Main Memory I

DM510 Operating Systems

Lars Rohwedder

Windows macOS Android iOS

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides: https://www.os-book.com/OS10/slide-dir/index.html

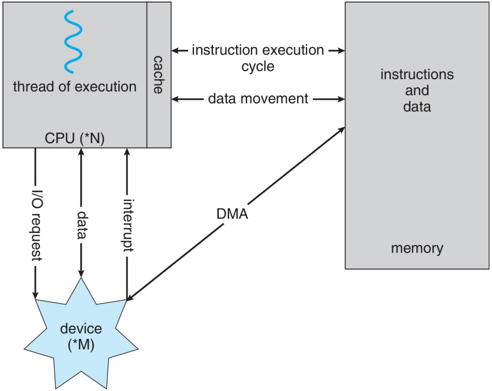
Today’s lecture

- Chapter 9 of course book

Overview

Role of main memory

CPU has direct access to main memory by loading and storing register values from and to main memory addresses

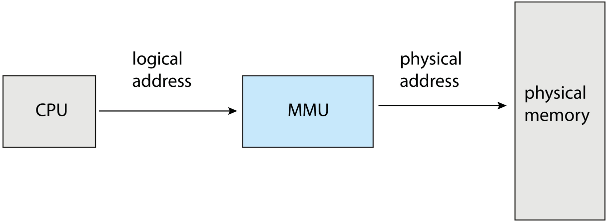


- Moving data between main memory and registers takes much more time than a CPU instruction (clock rate). It is called **memory stall** when the CPU needs to wait for data movement before continuing
- Recently used (or sometimes predicted) memory addresses are stored in **cache** for faster access

Logical and physical addresses

- CPU's instructions load and store to **logical** addresses. They are different from **physical** addresses that memory unit sees

- **Memory-management unit:** hardware that translates logical to physical addresses
- Many variants of logical-to-physical translation possible



Purpose of logical addresses

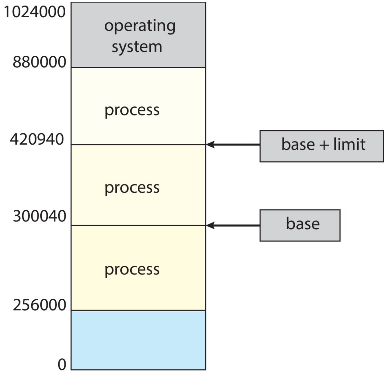
- Programs can have simple address space (continuous, huge, private), even if allocation of physical memory may be complicated
- Protect memory of processes from each other

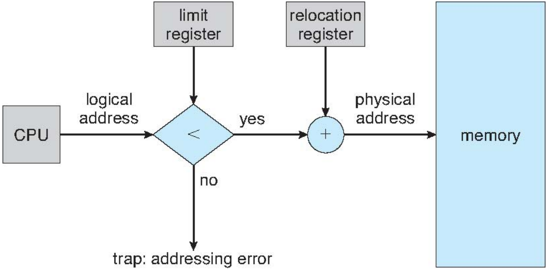
Contiguous Allocation

Contiguous allocation

Oversimplified version of address translation. Not used in modern systems

- Each process receives a contiguous section of physical memory addresses
- Before executing user code, the kernel sets the following registers, which have priviledged access:
  relocation register: first physical address (base) for process
  limit register: length of section





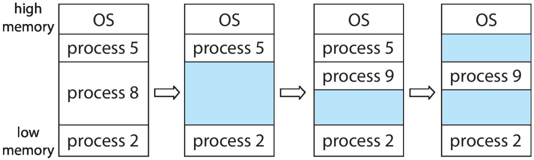
physical address = relocation register + logical address

Choosing memory sections

- We need a **variable size** partition of memory: cannot afford to give every process the same (maximum) amount of memory

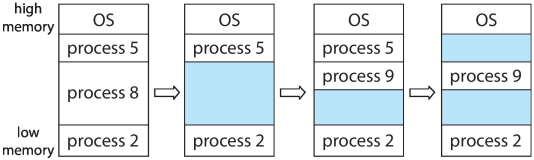
Choosing memory sections

- We need a **variable size** partition of memory: cannot afford to give every process the same (maximum) amount of memory
- Since new processes start and terminate, **holes** of free memory occur



Choosing memory sections

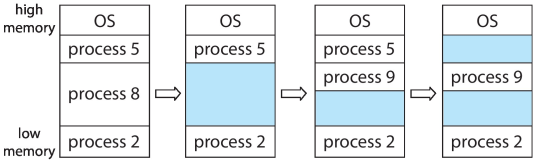
- We need a **variable size** partition of memory: cannot afford to give every process the same (maximum) amount of memory
- Since new processes start and terminate, **holes** of free memory occur



- Which hole of memory should we take for a new process?
  First-fit: First hole that is big enough
  Best-fit: Smallest hole that is big enough
  Worst-fit: Biggest hole that is big enough

Choosing memory sections

- We need a **variable size** partition of memory: cannot afford to give every process the same (maximum) amount of memory
- Since new processes start and terminate, **holes** of free memory occur



- Which hole of memory should we take for a new process?
  First-fit: First hole that is big enough
  Best-fit: Smallest hole that is big enough
  Worst-fit: Biggest hole that is big enough
- Empirical evidence shows that first-fit and best-fit perform better than worst-fit

Fragmentation

Continuous allocation (as well as most other methods) can result in inefficient memory usage, due to two reasons:

- **External fragmentation:** there is enough free memory, but it is not contiguous
- **Internal fragmentation:** the space allocated to processes is higher than requested (for example, rounded up to power of 2)
- **50% rule:** for N blocks allocated, approximately $0.5N$ blocks are lost due to fragmentation
- **Compaction:** shuffle around memory to make free memory contiguous (typically very slow)

Paging

Pages and frames

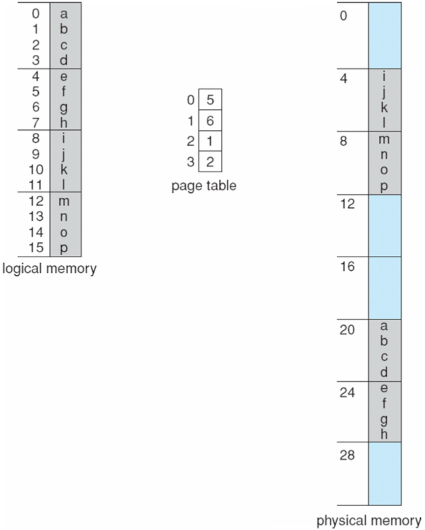
Paging is used to avoid external fragmentation and to make memory (re-)allocation to processes more flexible

- Physical memory is partitioned into **frames** of specific size, for example 4MB
- Logical memory (of each process) is partitioned into **pages** of the same size
- A per-process **page table** maps pages to frames
- Page tables reside in memory themselves. CPU has **page table base register** that stores location of page table for active process (needs to be updated at context switch)

Choosing the page/frame size

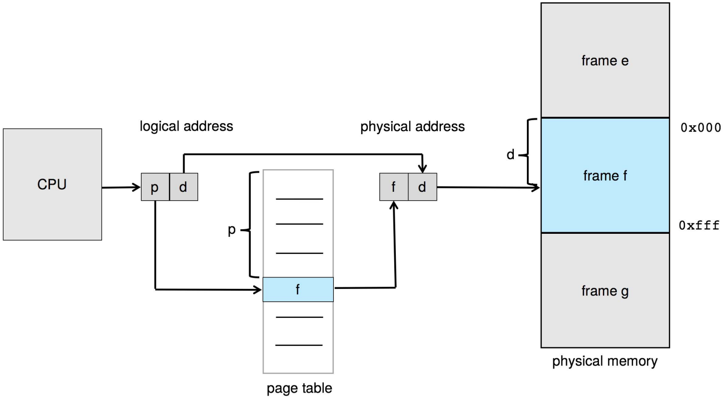
Memory allocation is only possible in multiples of page size

- too large page size leads to high internal fragmentation
- too small page size leads to large page tables



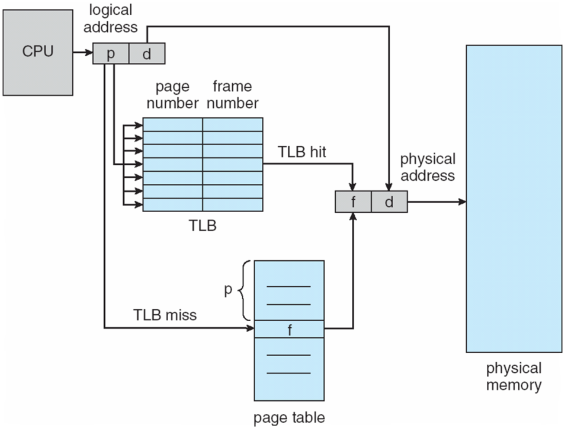
Address translation

- High order bits: page/frame number
- Low order bits: offset within page/frame
- Since page table is also stored in memory, each memory instruction results in two memory accesses



Translation look-aside buffer

- Small lookup table in hardware (TLB) stores recently used page numbers and their corresponding frame numbers



- Page in TLB: very fast access
- Page not in TLB: need to lookup table in main memory (slow), add page/frame combination to TLB
- Size of TLB is limited

Effective access time (TLB)

How long does a logical memory access take on average? (effective access time)

- Suppose we need 10 nanoseconds for physical memory access
- Further, in 80% of the accesses we find page in TLB (**hit ratio**)
- Thus, in 20% we need a second memory access
- EAT $= 0.8 \cdot 10 + 0.2 \cdot (10 + 10) = 12$ nanoseconds
- If hit ratio was 99%, then EAT $= 0.99 \cdot 10 + 0.01 \cdot (10 + 10) = 10.1$ nanoseconds $\Rightarrow$ 1% slowdown