Chapter 12

Paging an Virtual Memory Systems
1. **Virtual Memory** – giving the illusion of more physical memory than there really is (via demand paging)

2. **Pure Paging** – The total program is kept in memory as sets of (non-contiguous) pages
   1. No illusion of virtual memory

3. **Demand Paging** – A program’s “working set” is kept in memory, reference outside WS causes corresponding code to be retrieved from disk (“page fault”)
   - Provides the illusion of virtual memory
Paging Systems

1. Processes (programs) are divided into fixed size pieces called **Pages**

2. Main memory is divided into fixed size partitions called **Blocks (Page Frames)**

3. **Pure Paging** – entire program is kept in memory during execution, but pages are not kept in contiguous blocks

4. **Demand paging** – only parts of program kept in memory during execution, pages are not kept in contiguous blocks
A virtual address is represented as \(<\text{page}, \text{offset}>\). The page is a number in the range 0 – (page_size − 1), and the offset is a number in the range 0 – (page_size − 1), where the page is determined by dividing each process page into fixed size blocks (or page frames) and accommodating a process’ pages. The physical address (PA) then is \((\text{block number} \times \text{page size} + \text{offset})\). Memory is divided into fixed size blocks. In pure paging systems the entire VA space of a process must reside in physical memory during execution, but pages are not kept in contiguous blocks.
Pure Paging Virtual Addresses...

1. VA is determined from the compiled address

2. VA has two components:

<table>
<thead>
<tr>
<th>page number</th>
<th>address in page</th>
</tr>
</thead>
<tbody>
<tr>
<td>displacement</td>
<td>(or offset or)</td>
</tr>
</tbody>
</table>
Virtual Address to Physical Address Mapping

<table>
<thead>
<tr>
<th>VA Address Mapping</th>
<th>Page Frame 0</th>
<th>Page Frame 1</th>
<th>Page Frame 2</th>
<th>Page Frame 3</th>
<th>Page Frame 4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 3</td>
<td>0 − p−1</td>
<td>p − 2p−1</td>
<td>2p − 3p−1</td>
<td>3p − 4p−1</td>
<td>4p − 5p−1</td>
<td>...</td>
</tr>
<tr>
<td>Page 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program divided into Pages of size “p”
Key Idea in Paging Systems

CPU → Compiled Address

Address Translator:
Determines VA
Maps VA to PA
(Hardware + OS)

PA → Memory

Assume 256 bytes per page:

VA:

<table>
<thead>
<tr>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
</table>

8 bits

CS3204 – Arthur
Key Idea in Paging Systems...

<table>
<thead>
<tr>
<th>Page</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**VA space:**

| A[0] | 3 253 |
| A[1] | 3 254 |
| A[3] | 4 000 |

**Page Map Table:**

<table>
<thead>
<tr>
<th>Page</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**PA space:**

<table>
<thead>
<tr>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

...A[0..2]

**PA:**

<table>
<thead>
<tr>
<th>Page</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>253</td>
</tr>
<tr>
<td>3</td>
<td>253</td>
</tr>
</tbody>
</table>
Addressing Scheme

Virtual Address
\[ v = (p, d) \]

Physical Address
\[ PA = (b \times \text{page size}) + d \]
### Paging Mapping Example

#### PMT’s

<table>
<thead>
<tr>
<th>Page</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### MBT

<table>
<thead>
<tr>
<th>Page</th>
<th>Job</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>J1</td>
<td>P2</td>
</tr>
<tr>
<td>2</td>
<td>J2</td>
<td>P0</td>
</tr>
<tr>
<td>3</td>
<td>J1</td>
<td>P0</td>
</tr>
<tr>
<td>4</td>
<td>J3</td>
<td>P0</td>
</tr>
<tr>
<td>5</td>
<td>J1</td>
<td>P1</td>
</tr>
<tr>
<td>6</td>
<td>J2</td>
<td>P1</td>
</tr>
<tr>
<td>7</td>
<td>J3</td>
<td>P2</td>
</tr>
<tr>
<td>8</td>
<td>J3</td>
<td>P1</td>
</tr>
</tbody>
</table>

#### PMTAR

- **PMT addr of current Job**

---

**CS3204 – Arthur**
**Page Map Table (PMT):**

Contains VA page to PA block mapping

<table>
<thead>
<tr>
<th>Page</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>

1 PMT / job
1 Entry / page
Page Management

**Page Map Table Address Register (PMTAR):**

| Length of program in pages (# PMT entries) | Base address of current PMT |

Points to PMT for **currently executing job**

1 PMTAR / System
Memory Block Table (MBT)

Maps each block of main memory either to a process id and page number or to "free"

1 MBT / System
1 Entry / Block
Page Management

Process (Job) Control Block (PCB) (PCB)

Contains information about all jobs in the system

1 PCB / system

1 entry / job

Job Size

Location of PMT

Stores: Job Size
Page Addressing – Let’s get REAL

VA = < page, offset >

PA = block size * block + offset

Assume:

1 word PMT entries;

byte addressable MM

Example:

page & block size = 4 K bytes

VA = < 1, 1234 >

PA = 4096 * 16 + 1234
Determining Virtual Address <Page, Offset> from the Compiled Address

Compiled Address (relative to 0) : 18087

Page size: 2K (2048 bytes)
Memory is byte addressable

Virtual Address:
Page = Div (Compiled Address, Page Size)
Offset = MOD (Compiled Address, Page Size)

<8, 1703>

DIV => “Shift right”, 11 bits \((2048 = 2^{11})\)
MOD => “AND” 21 High–order bits with 0

CS3204 – Arthur
Assume:

2 bytes PMT entries; byte addressable MM
   
   page & block size = 4 K bytes

1) What is the maximum size for any program?

2) What PA corresponds to compiled address 220945?

2) What is the MBT length if MM size is 80M?
   (Assume MBT entries are 2 bytes long.)

3) What is the PMT length if compiled size = 300K?
procedure allocation (int Size) {
    NPpgm := ceiling( Size / P);
    NPmt := ceiling( (NPpgm * WS) / P );
    NPTot := NPpgm + NPmt;
    If ( NPTot > MaxBlocks )
        Then ERROR
    Else If ( NPTot blocks are not free in MBT )
        Then Add job to HOLDQ;
    Else {
        Allocate pages to blocks;
        Update MBT, PCB;
        Create, initialize PMT;
    }
}
Sharing Pages of Reentrant Code or Data Between Processes

Fig. 8.15 Sharing in a pure paging system.
Pros/Cons of Paging

😊 **Advantages:**

- Efficient memory usage
- Simple partition management due to discontiguous loading and fixed partition size
- No compaction necessary
- Easy to share pages
Disadvantages:

- Job Size <= Memory Size
- Internal fragmentation (half the page size on the average)
- Need special hardware for address translation
- Some main memory space used for PMT’s
- Address translation lengthens memory cycle times
Demand Paging

Jobs are paged, but not all pages have to be in memory at the same time.

VIRTUAL MEMORY

- The operating system creates the illusion of more memory
- Job size can exceed main memory size

- Pages are only brought in when referenced (on demand)
- Often page 0 is loaded initially when a job is scheduled
Demand Paging Motivation

1. What happens if job 3 references page 1?
2. What does the CPU do while J3P1 is being read?
Termiology

**Page fault:**

Interrupt that arises upon a reference to a page that is not in main memory.

**Page swapping:**

Replacement of one page in memory by another page in response to a page fault.
When a Page Fault Occurs

- Select a page to be removed
- Copy it to disk if it has been changed **
- Update old page’s PMT **
- Copy new page to main memory
- Update new page’s PMT
- Update MBT **

Thrashing occurs when a job continuously references pages which are not in main memory
Demand Page Management

**Page Map Table (PMT)**
- Maps page to blocks
- Status: Pointer to
  - Main Memory Block
  - `Indicator` Main/Secondary Memory

**Memory Block Table (MBT)**
- Maps block to page
- Contains: Job/Page Number
  - `Reference bit`
  - `Change bit`

**File Map Table (FMT)**
- Maps a job’s pages to secondary memory
- PMT for the Disk
- 1 FMT / job
- 1 entry / page
Demand Paging Schematic

PMTAR

PMT

FMT

DISK

Main Memory

MBT

OS

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

R C

J1 P1

J1 P2

CS3204 – Arthur
1. What happens if job 3 references page 1?
Summary of Data Structures

1) **Page Map Table (PMT):** Maps page to block

*Fields:*  
- page number (which page in memory)  
  - In_Memory  

2) **Memory Block Table (MBT):** Maps block to either process id and page number or to "free"

*Fields:*  
- Reference Count  
- Change Bit

3) **File Map Table (FMT):** Maps a job’s pages to secondary memory (like a PMT for the disk)

*Fields:*  
- In_Memory  

1 FMT / job, 1 entry / page
Local versus Global Page Replacement

A replacement page may be selected from the set of all blocks.

Local

Requires that each process remove a page from its own set of allocated blocks.

Global

Now we consider the decision of selecting which page to replace upon a page fault.
A Program’s Execution Profile

Question:

Does a program need all its pages in main memory at all times?
The Principle of Locality

At any time, the *locality* of a process is the set of pages that are actively being used together

**Spatial** There is a high probability that once a location is referenced, the one after it will be accessed in the near future

Sequential code, Array processing, Code within a loop

**Temporal** A referenced location is likely to be accessed again in the near future

Loop indices, Single data elements
More on Locality

Does a linked list help or hurt locality?

Does a recursive function display spatial or temporal locality?
1. A process’ **working set** is the number of pages currently being referenced during $(t, t+\Delta)$ for some small $\Delta$.

2. The working set size is an estimate of degree of locality.

3. A job should not be scheduled unless there is room for its entire working set.
   1. Why?
Idea Behind Working Set

Fig. 9.6 Primary storage allocation under working set storage management.
Motivation: Page Replacement Algorithms

Which page replacement rule should we use to give the \textit{minimum} page fault rate?

Page fault rate = \# faults / \#refs
Page Replacement Algorithm: Optimal Replacement

- Replace the page which will not be used for the longest period of time
- Lowest page fault rate of all algorithms
- Requires knowledge of the future

**Example:**

MM has 3 blocks containing 3,5,2.
Current and future refs:

4, 3, 3, 4, 2, 3, 4, 5, 1, 3, 4

Fault

OPT replaces 5
## Optimal Replacement Algorithm

<table>
<thead>
<tr>
<th>Page Trace:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Number</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

Page Fault Rate =

CS3204 – Arthur
Replacement Algorithm: FIFO

- Replace the "oldest" page
- A frequently used page may be swapped out

*Belady’s Anomaly:*

For some page replacement algorithms, the page fault rate may increase as the number of blocks increase.
### FIFO Page Replacement

**Page Trace:**

<table>
<thead>
<tr>
<th>Block Number</th>
<th>Page Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 0 1 4 0 1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

**Page Trace:**

<table>
<thead>
<tr>
<th>Block Number</th>
<th>Page Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 0 1 4 0 1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

---

CS3204 – Arthur
Replacement Algorithms: Least Recently Used (LRU)

- Uses the recent past as an approximation of the near future

- Stack algorithm
  - Does NOT suffer from Belady’s Anomaly

- Hardware / Overhead intensive
**Least Recently Used (LRU)**

<table>
<thead>
<tr>
<th>Block Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Trace:</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

<table>
<thead>
<tr>
<th>Page Trace:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Number</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

CS3204 – Arthur
Replacement Algorithms: LRU Approximation

- Uses reference bits in the MBT and a static reference pointer (RP)
- The reference pointer is not reinitialized between calls to LRU Approximation
- Set referenced bit to 1 when loading a page
- Set referenced bit to 1 on a R/W
- Set referenced bit to 0 if currently a 1 and scanning for a replacement page
- Replace page with reference bit = 0
LRU Approximation Algorithm...

Initially: \(RP \leftarrow -1\)

begin
\(RP := (RP + 1) \mod MBTSize;\)
While \((MBT[RP].Referenced = 1)\) Do
Begin
\(MBT[RP].Referenced := 0\)
\(RP := (RP + 1) \mod MBTSize;\)
End
return\((RP);\)

Note: referenced bit is set to 1 when a page is
(a) referenced, and
(b) when first loaded into memory
RP always points to last page replaced

CS3204 – Arthur
LRU Approximation

<table>
<thead>
<tr>
<th>Block Number</th>
<th>Page Trace:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 0 1 4 0 1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =

Page Fault Rate =
Replacement Algorithms:
Least Frequently Used (LFU)

- Keep a reference count, select page with lowest count
- Reference count is number of times a page has been referenced over its current stay in memory, not over the lifetime of the program

Page Trace:

<table>
<thead>
<tr>
<th>Block Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Page Faults =
Pros/Cons of Demand Paging

Advantages:

1. Can run program larger than physical memory
2. Allows higher multiprogramming level than pure paging
3. Efficient memory usage
4. No compaction is required
5. Portions of process that are never called are never loaded
6. Simple partition management due to discontinuous loading and fixed partition size
7. Easy to share pages
Disadvantages:

1. Internal fragmentation
2. Program turnaround time increases each time a page is replaced, then reloaded
3. Need special address translation hardware