

# Main Memory

Hui Chen <sup>a</sup>

<sup>a</sup>CUNY Brooklyn College

April 22, 2020

# Outline

- 1 Concept of Address Translation
- 2 Contiguous Memory Allocation
- 3 Paging
  - Concept and Basic Scheme
  - Fragmentation
  - Protection and Sharing
- 4 Implementing Paging
  - Access Latency Too High?
  - Page Table Too Large?
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Table

# Outline

- 1 Concept of Address Translation
- 2 Contiguous Memory Allocation
- 3 Paging
  - Concept and Basic Scheme
  - Fragmentation
  - Protection and Sharing
- 4 Implementing Paging
  - Access Latency Too High?
  - Page Table Too Large?
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Table

# MMU and Address Translation

- ▶ Policy. Each process has a separate memory space.
  - ▶ Protection. Keep each process isolated.
  - ▶ Sharing. Allow memory to be shared between processes.
  - ▶ Virtualization. Provide applications with the illusion of “infinite” memory.
- ▶ Mechanism. Mediating memory access via a hardware component called the memory management unit (MMU). MMU that translate a logical address to a physical address.
  - ▶ Logical address (or virtual address). An address generated by the CPU.
  - ▶ Physical address. An address generated by the MMU

## Base-Limit Registers

Introduce a pair of registers,

- ▶ Base register. It holds the smallest legal physical memory address.
- ▶ Limit register. It specifies the size of the range.

We can

- ▶ allow processes loaded in memory for concurrent execution, and
- ▶ have the ability to determine the range of legal addresses.

# Outline

- 1 Concept of Address Translation
- 2 Contiguous Memory Allocation**
- 3 Paging
  - Concept and Basic Scheme
  - Fragmentation
  - Protection and Sharing
- 4 Implementing Paging
  - Access Latency Too High?
  - Page Table Too Large?
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Table

# Contiguous Memory Allocation

Each process is contained in a *single section of memory* that is *contiguous* to the section containing the next process

- ▶ Use a simple realization of the base-limit registers
  - ▶ Limit register and relocation register (as the base register)
  - ▶ Memory protection.
  - ▶ Memory allocation.

# Memory Allocation

- ▶ Variable Partition Scheme. The OS assigns processes to variably sized partitions in memory, where each partition may contain exactly one process.
  - ▶ General dynamic storage allocation problem.
    - ▶ First-fit, best-fit, and worst-fit strategies.
- ▶ Problems?
  - ▶ Internal fragmentation.
  - ▶ External fragmentation.

# Outline

- 1 Concept of Address Translation
- 2 Contiguous Memory Allocation
- 3 Paging**
  - Concept and Basic Scheme
  - Fragmentation
  - Protection and Sharing
- 4 Implementing Paging
  - Access Latency Too High?
  - Page Table Too Large?
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Table

# Concept of Paging

A memory management scheme that permits a process's physical address space to be non-contiguous.

## Basic Paging Scheme

- ▶ Physical memory. Breaking physical memory into fixed-sized blocks called *frames*.
- ▶ Logical memory. Breaking logical memory into blocks of the same size called *pages*.
- ▶ Page size  $\equiv$  frame size
- ▶ A logical address consists of a page number and a page offset.
- ▶ A physical address consists of a frame number and a frame offset.
- ▶ Page offset  $\equiv$  frame offset
- ▶ MMU translates a logical address to a physical address via a page table.

## Querying Page Size

On Debian Linux,

```
1  $ getconf PAGESIZE
2  $ man 2 getpagesize
```

# Examples and Exercises

Let's examine a few examples and do a few exercises ...

# Fragmentation

- ▶ Does paging has internal fragmentation?
- ▶ Does paging has external fragmentation?

# Protection and Sharing?

## Recall

- ▶ Policy. Each process has a separate memory space.
  - ▶ Protection. Keep each process isolated.
  - ▶ Sharing. Allow memory to be shared between processes.
  - ▶ Virtualization. Provide applications with the illusion of “infinite” memory.

# Protection

- ▶ Associating protection bits with each frame
  - ▶ Read-write, read-only, executable?
  - ▶ Valid and invalid?
  - ▶ Page table size? (page-table length register (PTLR))?
- ▶ Storing these bits in page table

## Shared Pages

Paging allows us to realize shared library and shared memory efficiently.

# Outline

- 1 Concept of Address Translation
- 2 Contiguous Memory Allocation
- 3 Paging
  - Concept and Basic Scheme
  - Fragmentation
  - Protection and Sharing
- 4 Implementing Paging
  - Access Latency Too High?
  - Page Table Too Large?
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Table

# Access Latency of Basic Paging Scheme

How much time does it take to access the memory?

- ▶ Let's compare the Variable Partition scheme and the Basic Paging scheme.
- ▶ Constraints.
  - ▶ For most contemporary CPUs, page tables are large, and can only be kept in main memory.
  - ▶ page tables are located via a page-table base register (PTBR)
  - ▶ page table size is in page-table length register (PTLR)

## Translation Look-Aside Buffer

- ▶ To reduce access latency of the paging scheme, introduce cache memory called the Translation Look-Aside Buffer (TLB)
- ▶ Realized via an associative, high-speed memory.
  - ▶ When the associative memory is presented with an item, the item is compared with all keys simultaneously.
  - ▶ A TLB lookup in modern hardware is part of the instruction pipeline, essentially adding no performance penalty.

## Querying TLB

On Debian Linux,

```
1 $ sudo apt-get install cpuid
2 $ cpuid | grep -i tlb
```

# Access Latency with TLB

Let's do a few exercises to analyze access latency with TLB?

## How big is a page table?

- ▶ Page tables can be very large if implemented in straight-forward fashion.
- ▶ Let's do some exercise ...

# Hierarchical Page Tables

To avoid allocating a page table contiguously in main memory, we page the page table.

## 2-Level Paging

Let's consider 32-bit logical address space, and a logical address will have the format like the following,

page number		page offset
$p_1$	$p_2$	$d$
10	10	12

There are two level of pages tables

- ▶ Outer page table indexed by  $p_1$ .
  - ▶ Inner page table indexed by  $p_2$ .
1. How does TLB look like?
  2. What if we use 2-level paging for 64-bit logical address spaces?

## 3-Level Paging?

Let's consider 64-bit logical address space. We would structure a logical address as follows,

page number			page offset
$p_1$	$p_2$	$p_3$	$d$
32	10	10	12

How big is the 1st outer page table? How about the following?

page number					page offset
$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$d$
12	10	10	10	10	12

1. How does TLB look like?
2. What is the access latency when there is a TLB miss?

# Hashed Page Tables

Use a hash table (sometimes called a map)

- ▶ Key.  $H(\text{pagenumber})$
  - ▶ Value. A linked list of  $(\text{pagenumber}, \text{framenumbers})$ 
    - ▶ Why do we need a list instead of a single value of  $(\text{pagenumber}, \text{framenumbers})$ ?
1. How does TLB look like?
  2. What is the access latency when there is a TLB miss?

# Inverted Page Table

Observation. A computer system typically has small amount of physical memory when compared to logical address space.

- ▶ Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
  - ▶ Conceptually in a page table, an entry is for a page
  - ▶ In an inverted page table, an entry is for a frame (thus inverted)
- ▶ One page table entry for each real page (or frame) of memory. Each consists of
  - ▶ process identification information, and
  - ▶ frame information.
- ▶ Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
  - ▶ Use hash table to limit the search to one — or at most a few — page-table entries
  - ▶ TLB can accelerate access