Replacement policies for multiprogramming

Hardware support for page replacement algorithms

- Enhanced Second Chance
- Second Chance

LRU approximations:

Today

and improve performance

A good page replacement algorithm can reduce the number of page faults

switch occurs.

- Processes can share memory more efficiently, reducing the costs when a context switch occurs.
- Processes start faster because they only need to load a few pages (for code and data)

- Processes can run without being fully loaded into memory.

- Virtual address space can be larger than physical address space.

Benefits of demand paging:

Last Class: Demand Paged Virtual Memory
Why?

- With LRU, increasing the number of frames always decreases the number of page faults.

LRU:

Adding Memory with LRU

- With FIFO, the contents of memory can be completely different with a different number of page frames.

FIFO:

Does adding memory always reduce the number of page faults?

Adding Memory
Page fault still requires a search through all the pages.

Farther, since setting a single bit on each memory access,

Approximate, since it does not guarantee a total order on the pages.

On a page fault, the lowest numbered page is kicked out:

- High order bit.
- At regular intervals or on each memory access, shift the byte right, placing a 0 in the empty.

Additional Reference Bits: Maintain more than 1 bit, say 8 bits.

Hardware Requirements: Maintain reference bits with each page.

Implementations of LRU

Memory access (in the worst case)

- Problems: still too expensive, since the OS must modify 6 pointers on each page.
- On a page access, move the page to the front of the list. Doubly link the list.
- On a page access, move the page to the front of the list.
- On a page access, move the page to the front of the list.
- On a page access, move the page to the front of the list.
- On a page access, move the page to the front of the list.

2. Keep a list of pages, where the front of the list is the most recently used page, and the end is the least recently used.

- Problems: OS must record time stamp for each memory access, and to throw out

1. Keep a time stamp for each page, with the time of the last access, throw out the

Perfect LRU

All implementations and approximations of LRU require hardware

Implementations of LRU
What if all bits are 1? Will it always find a page?

- Simple hardware requirements.
  - A reference bit.

  Page fault is faster since we only search the pages until we find one with
  - Shift.

  Fast, since setting a single bit on each memory access, and no need for a
  - The algorithm.

  Indicates if the page was used at all since the last time it was checked by
  - Less accurate than additional-reference bits, since the reference bit only

Second Chance Algorithm

---

2. On a page fault, the OS

1. OS keeps frames in a circular list:

   Use a single reference hit per page.

Second Chance Algorithm (a.k.a. Clock)
0. Page is the same as the copy on disk.
1. Page is modified (different from the copy on disk)

Hardware keeps a modify bit (in addition to the reference bit)

- OS can give preference to pages out of un-modified pages
- OS need not write the page back to disk

It is cheaper to replace a page that has not been written

Enhanced Second Chance

Why not partition pages into more than two categories?

Young and old pages

One way to view the clock algorithm is as a crude partitioning into two categories:

Clock Example
- By the third pass, all the pages will be at (0,0).
- A (0,1) page is treated as on the first pass.
- \( P \) is being written out. Waits for the I/O to complete and then remove the
- changed to (0,0) \( P \) to the page if the reference bit is cleared.
- On the second pass, a page that was originally (0,0) or (1,0) might have been
- cleared the modified bit and continue the search
- until the I/O completes. Clear the modified bit, and continue the search.

The OS goes around at most three times searching for the (0,0) class.

**Page Replacement in Enhanced Second Chance**

Department of Computer Science, Uppsala University

---

**nonempty class:**

- On a page fault, the OS searches for the first page in the lowest
- where it can replace a page.

- 4. (1,0) recently used and modified - probably will be used again soon and the OS
- not write it out before replacing it
- 3. (1,0) recently used and unmodified - probably will be used again soon, but OS
- write out this page, but it might not be needed anymore.
- 2. (0,1) not recently used but modified - not as good to replace. Since the OS
- 1. (0,0) neither recently used nor modified - replace this page

**Enhanced Second Chance**

Department of Computer Science, Uppsala University
Disadvantages: Threshing might become even more likely (why?)

Advantages: Flexible, adjusts to diverse process needs

Threshing: the memory is over-committed and pages are continuously tossed out while they are still in use

Replacement Policies for Multiprogramming

Clock Example
Disadvantages: The OS has to figure out how many pages to give each process and if the working set size grows dynamically adjust its allocation.

Advantages: Threshing is less likely as processes only compete with itself.

- More consistent performance independent of system load.
- May need to suspend a process until overall memory demands decrease.
- The time it takes to handle a page fault.
- If the page fault frequency > a second threshold, take away some page frames.
- If the page fault frequency < some threshold, give it more page frames.
- Each process instead.

Working sets are expensive to compute so track page fault frequency of each process.

Per-Process Replacement

Replacemnt Policies for Multiprogramming

What happens if T is too small? Too big?

- Needs to be a whole lot bigger than 2 million instructions.
- 10 million = 2 million instructions
- 1 page fault = 10 milli

How does the OS pick T?

- More formally, it is the set of all pages that a process referenced in the past T seconds.
- Informally, the working set is the set of pages the process is using right now.

How do we figure out how many pages a process needs, i.e., its working set size?

Run only groups of processes that fit in memory, and kick out the rest.

Per-Process Replacement: Each process has its own pool of pages.
A critical issue the OS must decide is how many processes and the frames process can have.

The more processes running concurrently, the less physical memory each process approaches the virtual memory size.

All algorithms approach optimal as the physical memory allocated to a insufficient physical memory (less than half of their virtual address space).

Experiments show that all algorithms do poorly if processes have

UNIX and LINUX use Variants of Clock, Windows NT uses FIFO.

### Summary of Page Replacement Algorithms

- **LRU**
  - Larger slow down than they used to. Reducing the number of page faults is critical to
  - CPU speed is increasing faster than disk speed. As a result, page faults result in a
  - Also, internal fragmentation is less of a concern with abundant memory.
  - Physical memory is cheap. As a result, page tables could get huge with small pages.

- **FIFO**
  - For page faults to processes that exhibit locality of references
  - Amortizes disk overheads over a larger page
  - Smaller page tables

### Reasons for Large Pages

- Higher degree of multiprogramming possible.
  - More effective memory use.

### Reasons for Small Pages

- Page sizes are growing because:
  - Power page faults (for processes that exhibit locality of references)

---