Memory Management

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

Memory Management

"The memory management on the PowerPC can be used to frighten small children.''

– Linus Tovalds

Memory Management

"The memory management on the PowerPC can be used to frighten small children."

– Linus Tovalds

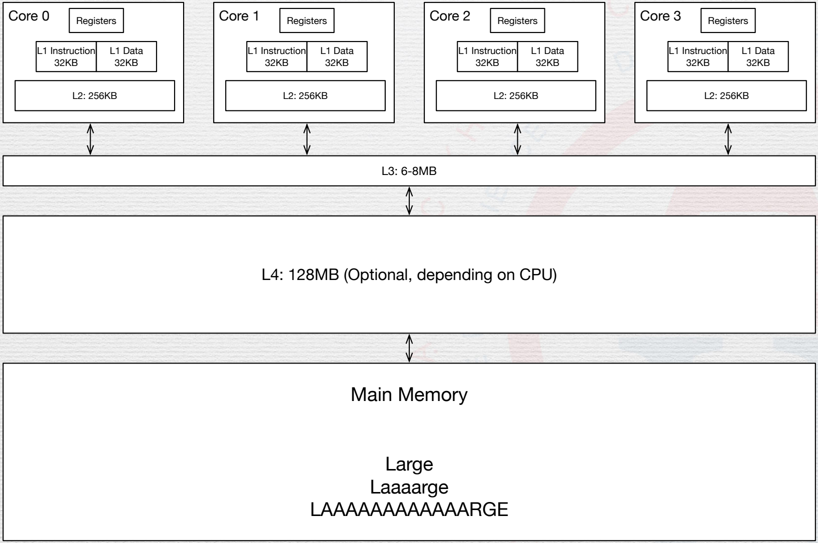
"640 K ought to be enough for anybody."

– Rumor attributed to Bill Gates, 1981

Contents

Physical Memory

Registers, Cache and Memory



Registers, Cache and Memory

- Lower level, smaller size
- Lower level, closer to ALUs
- Lower level, faster speed

Computer Architecture

- Execution
  - **Code inside memory**
  - CPU fetches instruction from memory at program counter
  - CPU decodes instruction
  - CPU executes instruction
  - CPU may save result back to memory

Physical Memory

Table 1: Characteristics of different DDR memory[1]

Names	Memory clock	I/O bus clock	Transfer rate	Max bandwidth
SDR-100 [2]	100 MHz	100 MHz	0.1 GT/s	0.8 GB/s
DDR-200	100 MHz	100 MHz	0.2 GT/s	1.6 GB/s
DDR2-800	200 MHz	400 MHz	0.8 GT/s	6.4 GB/s
DDR3-1600	200 MHz	800 MHz	1.6 GT/s	12.8 GB/s
DDR4-3200	400 MHz	1600 MHz	3.2 GT/s	25.6 GB/s

[1] Source: wikipedia and Transend-Info

[2] To be used with Intel Pentium

Physical Memory

- Similar to a **HUGE** array, shared by everything:
  - Kernel
  - All loaded kernel modules
  - All drivers
  - **All processes**

→ Memory management goal: how to effectively manage physical memory?

Physical Memory

- Effectively manage physical memory?
  - Share physical memory among OS and processes
  - Isolation: process A cannot access process B's memory
  - Dynamic allocation: process A should be able to ask for more memory at runtime
    - That's heap

Addressing

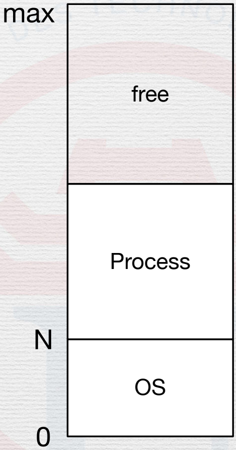
Memory Management Goal

- Effectively manage physical memory?
  - **Share physical memory among OS and processes**
  - Isolation: process A cannot access process B's memory
  - Dynamic allocation: process A should be able to ask for more memory at runtime

Addressing: Naive solution

- Example: simplest case
- Monoprogramming
  - One OS
  - One process

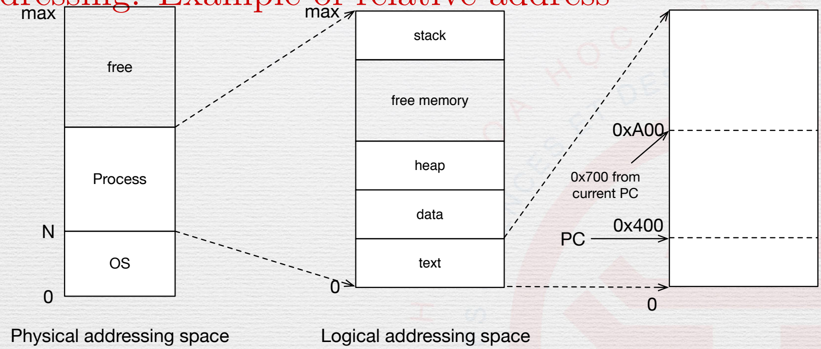
```
#include <stdio.h>

void main() {
    printf("Hello, world!\n");
}
```



Addressing: What?

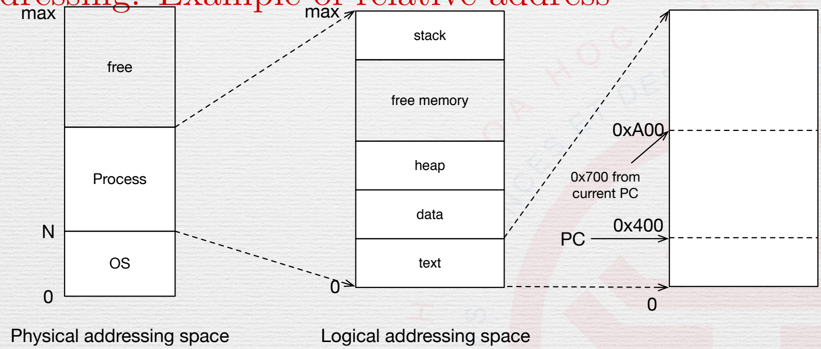
- Address in addressing space
  - **Absolute** address: the exact address in the addressing space
    - E.g. 0x401000 in the logical address space
    - E.g. 0xFF000000 in the physical address space
  - **Relative** address: use another address as a pivot
    - E.g. 0x700 bytes from the current program counter of process memory in the logical address space

Addressing: Example of relative address



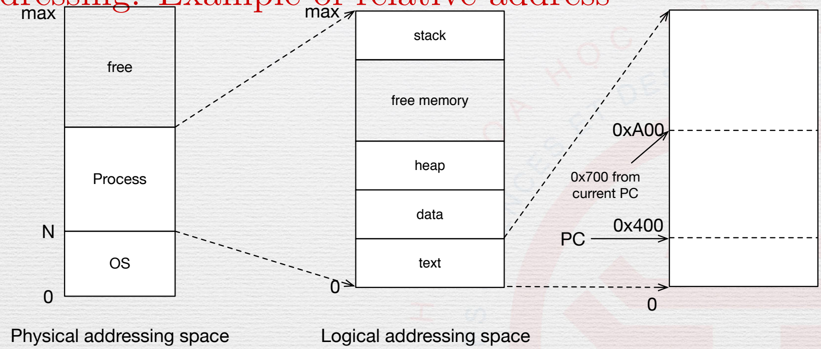
- 0x700 from current PC in the logical address space

Addressing: Example of relative address



- 0x700 from current PC in the logical address space
- Question: What is the absolute address of the above address in the physical addressing space?

Addressing: Example of relative address

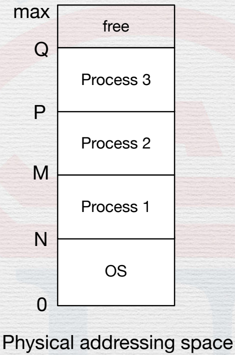


- 0x700 from current PC in the logical address space
- Question: What is the absolute address of the above address in the physical addressing space?
  - `N + 0x400 + 0x700`

Direct Mapping

Direct Mapping: What?

- Multiprogramming
  - One OS
  - Many processes
- Naive solution: put process memory next to each other in physical memory

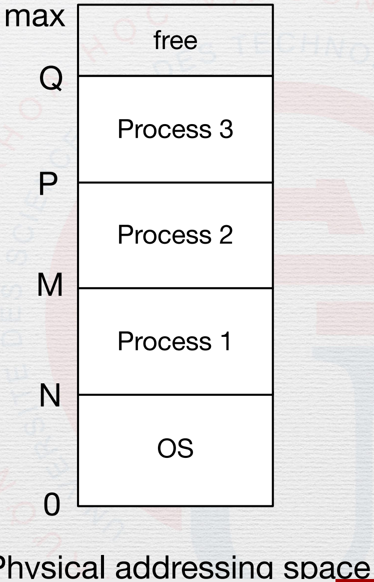


Direct Mapping: Why?

- Supports multiple processes
- The easiest way to share physical memory among OS and proceses

Direct Mapping: Problems

- Protection (Isolation)
- Fragmentation
- Dynamic allocation



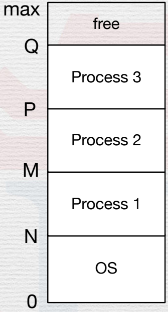
Direct Mapping: Protection

What?

- OS **protects** its memory and processes memory from illegal access

Why?

- OS memory should not be accessed by processes
  - System-wide crashes...
- Process A's memory should not be accessed by other processes
  - Privacy and security



Physical addressing space

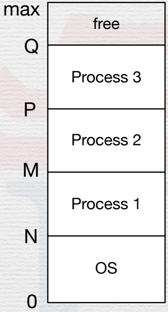
Direct Mapping: Protection

How to solve?

- Use a **limit** register (l) to specify memory range for each process
- Check all memory accesses of process i against l_i
  - Outside range: crash process
- OS is unrestricted: Why?

E.g.

- $l_1 = M - N$
- $l_2 = P - M$
- $l_3 = Q - P$

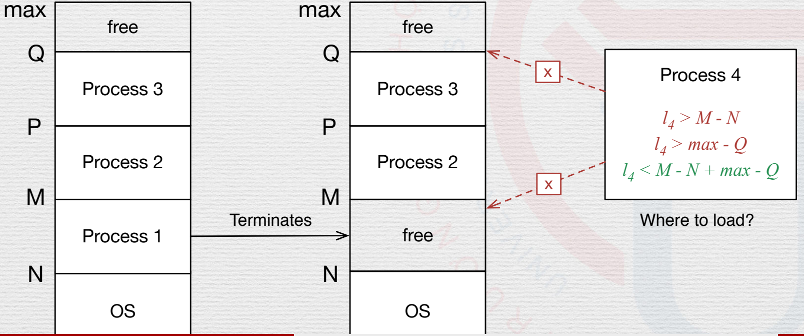


Physical addressing space

Direct Mapping: Fragmentation

What?

- Many small pieces of free memory
    - Process creations
    - Process terminations



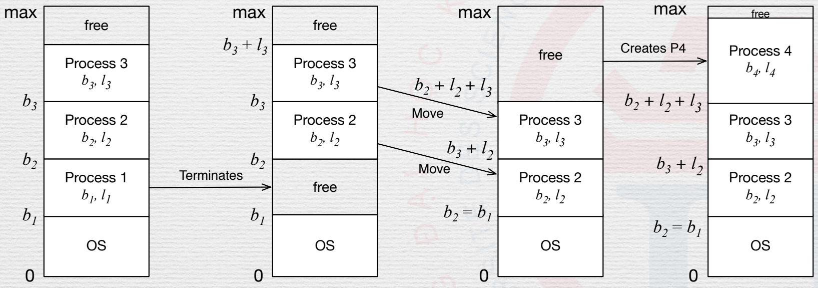
Direct Mapping: Fragmentation

How to solve?

- Use a **base** register b for each process
    - Starting position in memory (base)
    - Use relative addresses like described in "Addressing'', using b_i
- «Compact» memory regions
    - Make a large free memory block
- Create a new process in this free block

Direct Mapping: Fragmentation

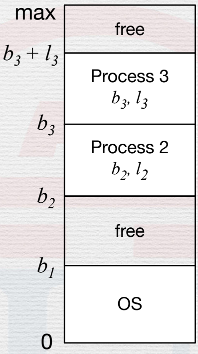
How?



Physical addressing space

Direct Mapping: Checkpoint

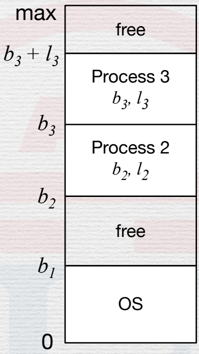
- What?
  - Put process memory next to each other in physical memory
- Why?
  - Naive



Direct Mapping: Checkpoint

Problems:

- **Protection (Isolation)**
  - Protects illegal memory access
  - Use **limit** register l
- **Fragmentation**
  - Free memory becomes little pieces
    - Do not fit new processes → Waste
    - Also called «holes»
  - Add **base** register b with relative address
  - «Compact» memory regions
    - Make a large free memory block
- Dynamic allocation
  - To be continued...



Direct Mapping: Dynamic Alloc

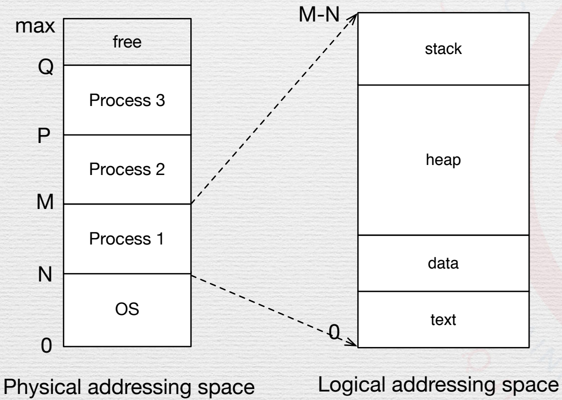
What?

- With fixed l: how does process heap grow?
- Remind: what's heap?

Why?

- Dynamic allocation is everywhere...
  - Objects (`new Student()`)
  - Buffers (`malloc()`)
  - Linked lists

Direct Mapping: Dynamic Alloc



Direct Mapping: Dynamic Alloc

How to solve?

- Solution 1: Limit amount of memory for each process
    - Consequence 1: fixed heap
        - How will games work?
    - Consequence 2: limit number of concurrent processes at any time
- Solution 2: virtual memory

Virtual Memory

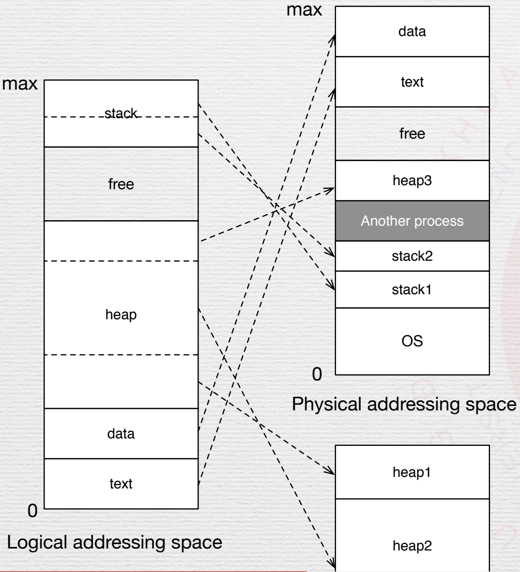
Virtual Memory: What?

- Per-process memory that is seen by processes
  - Can be larger than physical memory
  - Can be backed by different types of memory
  - Separation of process memory from physical memory
- Allow non-continuous addresses in physical address space
  - Complicated mapping methods than simple b and l registers
  - $\rightarrow$ Support dynamic allocation

$\rightarrow$ Program execution behaves identically every time the program is executed

- Even memory organization changes
- Even on different machine

Virtual Memory: What?



Virtual Memory Analogy: Exam Checklist!

Virtual memory ~ exam checklist

- Need : place for students
- Don't care : where students are actually placed
  - Same order?
  - Enough place?
  - Students in corridor?

UNIVERSITÉ DES SCIENCES ET DES TECHNOLOGIES DE HANO
UNIVERSITY OF SCIENCE AND TECHONLOGY OF HANOI
TRƯỜNG ĐẠI HỌC KHOA HỌC VÀ CÔNG NGHỆ HÀ NỘI

EXAMINATION ATTENDANCE CHECK LIST

Academic year: 2016-2017

Subject: Operating Systems — Exam. Venue: — No. of presence:
Subject code: — Exam. Date: — No. of absence:
Specialty: ICT — Bachelor: 2

No.	Surname	Last name	Student ID	Signatures	Note
1	John Philip	Anderson	USTHBI99-001		
2	Carmen	Bok	USTHBI99-002		
3	Ann Marie	Chang	USTHBI99-003		
4	Alberta	Ethan	USTHBI99-004		
5	Darwin	Holsworthy	USTHBI99-005		
6	Josephine S.	Jefferson	USTHBI99-006		
7	Curtis Jerome	McDonald	USTHBI99-007		
8	Matthew Donaldson	Truman	USTHBI99-008		
9	Alan	Bloomberg	USTHBI99-009		
10	Stephen	Bruno	USTHBI99-010		
11	Richard T.	Calvin	USTHBI99-011		
12	Kevin	Charles-Smith	USTHBI99-012		
13	Michael J.	DiAngelo	USTHBI99-013		
14	Florinda	Fernando Riviera	USTHBI99-014		
15	Jeffery Thomas	Gomez	USTHBI99-015		
16	Simon	Kelley	USTHBI99-016		
17	Samuel	Lee	USTHBI99-017		
18	Delores May	Luna	USTHBI99-018		
19	Steven Reginald	Michaels	USTHBI99-019		
20	Luis Raymond	Nathan	USTHBI99-020		
21	Marjorie	Otto	USTHBI99-021		
22	Rachael	Perez	USTHBI99-022		
23	Stephanie Michelle	Richardson	USTHBI99-023		
24	Emmanuel Joseph	Sanchez	USTHBI99-024		
25	Joseph Dean	Stowe	USTHBI99-025		
26	Stuart	Tobey	USTHBI99-026		

Assistant *(Signature & full name)* — **Invigilator** *(Signature & full name)*

Virtual Memory: Why?

- Dynamic allocation
  - More processes concurrency
  - No limit on amount of memory allocation
- Shared memory between processes
- Efficient process creation

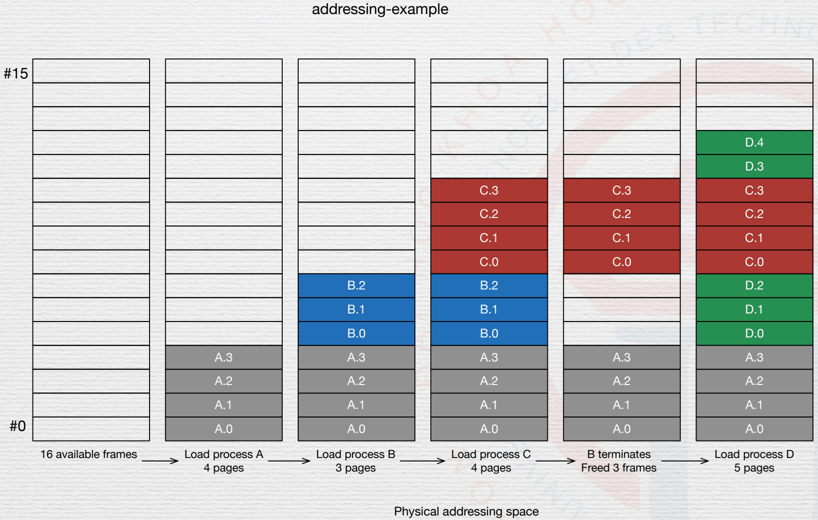
Virtual Memory: How?

- Paging
- Swapping
- Segmentation

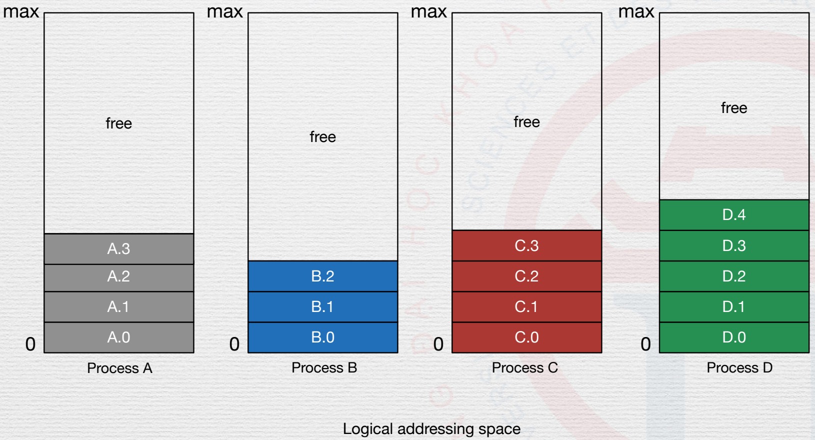
Paging: What?

- Divide physical memory into small fixed-size chunks (**frames**)
- Divide virtual memory into small fixed-size chunks (**pages**)
- Frame size == page size
- Assign a virtual memory page to a physical memory frame
- Popular size: 4KB
  - Too small page: too many pages
  - Too big page: waste memory (fragmentation)

Paging: Example



Paging: Example



Paging: Why?

- Allow dynamic allocation
- Allow process memory to be mapped to non-continous physical memory
- Easy «compactation»

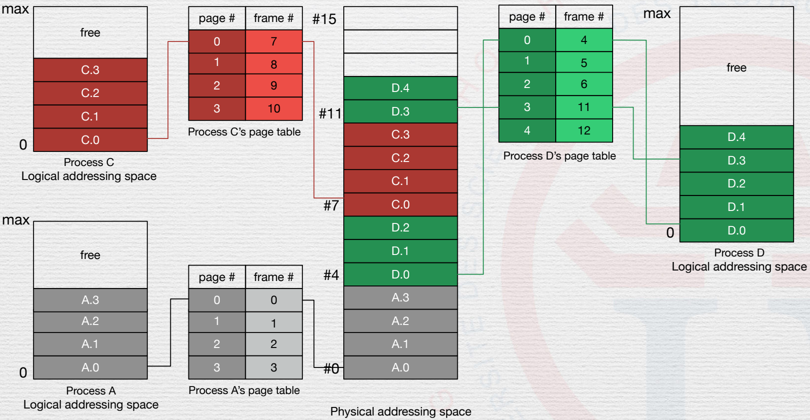
Paging: How?

- **Page table**
  - Map page to frame
- **Address Translation**
  - Virtual address → Physical address
  - Using page table

Paging: How - Page Table

- In-memory, per-process, 2-column table
  - Can (and should) be simplified to 1 column
- **Map** page π_j^i of process p_i to frame f_k in physical memory
  - Similar to a «lookup» table
- Accessible only by the OS
- Saved / loaded in context switch
- Can be multi-level

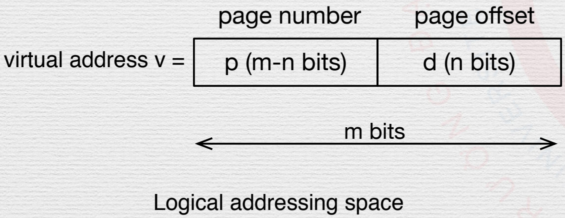
Paging: How - Page Table



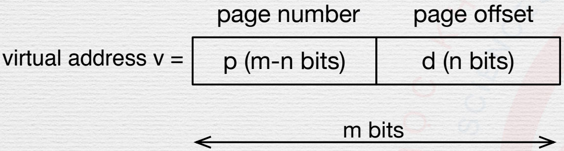
Only to map page, not exact address

Paging: How - Address Translation

- Split virtual address to 2 parts
  - page number: page id in the page table
  - page offset: distance (how far) from the beginning of this page ($0 \rightarrow$ `pageSize - 1`)
    - i.e. **where** in this page
- Map exact virtual address to exact physical address



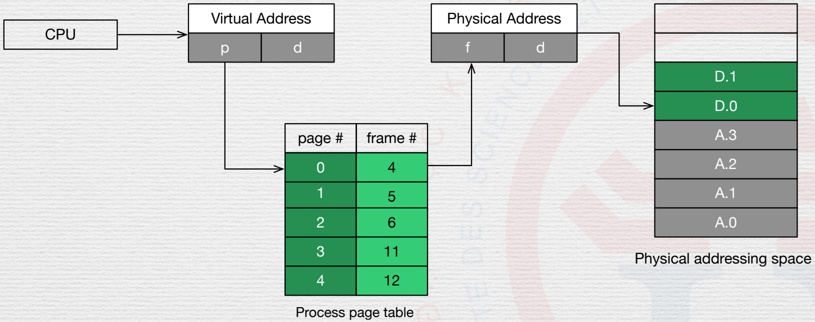
Paging: How - Address Translation



Logical addressing space

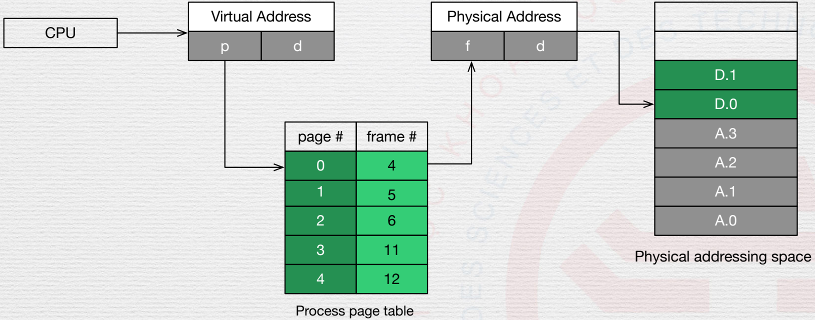
Physical address $\rho = \texttt{pageTable}[p] \times 2^n + d$

Paging: How - Address Translation



Physical address $\rho = \texttt{pageTable}[p] \times 2^n + d$

Paging: How - Address Translation Example

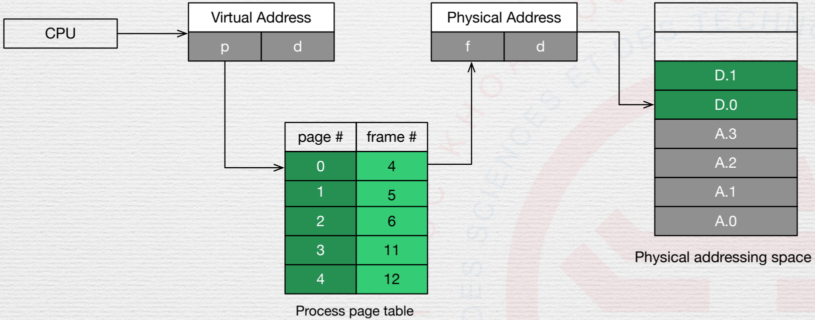


Virtual address v = 0x24$_{(16)} = 011000_{(2)}$

Virtual address size $= 64\ (= 2^6 \rightarrow m = 6)$

Page size $= 16\ (= 2^4 \rightarrow n = 4)$

Paging: How - Address Translation Example



$$v = 011000_{(2)}, m = 6, n = 4$$

- $\rightarrow d$ has 4 bits
- $\rightarrow p$ has $6 - 4 = 2$ bits

Paging: How - Address Translation Example

$$v = 011000_{(2)}$$

page number	page offset
01 (2 bits)	1000 (4 bits)

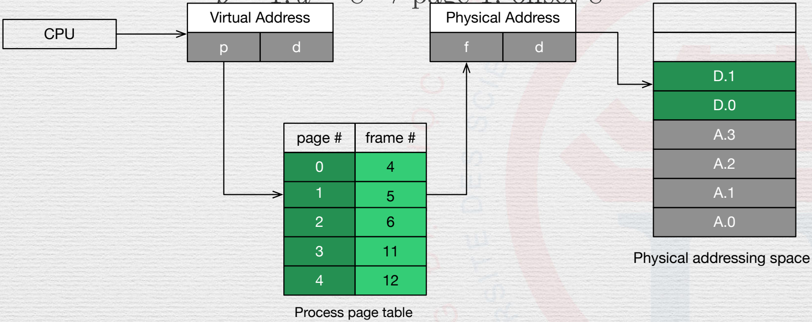
6 bits

$$p = 01_{(2)} = 1_{(10)}, d = 1000_{(2)} = 8_{(10)}$$

Paging: How - Address Translation Example

$$v = 011000_{(2)}$$

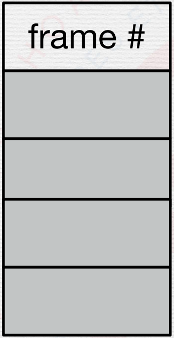
$$p = 1, d = 8 \rightarrow \text{page 1, offset 8}$$



Physical address $\rho = \texttt{pageTable[1]} \times 2^4 + 8 = 5 \times 16 + 8 = 88$

Paging: Page Table in Reality

- No column "page #' '
- One column only
  - Frame id
- Save memory



Paging: Advantages

- Solves fragmentation
  - Any free frame can be allocated to a process
- Solves dynamic allocation
  - Process can request as many pages as it needs

Paging: Problems

- Large
- Performance Inefficient
- Virtual memory must be smaller than physical memory

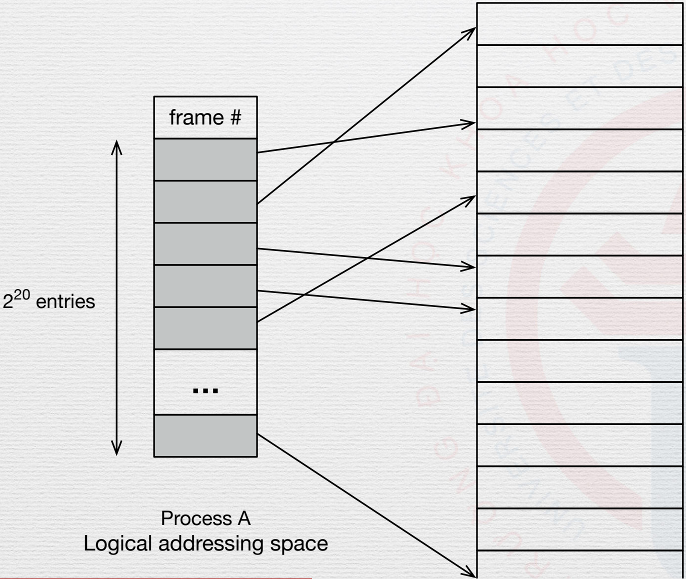
Paging: Problem 1 - Large

- E.g. 32bit OS: 4GB for virtual memory space
- 4KB page size ($n = 12$, or d is 12-bit)
- Each page table entry is 32-bit
- Question: How big is the page table for **each** process?

Paging: Problem 1 - Large

- E.g. 32bit OS: 4GB for virtual memory space
- 4KB page size ($n = 12$, or d is 12-bit)
- Each page table entry is 32-bit
- Question: How big is the page table for **each** process?
  - Number of entries: 4GB / 4KB $= \frac{4\times2^{30}}{4\times2^{10}} = 2^{20} = 1048576$
  - Size = number of entries $\times$ size per entry
    $= 1048576 \times 4 = 4194304$ bytes $=$ 4MB

Paging: Problem 1 - Large



Paging: Problem 1 - Large

On my own MacBook Pro and Hackintosh systems (which are used for writing this slide)

```
$ ps aux | wc -l
320
$ echo "`ps aux | wc -l` * 4194304" | bc
1337982976
```

Paging: Problem 1 - Large

- Exercise: how large is page table for 64bit OS?
  - Addressing only 48 bits (256TB addressable)
  - 4KB page size
  - Each page table entry is 32 bit
  - Question: How big is the page table for **each** process?

Paging: Problem 1 - Large

Solution: multi-level page tables

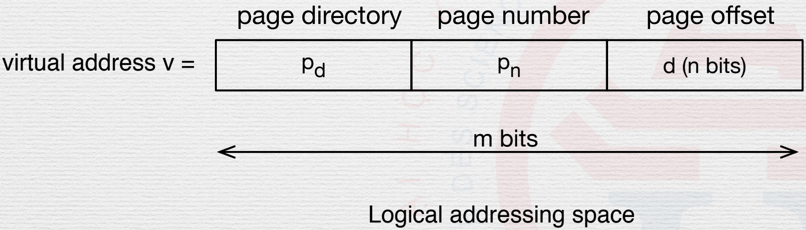
- Split page number to several parts
  - Page directory
  - Page number
- Similar to tree structure

Paging: Problem 1 - Large

Solution: multi-level page tables

- Split page number to several parts
  - Page directory
  - Page number
- Similar to tree structure
- Page table is paged ☺

Paging: Problem 1 - Large



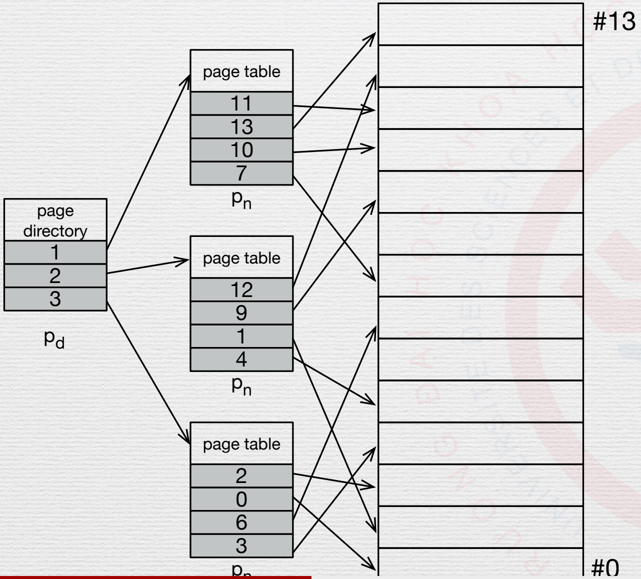
0

Paging: Problem 1 - Large

- Example for 32-bit OS
  - 32-bit logical address
  - Single level
    - 20-bit page number
    - 12-bit page offset (page size $= 2^{12} = 4$KB)
  - Two levels
    - 10-bit page directory
    - 10-bit page number
    - 12-bit page offset

0

Paging: Problem 1 - Large

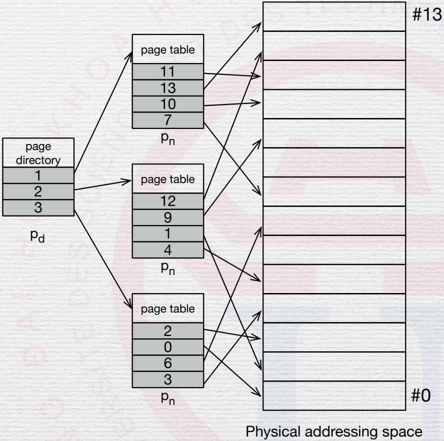


Paging: Problem 1 - Large

Exercise:

- 2-bit page directory
- 4-bit page number
- 12-bit page offset

What is physical address of
$v = 100001110001011100_{(2)}$



Paging: Problem 1 - Large

- Linux
  - 2.6.9 and below: 3 levels
    - Page Global Directory
    - Page Middle Directory
    - Page Table Entries
    - 512GB addressable
  - 2.6.10 (Jan 2005): 4 levels
    - Page Global Directory
    - Page Upper Drectory
    - Page Middle Directory
    - Page Table Entries

Paging: Problem 1 - Large

- Windows
  - 32bit: 2 levels
    - Page Directory (10bit)
    - Page Table (10bit)
    - 2GB addressable
  - 64bit: 4 levels[3]
    - Page Map Level 4 (9bit)
    - Page Directory Pointer Table (9bit)
    - Page Directory (9bit)
    - Page Table (9bit)
    - Before Windows 8.1: 8TB

Paging: Problem 1 - Large

- Why is multi-level page table better than single level (flat) one?

Paging: Problem 1 - Large

- In case processes with very few memory pages:
  - Flat page table is sparse
    - Only few elements are used
    - Always 4MB per page table (20-bit page number + 12-bit page offset)
  - Multi-level table only uses smallest number of page tables, best case:
    - 1 page directory
    - 1 page table

Paging: Problem 2 - Performance Inefficient

- One single memory location access requires 2 memory accesses
  - 1 for accessing page table
  - 1 for the data itself

Paging: Problem 2 - Performance Inefficient

- One single memory location access requires 2 memory accesses
  - 1 for accessing page table
  - 1 for the data itself
- Question: in Linux x64, how many memory accesses for this C instruction?

```
i++;
```

Paging: Problem 2 - Performance Inefficient

```
i++;
```

- Answer: 10 physical memory accesses

Paging: Problem 2 - Performance Inefficient

```
i++;
```

- Answer: 10 physical memory accesses
    - Reading content of `i` to register

Paging: Problem 2 - Performance Inefficient

```
i++;
```

- Answer: 10 physical memory accesses
  - Reading content of `i` to register
    - 1 / 1 / 1 / 1 / 1 for accessing PGD / PUD / PMD / PTE / Content
  - Writing content to `i` after increasing register by 1

Paging: Problem 2 - Performance Inefficient

```
i++;
```

- Answer: 10 physical memory accesses
  - Reading content of `i` to register
    - 1 / 1 / 1 / 1 / 1 for accessing PGD / PUD / PMD / PTE / Content
  - Writing content to `i` after increasing register by 1
    - 1 / 1 / 1 / 1 / 1 for accessing PGD / PUD / PMD / PTE / Content

Paging: Problem 2 - Performance Inefficient

Solution:

- «Cache-like» memory for page tables
- Translation Lookaside Buffer (TLB)
- On-chip, single level (flat)
  - Partially parallel
  - Fully parallel
  - Fast!
- Two columns
  - Page id
  - Frame id
- Can be multilevel as well!

Paging: Problem 2 - Performance Inefficient

Fast, but small [4]

Code name	Generation	TLB L1 entries	TLB L2 entries
Yorkfield	Core 2 Quad	16	256
Nehalem	1st gen, e.g. i7 920	64	512
Ivy Bridge	3rd gen, e.g. i7 3632QM	64	512
Haswell	4th gen, e.g. i7 4770HQ	64	1024
Skylake	6th gen, e.g. i7 6700K	64	1536
Kaby Lake	7th gen, e.g. i7 7700K	64	1536

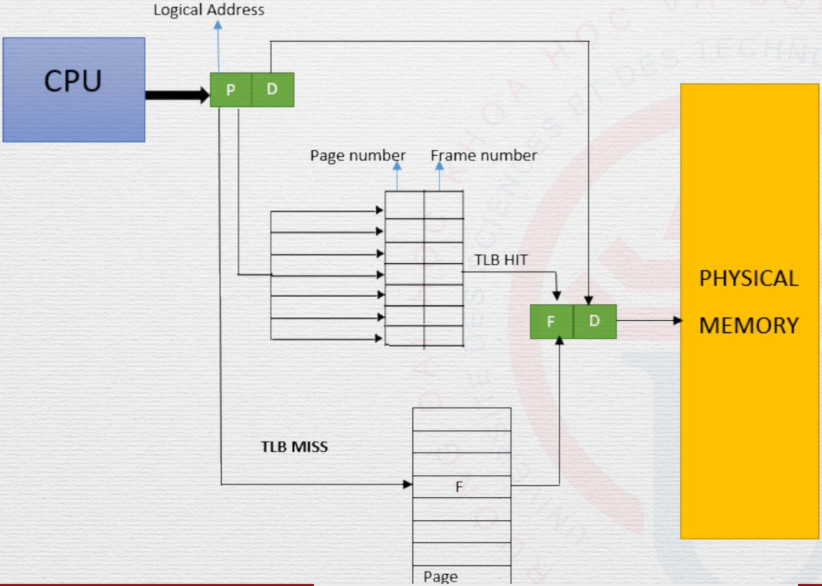
[4] Data courtesy of 7-CPU

Paging: Problem 2 - Performance Inefficient

Solution:

- On **every** memory access
  - Check page id in TLB
  - If exists, use frame id («TLB hit») without using page tables
  - Else, do full (possible multi-level) page table lookups («TLB miss»)

Paging: Problem 2 - Performance Inefficient



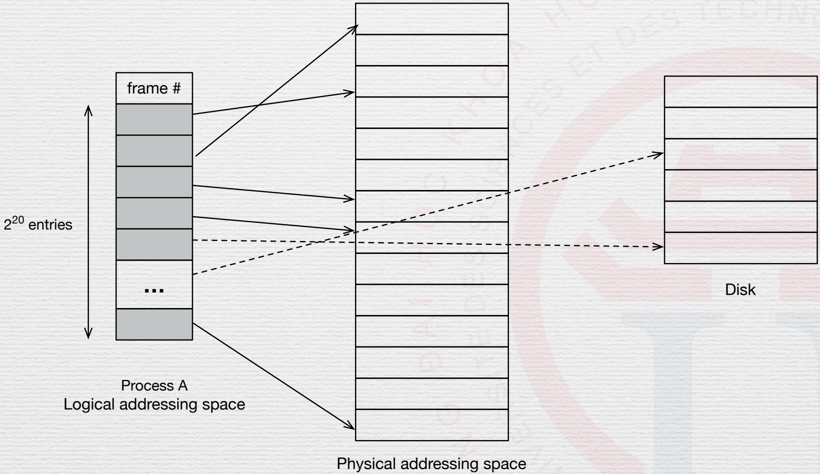
Paging: Problem 3 - Small Virtual Memory

Remind:

- Virtual Memory > Physical Memory
- Code / data must be in physical memory

→ temporarily use disk as storage for unused/least used pages

Paging: Problem 3 - Small Virtual Memory



Paging: Problem 3 - Small Virtual Memory

- When to swap to disk?
  - Low on memory
  - Very few access on a page

Paging: Problem 3 - Small Virtual Memory

- When to swap to disk?
  - Low on memory
  - Very few access on a page
- When to swap back?
  - Access to a swapped page
  - «Page fault»

Memory Management

"The memory management on the PowerPC can be used to frighten small children."

– Linus Tovalds

Memory Management

"The memory management on the PowerPC can be used to frighten small children."

– Linus Tovalds

"640 K ought to be enough for anybody."

– Rumor attributed to Bill Gates, 1981