Operating Systems

Paging Introduction

Hongliang Liang, BUPT

Sep. 2022

Major Challenge: External Fragmentation

Compaction requires high copying overhead

Basic assumption until now: memory is allocated **contiguously** in **variable** sizes

Why not allocate memory in **non-contiguous** and **fixed-size** units?

- no external fragmentation!
- internal fragmentation < 1 unit

How big should the units be?

- Smaller: better for internal fragmentation
- Larger: less management overhead

Paging: Non-contiguous fixed-size allocation

Each fixed size unit in physical memory is called a **physical frame (or "frame")**

- Physical frame size $= 2^n$ bytes of physical memory

Each fixed size unit in the virtual address space of a program is called a **virtual page (or "page")**

- Each page has the same size as a frame

Pages are contiguous, but frames allocated to the address space are non-contiguous

Paging: Dynamic Address Translation

But how do we associate physical frames with processes?

- Specifically, need to map virtual address space to noncontiguous physical frames **at run time**

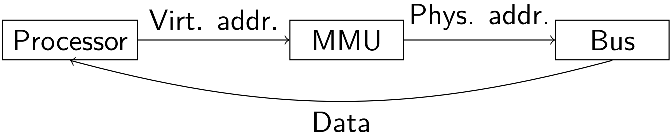
Recall: MMU performs dynamic address translation

- Processes use virtual addresses
- CPU puts physical addresses on the shared bus
- Hardware support for virtual to physical address translation
- A simple base and bounds MMU adds an offset to a virtual address to produce a physical address
- Can we make the MMU "smarter" than base and bounds?

Recall: Memory Management Unit (MMU)

The MMU provides a layer of indirection between the processor and the physical memory

More flexibility!

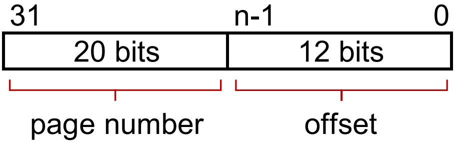


Virtual Addresses

Consist of (**Page number, byte offset** in page)

Low order *n* bits are the byte offset

Remaining high order bits are the page number



Example: 32 bit virtual address

Page size $= 2^{12} = 4\text{KB}$

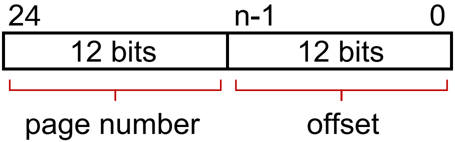
Address space size $= 2^{32}$ bytes $= 4\text{GB}$

Physical Addresses

Consist of (**Frame number, byte offset** in page)

Low order *n* bits are the byte offset

Remaining high order bits are the Frame number

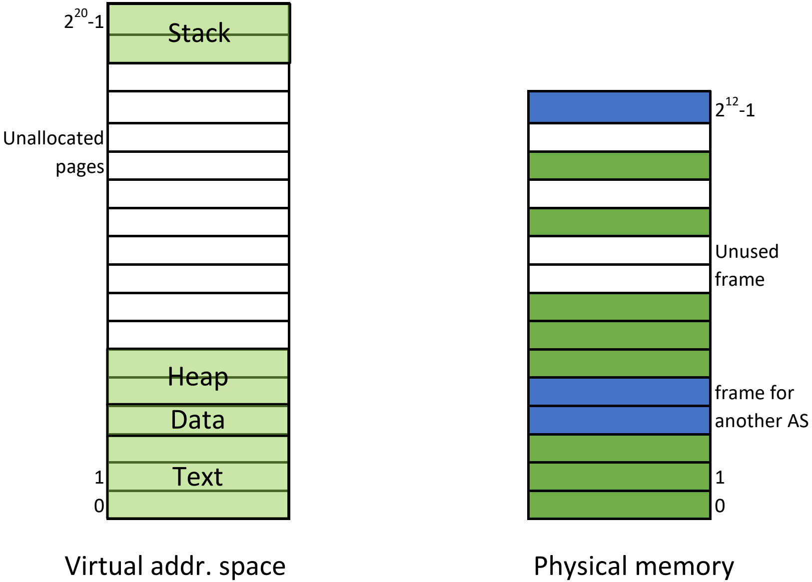


Example: 24 bit physical address

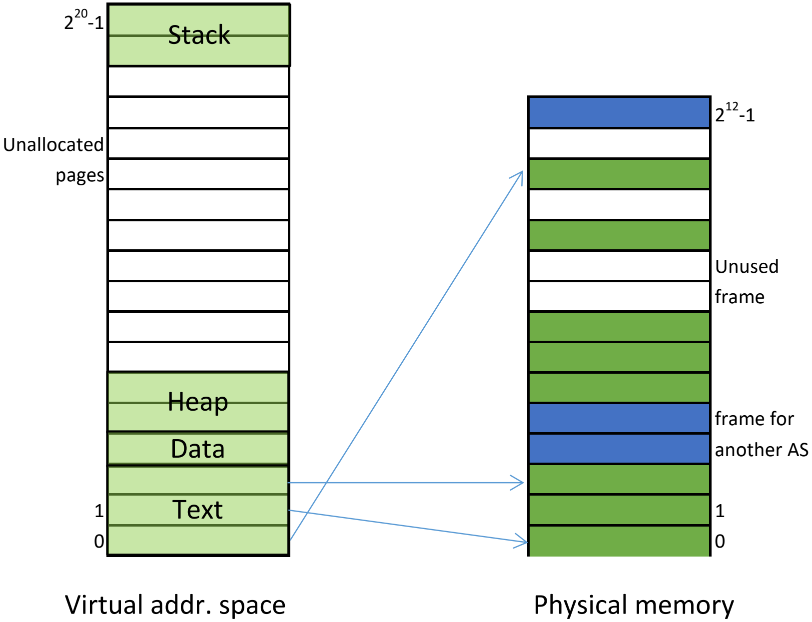
Frame size $= 2^{12} = 4$KB (same as page size)

Address space size $= 2^{24}$ bytes $= 16$MB

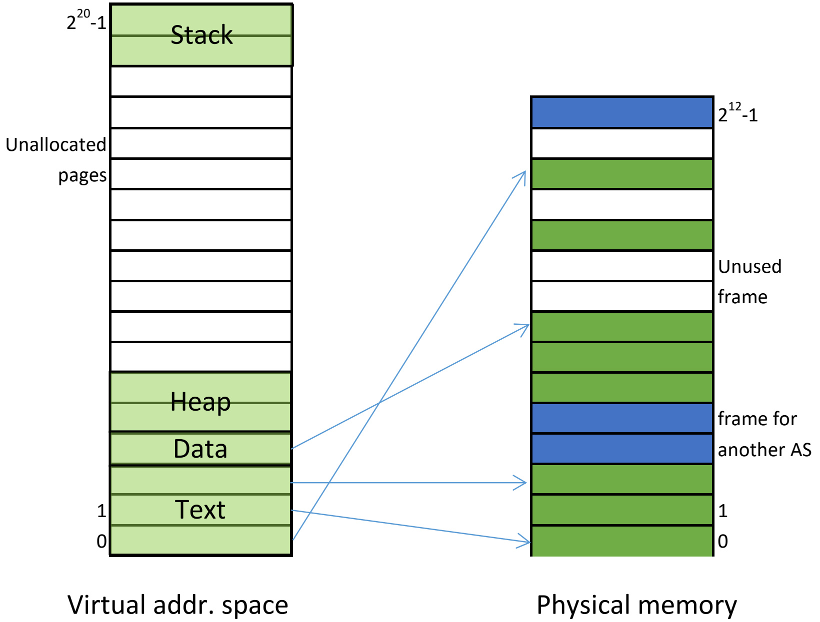
Dynamic Address Translation



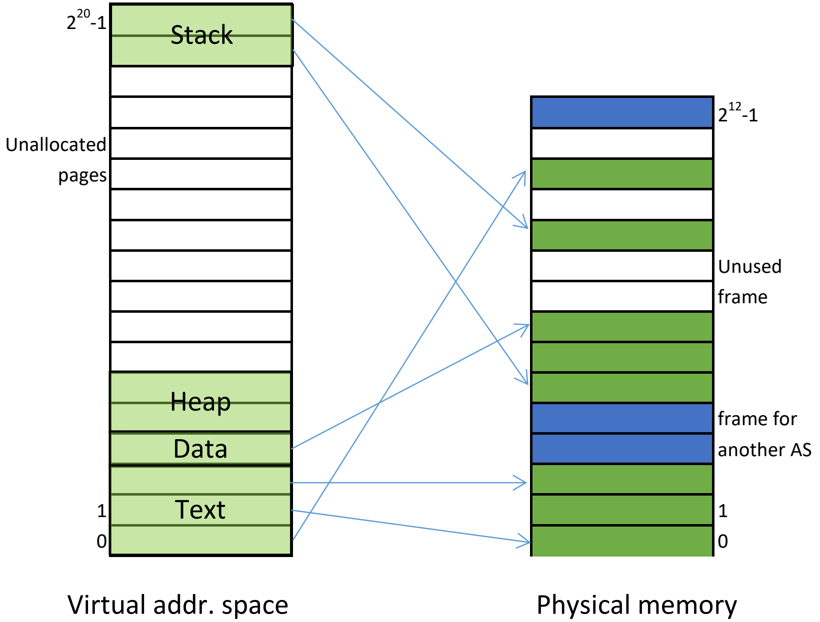
Dynamic Address Translation



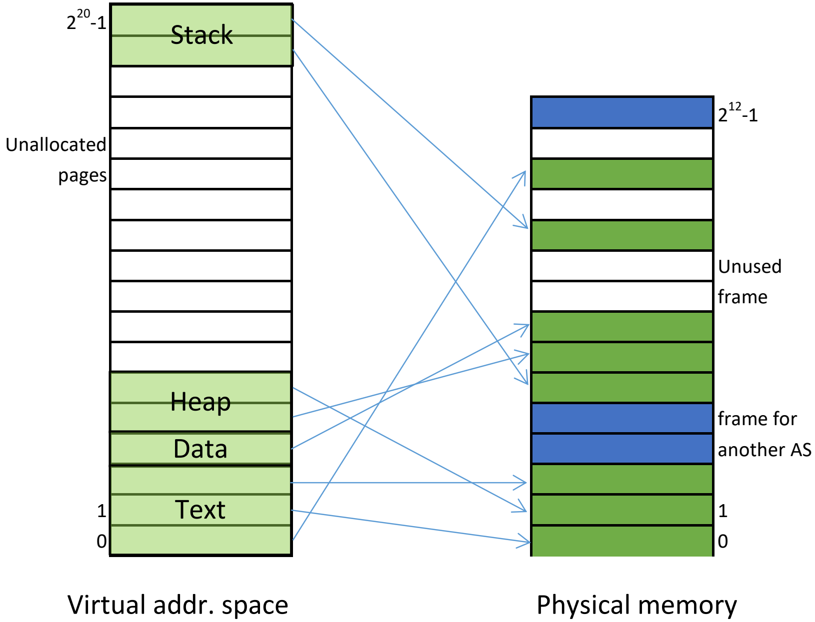
Dynamic Address Translation



Dynamic Address Translation



Dynamic Address Translation



Translating Virtual to Physical Addresses

MMU needs to map page numbers to frame numbers on **each memory reference**

- Conceptually, MMU has a separate register for each page number
- The register for each page contains the frame number
- Similar to a base register, except register value is substituted for (rather than added to) the page number
- Why don't we need a bounds register for each page?

Where is all this translation information stored?

Page Table: Where the Page Map is Stored

Virtual to physical address mappings are stored in a **page table** in the main memory

- Typically we have one page table per process

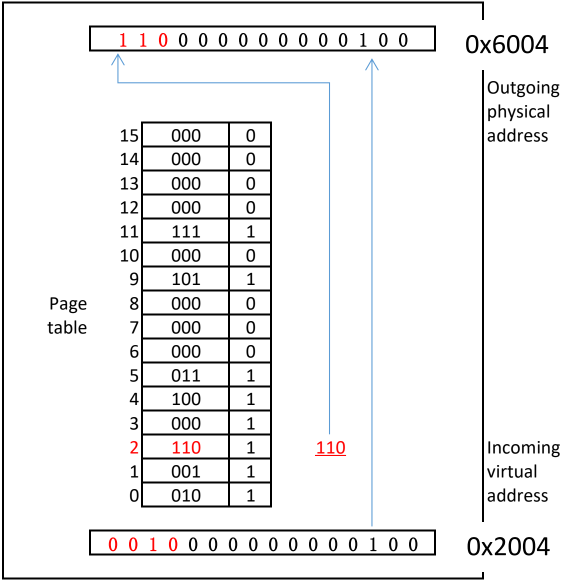
A page table contains a number of **page table entries**

- Each entry contains a mapping from a page to a frame
- Each entry also contains various useful bits
- Example: The **Valid** bit says whether the mapping is valid or not

A Linear Page Table

```
Virtual address size:
 16 bits
Page size:
 12 bits
# of pages:
 16 (4 bits)

Physical address size:
 15 bits
# of frames:
 8 (3 bits)
Page table entry size:
 4 bits
```



Page Table Entries

Each entry contains a mapping from a page to a frame, it contains:

- Frame number that the page is mapped to
- The valid bit
- The dirty bit: has the content of frame been changed?
    - Intuitively, why do we need the dirty bit?
- Protection bits: read/write/execute
- Other bits that we will discuss later