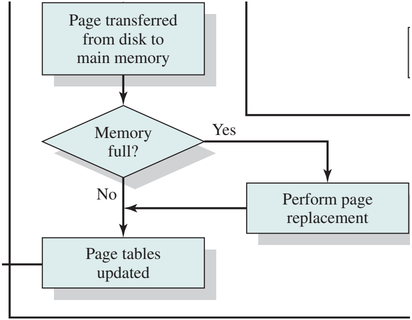


CSL 301

OPERATING SYSTEMS

Lecture 14

Beyond Physical Memory

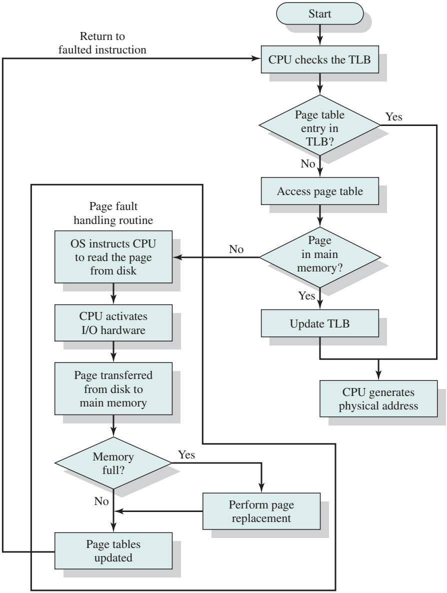
Page Replacement Policies

Instructor

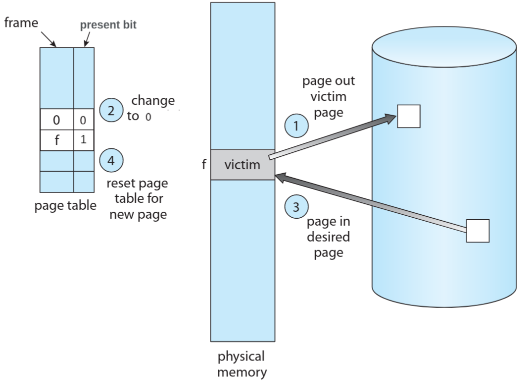
Dr. Dhiman Saha

Image Source: Google

Why Page Replacement?



The Scenario



Page Replacement Policy = How to find victim page

Page Replacement Policy

Deciding which page (or pages) to evict is encapsulated within the replacement policy of the OS.

- Historically, it was one of the most important decisions the early virtual memory systems made, as older systems had little physical memory.

How can the OS decide which page (or pages) to evict from memory?

Memory as a cache for virtual memory pages

- Main memory holds some **subset** of all the pages in the system

Replacement Policy | **Goal for cache**

To minimize the number of **cache misses**, i.e., to minimize the number of times that we have to **fetch a page from disk**

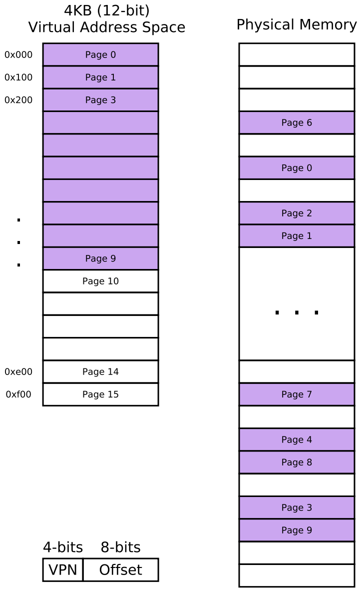
$$AMAT = T_M + P_{miss} \cdot T_D$$

- T_M represents the cost of accessing memory,
- T_D the cost of accessing disk, and
- P_{miss} is the probability of not finding the data in the cache
  - **A Miss**

Example

4KB Address Space
256 byte pages

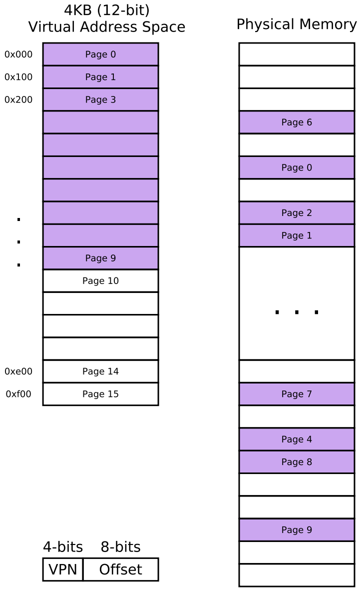
- Process can access 2^4 virtual pages
- Process generates following memory references: $\texttt{0x000}, \texttt{0x100}, \cdots, \texttt{0x900}$
- Refer to first byte of each of first 10 pages



Example

4KB Address Space
256 byte pages

- Process can access 2^4 virtual pages
- Process generates following memory references: $\texttt{0x000,0x100,}\cdots\texttt{,0x900}$
- Refer to first byte of each of first 10 pages
- Assume Page 3 not in memory



Example

- Memory reference sequence: h (hit), m ($miss$)

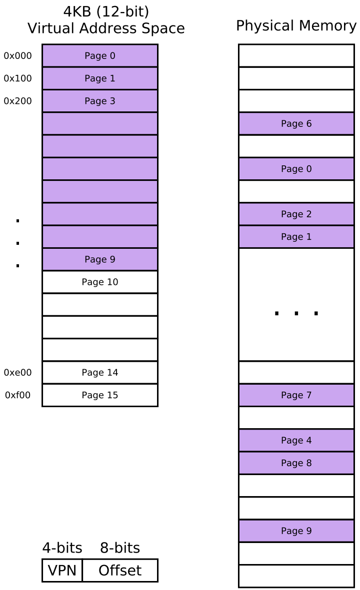
$$h, h, h, \color{red}{m}, h, h, h, h, h$$

- hit-rate

$$\frac{\#hit}{\#hit + \#miss} = 0.9\ (90\%)$$

- If $T_D = 10ms$, $T_M = 100ns$

$$\begin{aligned} AMAT &= T_M + P_{Miss} \cdot T_D \\ &= 100ns + 0.1 \cdot 10ms \\ &= 1.0001ms \end{aligned}$$



- Memory reference sequence: h (hit), m ($miss$)

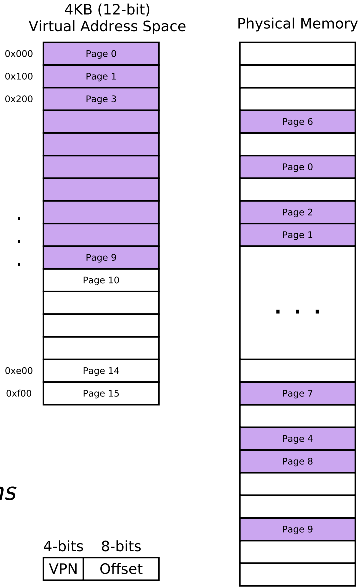
$$h, h, h, \color{red}{m}, h, h, h, h, h$$

- hit-rate

$$\frac{\#hit}{\#hit + \#miss} = 0.9\ (90\%)$$

- If $T_D = 10ms$, $T_M = 100ns$

$$\begin{aligned} AMAT &= T_M + P_{Miss} \cdot T_D \\ &= 100ns + 0.001 \cdot 10ms \\ &= 10.1\mu s \end{aligned}$$



---

100 times faster that $P_{Miss} = 0.1$

Need for Smart Replacement Policies

T_D dominates $AMAT$ even for a low P_{Miss}

Goal: Avoid as many misses as possible

The Optimal Replacement Policy

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State
0			
1			
2			
0			
1			
3			
0			
3			
1			
2			
1			

Tracing The Optimal Policy

- Optimal w.r.t what?
- Minimum cache misses possible in this stream
- Assume you know the future 😄

What is the use of such a policy?

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	2	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	3	0, 1, 2
1	Hit		0, 1, 2

Tracing The Optimal Policy

- What is the hit-rate?
- Recompute discarding the first 3 misses. Why?
- See next slide

What is the use of such a policy?

- Compulsory Miss (or Cold-Start Miss)
  - First reference to the item
- Capacity Miss
  - The cache ran out of space
- Conflict Miss
  - Arises due to set-associativity
  - Not applicable in this context
  - In our case it is fully associative

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State
0			First-in→
1			First-in→
2			First-in→
0			First-in→
1			First-in→
3			First-in→
0			First-in→
3			First-in→
1			First-in→
2			First-in→
1			First-in→

Tracing The FIFO Policy

What is the hit-rate w.r.t optimal?

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		First-in→	0
1	Miss		First-in→	0, 1
2	Miss		First-in→	0, 1, 2
0	Hit		First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2
3	Miss	0	First-in→	1, 2, 3
0	Miss	1	First-in→	2, 3, 0
3	Hit		First-in→	2, 3, 0
1	Miss	2	First-in→	3, 0, 1
2	Miss	3	First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2

Tracing The FIFO Policy

What is the hit-rate w.r.t optimal?

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	0	1, 2, 3
0	Miss	1	2, 3, 0
3	Hit		2, 3, 0
1	Miss	3	2, 0, 1
2	Hit		2, 0, 1
1	Hit		2, 0, 1

Figure 22.3: **Tracing The Random Policy**

FIFO/Random can evict an important page

Do not take locality into consideration

Need to use history data

- What qualifies as history?
  - Frequency of access
  - Recency of access

Spatial/Temporal **Leverage Principle of Locality**

A heuristic that often proves useful

- The Least-Frequently-Used (LFU) policy
- The Least-Recently-Used (LRU) policy

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State
0			LRU→
1			LRU→
2			LRU→
0			LRU→
1			LRU→
3			LRU→
0			LRU→
3			LRU→
1			LRU→
2			LRU→
1			LRU→

Tracing The LRU Policy

What is the hit-rate w.r.t optimal?

- Assume a program accesses the following stream of virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1.

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1	Hit		LRU→	0, 3, 1
2	Miss	0	LRU→	3, 1, 2
1	Hit		LRU→	3, 2, 1

Tracing The LRU Policy

What is the hit-rate w.r.t optimal?

Belady's Anomaly

Point-to-Ponder

- What if the cache-size is increased?
- Will hit-rate always increase?
- Or does it depend on the policy used?

- Assume a program accesses the following stream of virtual pages: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Access	Hit/Miss?	Evict	Resulting Cache State	Hit/Miss?	Evict	Resulting Cache State
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→

- How does the hit-rate change if cache-size is changed from 3 to 4?

Belady's Anomaly — The LRU Policy

- Assume a program accesses the following stream of virtual pages: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Access	Hit/Miss?	Evict	Resulting Cache State	Hit/Miss?	Evict	Resulting Cache State
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→
			First-in→			First-in→

- How does the hit-rate change if cache-size is changed from 3 to 4?

- General expectation:
  - Cache hit rate to increase when the cache gets larger
- But in this case, with FIFO, it gets worse! -**Belady's Anomaly**
- LRU does not suffer from this problem. Why?
- LRU has what is known as a **stack property**.

Stack Property

For algorithms with this property, a cache of size $N + 1$ naturally includes the contents of a cache of size N. Thus, when increasing the cache size, hit rate will either stay the same or improve.

- FIFO and Random (among others) do not obey the stack property
- So are susceptible to anomalous behavior.

Implementing "Historical" Algorithms

- How to implement LRU/FIFO in practice
- FIFO - relatively easy
  - Use a Queue-like data structure
- What about LRU?
  - Must mark currently referenced page as most-recently used
  - Implies some accounting work on **every** memory reference

One Solution — **With Hardware Support**

- Time-stamping on every page access
- **Using** this info while evicting a page
- What is the issue in this approach?
- Hint: What is the search space?

How to implement LRU in practice?

Perfect LRU is expensive!!!

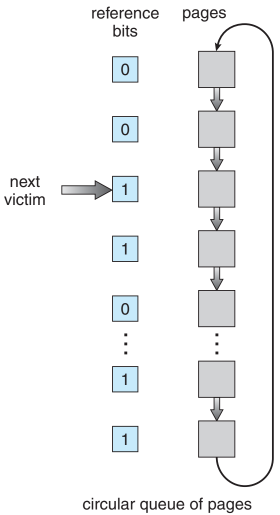
Can we approximate it with desired results?

Idea

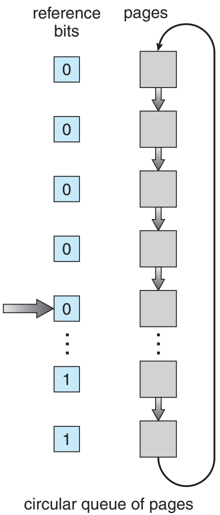
- Differentiate between NRU and not-NRU pages
- Ignore the order

- The **reference (use)** bit
- One-bit/page (somewhere in memory)
  - Can be associated with each entry in the page table
- Initially set to 0 for all pages by OS
- Set to 1 by hardware whenever a page is referenced
- How to use this bit?

Second-Chance Algorithm



Clock Algorithm



The Dirty (Modified) Bit

Intuition

If a page has been modified and is thus dirty, it must be written back to disk to evict it, which is expensive.

- **Dirty** bit captures clean/modified pages
- Bit is set any time a page is written

- Dirty bit can be incorporated into the page-replacement algorithm.
- **Clean** pages preferred for replacement over dirty ones

When to bring a page into memory?

- Page Selection
  - Demand paging
  - Pre-fetching

When to write pages out to disk?

- Page write-back
  - Clustering/grouping pages to be written
  - Better than one-at-a-time

Thrashing

What if

Memory demands of the set of running processes **simply exceeds** the available physical memory?

- Meaning **working sets**[1] of processes is not fitting in memory
- In this case, the system will constantly be **paging**, a condition sometimes referred to as **thrashing**

- Possible solutions:
  - Admission control
  - Out-of-memory killer daemon (ft. Some Linux distros)

Read from OSTEP workload vs replacement policy behavior

[1] The set of pages being actively used by a process