University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 19&20: Memory Management 5&6

Edited by: Dr Qian Zhang
University of Nottingham, Ningbo China
2024

Recall

Last Lecture

- 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, multi-level, inverted, etc)

Goals for Today

Overview

- 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**
  - **How many pages** are allocated to a processes and are they **local or global**
  - **When** are pages **removed** from memory → paging daemons
- What **problems** may occur in virtual memory →thrashing

Demand Paging
On Demand

- **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** (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** (→ a similar mechanism is used when processes are **swapped out/in**)

Goals for Today

Overview

- 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**
  - **How many pages** are allocated to a processes and are they **local or global**
  - **When** are pages **removed** from memory → paging daemons
- What **problems** may occur in virtual memory →thrashing

Virtual Memory
Implementation Details

- Avoiding **unnecessary pages** and **page replacement** is important!
- Let ma, p, and pft denote the **memory access time** (10-200ns), **page fault rate**, and **page fault time**, the **page access time** is then given by:

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

Demand Paging

Performance Evaluation of Demand Paged Systems

- With an memory **access time** of 100ns (10^{-9})

(Therefore, 2 accesses->200ns) and a **page fault time** of 8ms (10^{-3})

$$(1-p)\times 200+p\times 8000000$$

- The expected access time is **proportional to page fault rate** when keeping page faults into account

Page Replacement

Concepts

- The OS must choose a **page to remove** when a new **one is loaded** (and all are occupied)
- **Objective** of replacement: the page that is removed should be the page **least likely** to be referenced in the **near future**. (reduce page fault rate)
- 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 modifed** (only modified pages need to be written)
- Replacement choices have to be **made intelligently** (<=> random) to **save time**/avoid **thrashing**

Page Replacement
Algorithms

- **Optimal** page replacement
- **FIFO** page replacement
  - Second chance replacement
- **Not recently used** (NRU)
- **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 will 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** → what would have been the minimum number of page faults
  - It provides a **lower bound** 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**
- 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

Page Replacement
Second Chance FIFO

- Second chance is a **modification of FIFO**:
  - If a page at the front of the list has **not been referenced** it is **evicted**
  - If the reference bit is set, the page is **placed at the end** of list and its **reference bit reset**

- The **(dis-)advantages** of second chance FIFO include:
  - It **works better** than standard FIFO
  - The algorithm is **relatively simple**, but it is **costly to implement** because the list is constantly changing (pages have to be added to the end of the list again)
  - The algorithm **can degrade to FIFO** if all pages were initially referenced

Page Replacement
Not Recently Used (NRU)

- **Referenced** and **modified** bits are kept in the page table
  - Referenced bits are clear at the start, and **nulled at regular intervals** (e.g. system clock interrupt)

- Four different **page "types"** exist
  - class 0: not referenced, not modified
  - class 1: not referenced, modified
  - class 2: referenced, not modified
  - class 3: referenced, modified

Page Replacement
Not Recently Used (NRU, Cont.)

- **Page table entries** are inspected upon every **page fault** →a page from the **lowest numbered non-empty class** is removed (can be implemented as a clock)

- The NRU algorithm provides a **reasonable performance** and is easy to understand and implement

Page Replacement
Least-Recently-Used

- Least recently used **evicts the page** that has **not been used the longest**
  - The OS must **keep track** of when a page was **last used**
  - Every **page table entry** contains a **field for the counter**
  - This is **not cheap** to implement as we need to maintain a **list of pages** which are **sorted** in the order in which they have been used (or search for the page)

- The algorithm can be **implemented in hardware** using a **counter** that is incremented after each instruction

Page Replacement
Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- 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 **12**

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

Figure: Least Recently Used

Page Replacement
Summary

- **Optimal** page replacement: Optimal but not realisable.
- **FIFO** page replacement: Poor performance, but easy to implement.
  - Second chance replacement: Better than FIFO, not a great implementation.
- **Not recently used** (NRU): Easy to understand, moderately efficient (Kind of an approx. of LRU).
- **Least recently used** (LRU): Good approx. to optimal. More difficult to implement (hardware may help).

Goals for Today
Overview

- 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**
  - **How many pages** are allocated to a processes and are they **local or global**
  - **When** are pages **removed** from memory → paging daemons
- What **problems** may occur in virtual memory →thrashing

Resident Set

Size of the Resident Set

- How many pages should be allocated to individual processes:
  - **Small resident sets** enable to store **more processes** in memory → improved CPU utilisation
  - **Small resident sets** may result in **more page faults**
  - **Large resident sets** (unnecessary pages) may **no longer reduce** the **page fault rate**

- A trade-off exists between the **sizes of the resident sets** and **system utilisation**

Resident Set

Size of the Resident Set

- Resident set sizes may be **fixed** or **variable** (i.e. adjusted at runtime)
- For **variable sized** resident sets, **replacement policies** can be:
  - **Local scope**: a page of the same process is replaced
  - **Global scope**: a page can be taken away from a different process

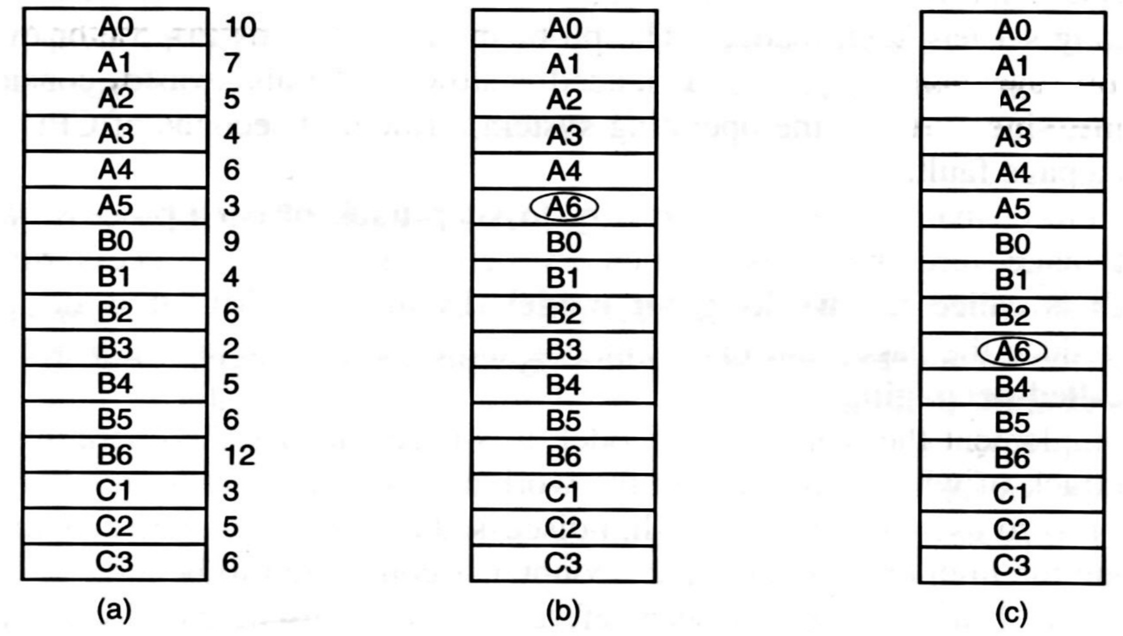


Figure: Local vs. global page replacement. (a) Original config, number at the right represents loading time (b) Local (c) Global (Tanenbaum)

Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

- Windows uses a **variable** approach with **local** replacement
- Variable sized sets require **careful evaluation of their size** when a **local scope** is used (often based on the **working set** or the **page fault frequency**)

Working Sets

Defining and Monitoring Working Sets

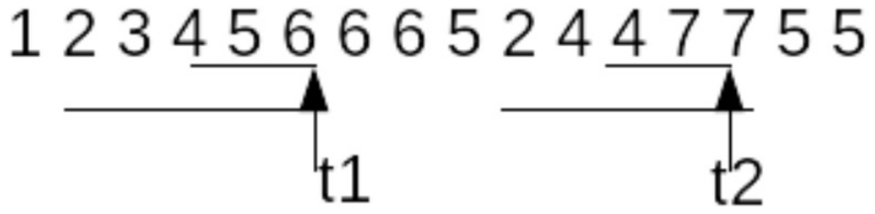
- The **resident set** comprises the set of pages of the process that are in memory

- The **working set** W(t, k) comprises the set referenced pages in the last k virtual time units for the process at time t

- The **working set size** can be used as a guide for the number frames that should be allocated to a process

Working Sets
Defining and Monitoring Working Sets

- Consider the following page **references in order**:



- If k = 3:
  - At $t1$, $W(t1,3)$ = {4,5,6}
  - At $t2$, $W(t2,3)$ = {4,7}
- If k = 5:
  - At $t1$, $W(t1,5)$ = {2,3,4,5,6}
  - At $t2$, $W(t2,5)$ = {2,4,7}

Working Sets

Defining and Monitoring Working Sets

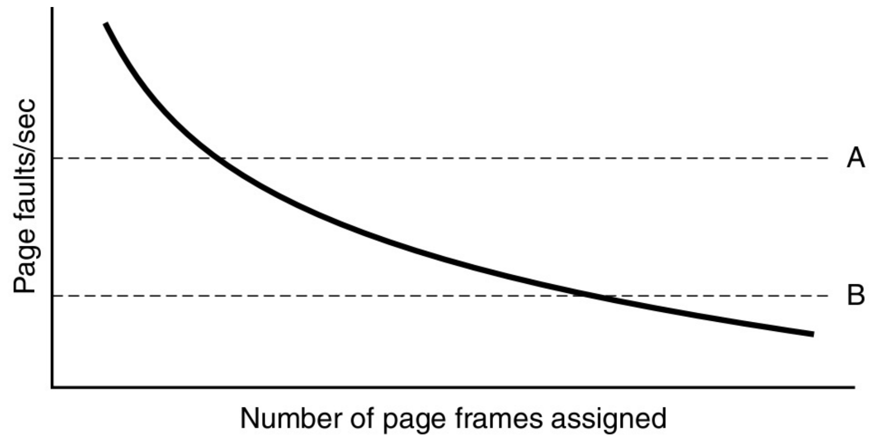
- The working set is a **function of time** t:
  - Processes **move between localities**, hence, the pages that are included in the working set **change over time**
- Choosing the right value for k is paramount:
  - Too **small**: inaccurate, pages are missing = > increase PF frequency
  - Too **large**: too many unused pages present => decrease the system usage
  - **Infinity**: all pages of the process are in the working set => decrease CPU utilisation

Working Sets

Monitoring Working Sets

- Working sets can be used to guide the **size of the resident sets**
  - Monitor the working set
  - Remove pages from the resident set that are not in the working set (LRU)

- The working set is costly to maintain → **page fault frequency** can be used as an approximation



- If the PFF is increased -> we need to increase k
- If the PFF is reduced -> we may try to decrease k

Goals for Today
Overview

- 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**
  - **How many pages** are allocated to a processes and are they **local or global**
  - **When** are pages **removed** from memory → paging daemons

- What **problems** may occur in virtual memory →thrashing

Paging Daemon

Pre-cleaning (demand-cleaning)

- It is more efficient to **proactively** keep a number of **free pages** for **future page faults**
  - If not, we may have to **find a page** to evict and we **write it to the drive** (if modified) first when a page fault occurs
- Many systems have a background process called a **paging daemon**
  - This process **runs at periodic intervals**
  - It inspect the state of the frames and, if too few pages are free, it **selects pages to evict** (using page replacement algorithms)
- Paging daemons can be combined with **buffering** (free and modified lists), i.e., write the modified pages but keep them in main memory when possible

Thrashing

Defining Thrashing

- Assume **all available pages are in active use** and a new page needs to be loaded:
  - The page that will be **evicted** will have to be **reloaded soon afterwards**, i.e., it is still active
- **Thrashing** occurs when pieces are swapped out and loaded again immediately

Thrashing
A Vicious Circle?

- CPU utilisation is too low → scheduler (medium term scheduler) **increases degree of multi-programming**
  - → Frames are allocated to new processes and **taken away from existing processes**
    - → I/O **requests are queued** up as a consequence of page faults

- CPU **utilisation drops further** → scheduler increases degree of multi-programming

Thrashing
Causes/Solutions

- **Causes** of thrashing include:
  - The degree of multi-programming is too high, i.e., the total **demand** (i.e., the sum of all **working set** sizes) **exceeds supply** (i.e. the available frames)
  - An individual process is allocated **too few pages**

- This can be **prevented** by, e.g., using good **page replacement policies**, reducing the **degree of multi-programming** (medium-term scheduler), or adding more memory

- The **page fault frequency** can be used to detect that a system is thrashing

Summary

Take-Home Message

- Fetching policies (demand paging, pre-paging)
- Page replacement strategies
  - Second Chance FIFO, NRU, LRU page replacement
- Page allocations to processes (variable, fixed, local, global)
- Page Daemons
- Thrashing

EXERCISE

Assume we have a process with 8 logical pages and a system with 4 physical frames. Considering the order in which the pages are referenced given below, demonstrate how many page faults would be generated by the least recently used page replacement algorithm?

1,0,2,2,1,7,6,7,0,1,2,0,3,0,4,5,1,5,2,4,5,6

EXERCISE

Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; 2^{20} bytes of page size; 2^{44} pages of logical address space.

i) How many bits are in a logical address?
ii) How many bytes in a frame?
iii) How many bits in the physical address specify the frame number?
iv) How many entries would you expect to find in an inverted page table?