COMP2004 Programming Practice 2002 Summer School

Kevin Pulo
School of Information Technologies
University of Sydney

Assignment 3

- Paging simulation

- Background
- Your task
- Paging algorithms
  - First In First Out (FIFO)
  - Least Recently Used (LRU)
  - Pipelined LRU (PLRU)

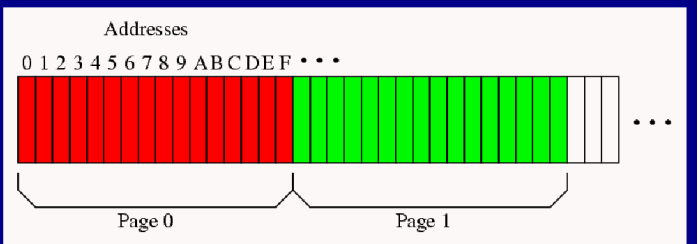
Background

- Read information from disk to memory as needed
  - Disk is slow
  - Memory is expensive

Addresses and pages

- Address
  - Specifies a byte on disk (>= 0)
- Page
  - A block of bytes on disk
- Page number
  - Specifies a page on disk (>= 0)
- Page size
  - Number of bytes per page
  - All pages have the same size

Pages diagram



- Each small box is a byte
  - Each has an address along the top
- A page is a block of bytes
  - Numbered from 0
- This diagram has a page size of 16

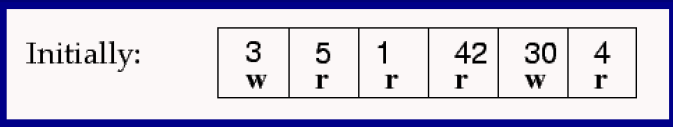
Finding page numbers

- For page size of 512 bytes:
  - Page 0 is addresses 0 to 511
  - Page 1 is addresses 512 to 1023
  - Page 2 is addresses 1024 to 1535
  - ...
- page number = address / page size
  - Using integer division
- Eg: page size of 4096, address 372921
  - Page number = 372921 / 4096
    = 91.045166...
    = 91

Frames

- Pages are read into frames
  - Pages are on disk
  - Frames are in memory
- A collection of frames is called a cache
  - Has only few frames
  - Compared to number of pages
  - ie. number of frames is limited
  - Stores only the pages needed right now

Cache diagram



- Each box is a frame
- Inside box:
  - Number indicating the page stored in that frame
  - r/w indicating frame clean/dirty
    - We'll get to this later

Accessing information

- As a program runs it needs to access data from disk
- Causes pages to be loaded into frames
- Program then accesses data in frame
- Entire page loaded into frame
- Each page present in at most one frame

Loading pages

- When a page is required, it may:
  - already be in a frame - cache hit
  - not be in a frame - cache miss
    - Requires page to be loaded into a frame before it can be used

Cache misses

- Initially all frames unused
  - Can load a page into any unused frame
- When all frames used
  - Must remove a frame before loading page
- Paging algorithm
  - Decides which frame to remove

Writing to frames

- Data access can be read or write
  - Read doesn't modify data value
  - Write may modify data value
- Writing causes data in frame to change
  - But not in page on disk
- Now data in frame and page differ
- Data in frame is more recent
- Frames in this state are dirty
- Frames identical to pages on disk are clean

Cache diagram revisited

Initially:	3 w	5 r	1 r	42 r	30 w	4 r

- `r` indicates frame is clean
- `w` indicates frame is dirty

Write-back

- What happens if a dirty page is removed?
- Changes in frame must be put back to disk
  - This is called write-back
- Otherwise changes would be lost

Types of cache miss

- Cache miss causes frame to be removed
- Cache miss without write-back
  - Frame removed is clean
  - Unused frame available
- Cache miss with write-back
  - Frame removed is dirty

Your task

- Imagine a series of read/write operations
- Each results in one of
  - cache hit
  - cache miss without write-back
  - cache miss with write-back

Your code

- Is told about each operation in turn
- Simulates the paging algorithm
  - No need to actually read/write pages to/from disk/memory
  - Store which page is in each frame
  - And which frames are dirty
- Counts number of
  - cache hits
  - cache misses without write-back
  - cache misses with write-back

Classes

- Don't write a main() program
- One is supplied
  - Takes care of input/output
  - Calls methods in your class
  - Must be used
- Write the Cache and CacheFactory classes
- Must be defined in Cache.h and CacheFactory.h files

Creating Cache objects

- How Cache objects are created is up to you
- Will be created with
- `string s; getline(cin, s);`
- `Cache *c = CacheFactory::createCache(s);`
- Takes a string, constructs an appropriate Cache object
- Returns a pointer to that newly constructed object

Creation string format

- Creation string is the first line read by the main program
- Has format:
  - F *page_size num_frames*
  - L *page_size num_frames*
  - P *page_size num_frames lookahead check_frames*
- *page_size*, *num_frames*, *lookahead*, *check_frames* are integers

Main loop

```
char mode;
Cache::addr_t address;
while (cin >> mode >> address) {
    if (mode == 'r') {
        c->read(address);
    } else if (mode == 'w') {
        c->write(address);
    }
    outputStats(*c);
}
```

Cache typedefs

- Various typedefs defined in the Cache class
- Each has an intended usage
- Cache::addr_t
  - Stores an address
- Cache::page_t
  - Stores a page number
- Cache::counter_t
  - Stores a counter
  - eg. num of cache hits, etc

Output

```
void outputStats(const Cache &c) {
    Cache::counter_t hits = c.getHits(),
        misses = c.getMisses(),
        missesWB = c.getMissesWB();

    cout << hits << " " <<
            misses << " " <<
            missesWB << endl;
}
```

- Performed after every read/write operation and at end of program

Paging algorithms

- First In First Out (FIFO)
- Least Recently Used (LRU)
- Pipelined LRU (PLRU)

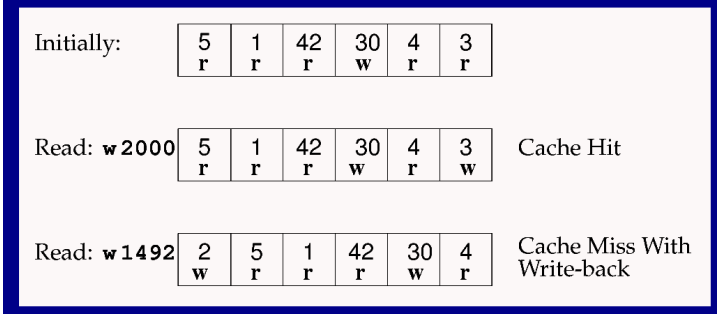
- This order is easiest to hardest
- For automarking:
  - FIFO : 40%
  - LRU : 40%
  - PLRU : 20%

First In First Out (FIFO)

- Very simple
- Remove frame which has been in the cache for the longest time
- Can think of the cache as a list/queue
  - New frames go at front
  - Old frames removed from back
- Don't have to store it as a list/queue
  - Can store however you like
  - Choose an efficient method

FIFO Example

- num_frames = 6, page_size = 512
- `w 2000` (page 3)
- `w 1492` (page 2)

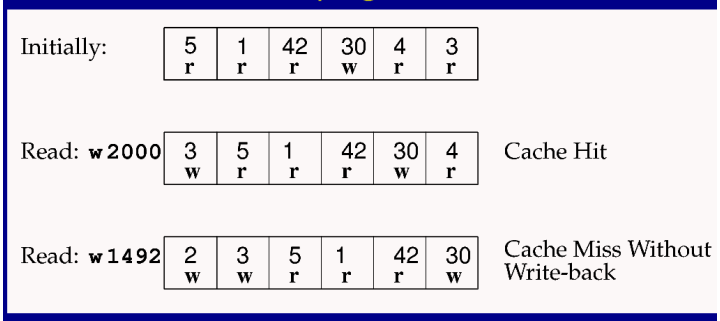


Least Recently Used (LRU)

- Improves on FIFO
- When a frame is accessed (cache hit)
  - Moved to the front of the list/queue
- Now recently used frames are at front
- Frames not recently used are at back
- Still removes frame from back
  - The least recently used frame

LRU Example

- num_frames = 6, page_size = 512
- `w 2000` (page 3)
- `w 1492` (page 2)



Pipelined LRU (PLRU)

- Hardest/most complex
  - Worth less than FIFO and LRU
  - Therefore attack it last
  - Don't let it ruin your design
- Improves on LRU
- Doesn't remove frames which will be needed soon
- Acheives this by examining the upcoming read/write operations

Pipelining and lookahead list

- Lookahead list is a list of upcoming read/write operations
- Has max length given by *lookahead*
- Best way to understand is with an example
  - Lookahead list length will be 4
  - Lookahead list shown in yellow
  - `*` indicates current input position
  - `+` indicates operation just processed