Operating Systems and Concurrency

Lecture 16: Memory Management V

G52OSC

Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

- Virtual memory relies on **localities** which constitute **groups of pages** that are **used together**, e.g., related to a function (code, data, etc.)
  - Processes move **from locality to locality**
  - If all required pages are **in memory**, **no page faults** will be generated
- **Page tables** become **more complex** (present/absent bits, referenced/modified bits) and **larger** (e.g. multi-level)

Recall

Page Tables Revisited: Multi-level Page Tables

- **Solution**: Page the page table!
- We keep tree-like structures to hold page tables

- Divide the page number into
  - An index to a page table of second level
  - A page within a second level page table

- No need to keep all page tables in memory all time

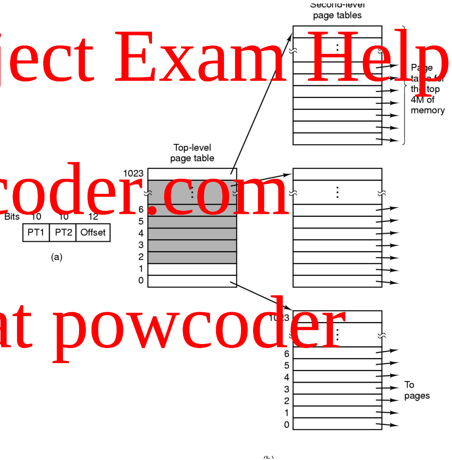

Figure: Multi-level page tables (from Tanenbaum)

Goals for Today

Overview

- **Page tables**: maintaining **performance** and **inverted page tables**
- Several **key decisions** have to be made when **using virtual memory**

  - When are pages **fetched** ⇒ demand or pre-paging
  - **What pages** are **removed** from memory ⇒ page **replacement algorithms**
- The **optimal** and **FIFO** page replacement algorithm

- **Memory organisation** of multi-level page tables:
  - The **root page table** is always maintained in memory
  - Page tables themselves are maintained in **virtual memory** due to their size
- Assume that a **fetch** from main memory takes T nano seconds
  - With a **single page table level**, access is $2 \times T$
  - With **two page table levels**, access is $3 \times T$
  - ...

Virtual Memory

Access Speed on [multi-level] Page Tables

- **Memory organisation** of multi-level page tables:
  - The **root page table** is always maintained in memory
  - Page tables themselves are maintained in **virtual memory** due to their size
- Assume that a **fetch** from main memory takes T nano seconds
  - With a **single page table level**, access is $2 \times T$
  - With **two page table levels**, access is $3 \times T$
  - ...
- With two levels, every memory reference already becomes **3 times slower**:
  - Assuming that the second level page table is **already in main memory**
  - Memory access already forms a **bottleneck** under normal circumstances

- **Translation look aside buffers** (TLBs) are (usually) located inside the memory management unit
  - They **cache** the most frequently used page table entries
  - They can be searched **in parallel**

- The principle behind TLBs is similar to other types of **caching in operating systems**
- Remember: **locality** states that processes make a large number of references to a small number of pages

Virtual Memory

Translation Look Aside Buffers (TLBs)

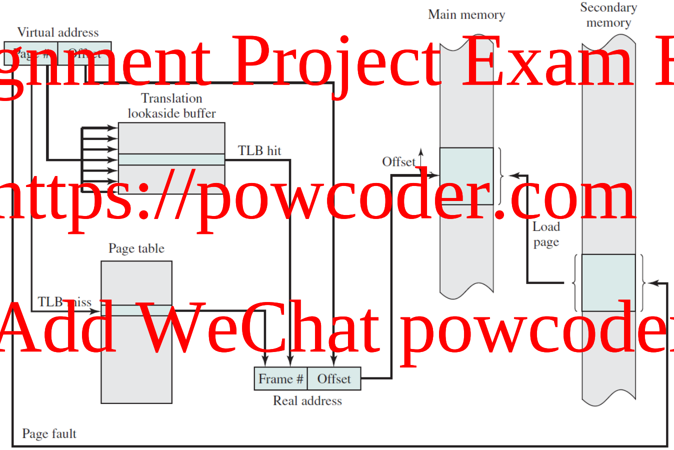

Figure: TLB Address Translation with a single-level page table(from Stallings)

- Memory access with TLBs:
  - Assume a single-level page table
  - Assume a 20ns associative **TLB lookup**
  - Assume a 100ns **memory access time**
    - **TLB Hit** $\Rightarrow$ 20 + 100 = 120 ns
    - **TLB Miss** $\Rightarrow$ 20 + 100 + 100 = 220 ns
- Performance evaluation of TLBs:
  - For an 80% hit rate. the estimated access time is: $120 \times 0.8 + 220 \times (1 - 0.8) = 140$ns (i.e. 40% slowdown – relative to absolute addressing)
  - For a 98% hit rate, the estimated access time is: $120 \times 0.98 + 220 \times (1 - 0.98) = 122$ns (i.e. 22% slowdown)
- Note that **page tables** can be **held in virtual memory** $\Rightarrow$ further (initial) slow down due to page faults

- A **"normal" page table's size** is proportional to the number of pages in the virtual address space ⇒ this can be prohibitive for modern machines
- An **"inverted" page table's** size is proportional to the size of main memory

  - The inverted table contains one **entry for every frame** (i.e. not for every page), and it **indexes entries by frame number**, not by page number.
  - When a process references a page, the OS must search the (entire) inverted page table for the corresponding entry (i.e. page and process id) ⇒ this could be too slow.

  - *Solution*: Use a **hash function** that transforms page numbers (n bits) into frame numbers (m bits) - Remember: $n > m$.

- The **frame number** will be the index of the inverted page table
- Process Identifier (**PID**) - The process that owns this page.
- Virtual Page Number (**VPN**)
- **Protection** bits (Read/Write/Execution)
- **Chaining Pointer** - This field points toward the next frame that has exactly the same VPN. We need this to solve collisions.

Frame number	PID	VPN	RWX	Chaining Ptr

Figure: Example of an Inverted Page Table Entry (other info bits are not shown here)

Virtual Memory

Page Tables Revisited: Inverted Page Tables - Address translation

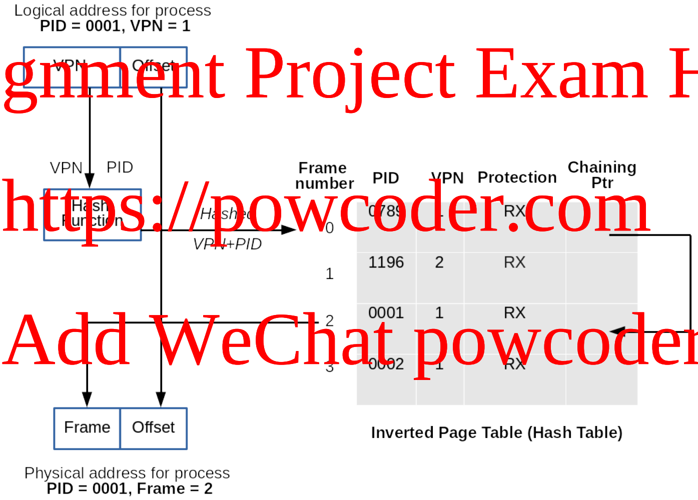

Figure: Address Translation with an Inverted Page Table

- Advantages:
  - The OS maintains a **single inverted page table** for all processes
  - It **saves lots of space**(especially when the virtual address space is much larger than the physical memory)

- Disadvantages:
  - Virtual-to-physical **translation becomes much harder/slower**.
  - Hash tables eliminates the need of searching the whole inverted table, but we have to handle collisions (that will also slow down the translation).

- TLBs are particularly necessary to improve their performance.
- Commonly used on 64-bit machines (e.g. Windows 10)
  `http://answers.microsoft.com/en-us/windows/forum/windows_10-performance/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938`

- Two key decisions have to be made using virtual memory
  - What pages are **loaded** and when $\Rightarrow$ predictions can be made
  - What pages are **removed** from memory and when $\Rightarrow$ **page replacement algorithms**

- **Pages are shuttled** between primary and secondary memory

- **Demand paging** starts the process with **no pages in memory**
  - The first instruction will immediately cause **a page fault**
  - **More page faults** will follow, but they will **stabilise over time** until moving to the **next locality**
  - The set of pages that is currently being used is called its **working set** ($\Leftrightarrow$ resident set)
- Pages are only **loaded when needed**, i.e. following **page faults**

Pre-Paging

Predictive

- When the process is started, all pages expected to be used (i.e. the working set) could be **brought into memory at once**
  - This can drastically **reduce the page fault rate**
  - Retrieving multiple (**contiguously stored**) pages **reduces transfer times** (seek time, rotational latency, etc.)
- **Pre-paging** loads pages (as much as possible) **before page faults are generated** ($\Rightarrow$ a similar mechanism is used when processes are **swapped out/in**)

- Avoiding **unnecessary pages** and **page replacement** is important!
- Let *ma*, *p*, and *pft* denote the **memory access time** (2 times for single-level page tables) (ranging from 10 to 200ns), **page fault rate**, and **page fault time**, respectively, the **effective access time** is then given by:

$$T_a = (1-p) \times ma + pft \times p \tag{1}$$

- Note that we are not considering here TLBs.

- Assuming a single-level page table
- With a memory **access time** of 100ns (10^{-9}) (Therefore, 2 accesses -> 200ns) and a **page fault time** of 8ms (10^{-3})

$$T_a = (1-p) \times 200 + p \times 8000000 \tag{2}$$

  - Recall that access to hard drives is very slow (e.g. at 7200 RPM, half a turn of the hard drive takes about 4.2 mili-seconds)

- The expected/effective access time is **proportional to page fault rate** when keeping page faults into account
  - Ideally, all pages would have to be loaded without demand paging

Page Replacement

Concepts

- The OS must choose a **page to remove** when a new **one is loaded** (and all are occupied)
- This choice is made by **page replacement algorithms** and **takes into account**:
  - When the page is **last used/expected to be used** again
  - Whether the **page has been modified** (only modified pages need to be written)
- Replacement choices have to be **made intelligently** ($\Leftrightarrow$ random) to **save time**/avoid **thrashing**

Page Replacement

Algorithms

1. **Optimal** page replacement
2. **FIFO** page replacement
   - Second chance replacement
   - Clock replacement
3. **Not recently used** (NRU)
4. **Least recently used** (LRU)

Page Replacement
Optimal Page Replacement

- In an **ideal/optimal** world
  - Each page is labeled with the **number of instructions** that will be executed/length of time before it is **used again**
  - The page which **is not going to be not referenced** for the **longest time** is the optimal one to remove

- The **optimal approach** is **not possible to implement**
  - It can be used for **post-execution analysis** $\Rightarrow$ what would have been the minimum number of page faults
  - It provides a **lowerbound** on the **number of page faults** (used for comparison with other algorithms)

Page Replacement
First-In, First-Out (FIFO)

- FIFO maintains a **linked list** and **new pages** are added at the end of the list

- The **oldest page** at the **head of the list** is evicted when a page fault occurs
- The **(dis-)advantages** of FIFO include:
  - It is **easy** to understand/implement

  - It **performs poorly** ⇒ heavily used pages are just as likely to be evicted as a lightly used pages

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1																								
PF2																								
PF3																								
PF4																								

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0																							
PF2	-																							
PF3	-																							
PF4	-																							

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0																						
PF2	-	2																						
PF3	-	-																						
PF4	-	-																						

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)

- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0																					
PF2	-	2	2																					
PF3	-	-	1																					
PF4	-	-	-																					

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0																				
PF2	-	2	2	2																				
PF3	-	-	1	1																				
PF4	-	-	-	3																				

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5																			
PF2	-	2	2	2	2																			
PF3	-	-	1	1	1																			
PF4	-	-	-	3	3																			

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5																		
PF2	-	2	2	2	2	4																		
PF3	-	-	1	1	1	1																		
PF4	-	-	-	3	3	3																		

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5																	
PF2	-	2	2	2	2	4	4																	
PF3	-	-	1	1	1	1	6																	
PF4	-	-	-	3	3	3	3																	

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5																
PF2	-	2	2	2	2	4	4	4																
PF3	-	-	1	1	1	1	6	6																
PF4	-	-	-	3	3	3	3	3																

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5															
PF2	-	2	2	2	2	4	4	4	4															
PF3	-	-	1	1	1	1	6	6	6															
PF4	-	-	-	3	3	3	3	3	7															

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5													
PF2	-	2	2	2	2	4	4	4	4	4	4													
PF3	-	-	1	1	1	1	6	6	6	6	6													
PF4	-	-	-	3	3	3	3	3	7	7	7													

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3												
PF2	-	2	2	2	2	4	4	4	4	4	4	4												
PF3	-	-	1	1	1	1	6	6	6	6	6	6												
PF4	-	-	-	3	3	3	3	3	7	7	7	7												

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3											
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4											
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6											
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7											

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3										
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5										
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6										
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7										

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3								
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5								
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6								
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7								

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3	3							
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5	5							
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6	1							
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7	7							

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3				
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5				
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6	1	1	1	1				
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7				

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3			
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5			
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6	1	1	1	1	1			
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	2			

Figure: FIFO Page Replacement

Page Replacement
FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	3	
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	1	1	
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	2	2	2	

Figure: FIFO Page Replacement

Page Replacement

FIFO Simulation

- Assume we have a system with **eight logical pages** and **four physical frames** (PFs)
- Consider the following page **references in order**:
  0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 1 4
- The number of **page faults** that are generated is **13**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	3	4
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	5
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	1	1	1
PF4	-	-	-	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	2	2	2	2

Figure: FIFO Page Replacement

Recap
Take-Home Message[1]

- Translation look aside buffers to speed up access to page tables
- Inverted page tables
- Fetching policies (demand paging, pre-paging)
- Page replacement strategies

[1] Tanenbaum Section 3.3, 3.4,

Page Replacement

Exercise: FIFO vs. optimal page replacement

- Compare FIFO with **the optimal page replacement** algorithm. The process starts up with none of its pages in memory.
- What would be the minimum number of **page faults** that would be generated by the optimal approach?

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1																								
PF2																								
PF3																								
PF4																								

Figure: Optimal Page Replacement

Submit your answer at:

`https://b.socrative.com/login/student/`

Room name: G52OSC