ מספר ה-page faults יהיה $2^{19}+2^7$.