
Chapter 8: Main Memory

This chapter deals with how memory is managed and allocated.
It will focus on multitasking OSes that must share memory among all processes.
Figure (memory of processes)

Linkers/Loaders

Linking - combines all object modules into a single executable file.
          The Pthread library is invoked with the linker when we entered -lpthread
Loading - this loads the file from disk into memory where it is available for scheduling by the OS.
          The loader is typcially invoked by entering the program name on the command line - i.e. a.out
or double-clicking on an icon.

Dynamic Loading - To conserve memory, defer loading (also called lazy loading) of a library until it is
needed.
                  (We did this with Linux loadable kernel modules.)
Dynamic Linking - Similar to dynamic loading, except that even the linking of a library is deferred until
it is needed.
                  Hence, the library is linked during runtime. Windows DLLs are a good example of this.

Dynamic linking is especially useful for shared libraries.
Consider the pthread library - it could be loaded either statically or dynamically.
Static loading requires the library be loaded directly into the address space for the process.
This is especially wasteful considering if other processes use the same library.
Dynamic loading allows the library to be loaded dynamically at run time.
Using virtual memory techniques (covered later), the separate processes using the shared library
  can all share the same memory allocated for the library.

Address Binding

Source code refers to identifiers such as the following:
count = 5;
"count" is symbolic and ultimately must be mapped to a physical address.

Address binding can take place during:
1. compile time - this requires knowing where the program will be loaded into memory.

2. load time - the compiler generates relocatable code and the loader knows where to load the code into
memory.

3. run time - binding an address now allows the program to be moved while it is running.

Logical vs. Physical Address Space

An address generated by the CPU is considered logical - it doesn't refer to an actual memory location.
The memory management unit (MMU) maps a logical address to its corresponding physical address.

A MMU may work using dynamic relocation - Figure 7-4.
This is rather simplistic and we will look at more realistic (and complicated) models shortly.

Swapping

Swapping is the act of moving processes between memory and a backing store.
This is done to free up available memory. Swapping is necessary when there are more processes than
available memory.
At the coarsest level, swapping is done a process at a time. That is, an entire process is swapped in/

out.
We will look at finer-grained swapping later in this chapter.

Mobile systems generally do not support swapping (i.e. iOS and Android) as flash memory is typically used for file systems in mobile devices.
Three primary reasons:
(1) There is typically less flash memory on mobile systems that hard disk space on desktop and laptops systems.
(2) Flash typically has a limited number of writes before it becomes less reliable.
(3) Often poor throughput between main memory and flash memory in mobile devices.
Both Android and iOS adopt a strategy where they request apps to release memory when available memory becomes low.
If there still isn't sufficient memory, the operating system may simply terminate applications.

Memory Allocation

Approach: Contiguous Memory Allocation - Allocate a fixed-sized partition for each process in memory.

The problem with this is that it can leave to a series of holes scattered around memory.

How to allocate a free hole:

1) First Fit

2) Best Fit

3) Worst Fit

Regardless of the scheme, external fragmentation may result.
This is having many holes too small to use.
We could perform compaction (ala garbage collection), but it is costly.

Approach: Paging
- Allocate memory in smaller-size chunks.

Physical memory is broken up into page frames.

Logical memory is broken up into pages.

Logical pages are mapped to corresponding frames in memory.

A logical address consists of

Page Number (P) Page Offset (D)
The page number P is used as an index into the page table for the process.

The page table contains the corresponding physical page frame for the logical page.

Page Size

There will be some internal fragmentation in a page, the larger the page, the more memory lost to fragmentation.

Typical sizes are 4 KB or 8 KB, although several systems provide support for multiple page sizes.
Solaris (on Sparc systems) provides support for 8 KB and 4 MB page sizes. Intel Pentium provides support for 4 KB or 4 MB.

To get the pagesize on your Linux system, enter
getconf PAGESIZE
on the terminal.

Page Allocation

Frames are allocated to a process and the page table for each process is adjusted.
The OS keeps track of available page frames in a free frame list.

Important Points:

*** All addresses must be resolved in the page table. We will talk more about this with hardware support

*** When a context switch occurs, the OS must now reference the page table for the process being switched in and this increases context switch overhead.

< Address Translation Exercise >
Hardware Support

Hardware must support paging.
Every memory access requires 2 memory accesses:

(1) accessing the page table

(2) accessing the actual byte

If the page table can be represented as a set of hardware registers, step (1) can be done in hardware and not in memory. However, implementing a page table in registers would require too many registers for a large address. Consider a 16-bit address - this translates to 64 KB. If the page size were 4 KB, this would require 16 registers. Now consider a 20-bit address - this would require 256 registers. Many architectures are now 64 bits which leads to a prohibitive number of registers.

Translation Look-Aside Buffer (TLB)

The TLB is a set of associative registers

TLB Hit --> go to memory.

TLB Miss --> "walk" the page table.

There is only one TLB (well, sometimes there is a separate TLB for code and data) and it must be flushed and updated for every new process.
 This also adds to context switch time.

TLB Management

TLBs generally consist of 32-64 entries containing the most recent memory accesses.
Replacement strategies are generally FIFO.
Furthermore, some systems allow a TLB entry to be wired down to prevent flushing or replacing it.
Kernel code is a good candidate for wiring down in the TLB.

Some TLBs store address-space identifiers (ASIDs) in each entry of the TLB.
An ASID uniquely identifies eahc process and is used to provide accress space protection for that process.
Then the TLB attempts to resolve page numbers, it ensures the ASID for the process matches the ASID in the page table entry.
If they do not, it is treated as a TLB miss.
ASIDs also allow the TLB to contain entries for several different processes at the same time.
ASIDs also avoid the flushing of the TLB between context switches.

Many architectures now provide separate TLBs for instructions and for data.

TLB hit ratio - the percentage of time memory references are resolved in the TLB. Given a "warm" TLB, hit ratios of 90% are not uncommon.

Protection

Protection is ensured for each page frame entry in the page table.

- Read/Write bits ensure read-only pages aren't being written to.

- Valid/Invalid bits ensure the entry is valid or not (i.e. the page is in fact part of the process's address space.)

Structure of the Page Table

The page table shown in Figure 7-10 is considered a single-level page table. However, consider the following:

--> logical address is 32 bits

--> page size is 4 KB (requires 12 bits)

This equals 2^(20) possible page frames. Each page table entry might be 4 bytes - the page table would be 4 MB. Now consider 64-bit addresses ....

Alternatives

(1) Multilevel Paging (aka Hierarchical Paging)

This requires dividing up the logical address into 3 components:

P1
P2
D

Where P1 and P2 are the page number entries and D is the offset in the page frame.

Because address translation works from the outer page table inwards, this is also known as a forward-mapped page table. Intel architectures works this way.

Because the root (outer) page table is always referenced, it is wired down in memory.

For larger address spaces than 32 bits, multilevel paging is required. The Sparc architecture (with 32 bits) supports three-level paging.

Note that even multi-level paging is insufficient for a 64-bit architecture. The 64-but Ultra Sparc would require seven levels of paging!

(2) Hashed Page Tables

This is more appropriate for larger address sizes. The virtual page number is hashed into the page table. The hash bucket contains all page frames that hash to this location. The hash bucket is searched for the matching page number, once found, the corresponding physical page frame is extracted and the physical address is determined.

(3) Inverted Page Tables

An inverted page table reverses the idea of a page table. Rather than having an entry for each logical page, an inverted page table has an entry for each physical page frame.

Each entry in the inverted page table contains the virtual page mapping to the physical frame. To speed up searches, inverted page tables are often implemented as hashed page tables where the value hashed is (page number + ASID).

Shared Pages
Most operating systems allow pages to be shared by several different processes. Sharing can occur for

- libraries

- memory

- executable files

Virtual Memory (Chapter 9)

```
-----------------------------------
 - virtual memory is an abstraction of physical memory
 - virtual memory can be larger than physical memory
 - multiple virtual addresses can be mapped to a single physical address
   (for sharing of system libraries)
   (for sharing memory between processes)

Demang Paging
--------------------
 - only pages that are going to be used need to be in physical memory
 - when a process needs a page that is not in physical memory, we have a PAGE FAULT and a trap
(interrupt) occurs
 - reacting to a page fault:
    - find a free frame in physical memory
    - schedule load of data into memory
    - when load is complete, update page table with address of loaded data
    - restart the instruction that was interrupted
 - performance in memory access is directly proportional to page-fault rate

Page Replacement
-----------------------
 - as more processes are running concurrently, more memory is used
 - when we run out of physical memory, the OS needs to decide how to free memory
 - how do we choose a victim?
 - we use page replacement to solve this problem
 - algorithms:
    - FIFO (simple, possibly inefficient)
    - optimal (replace the page that will not be used for the longest period of time) requires
prediction / foreknowledge
    - least recently used (LRU)
    - least recently used approximation (LRUA) (bit to store use of page)
    - improvements: additional reference bits.  second chance algorithm.


Thrashing
---------------------------
 - high paging activity is called thrashing
 - we may say a process is "thrashing" if it is spending more time paging than executing
 - Figure CPU UTILIZATION GOES UP WITH MULTIPROGRAMMING UNTIL THRASHING STARTS, then drops dramatically
 - program locality:
 - working set model: keep a fixed number of page references for a process (exploits locality)
 - page fault fequency model: set upper and lower bounds for fault rate.  allocate when upper bound is
crossed, deallocate when lower..

Memory Mapped Files
---------------------------
 - map an area of memory as a file
 - enables file operations on memory space (fast!)
 - lots of "files" in the virtual filesystem are just pointers to memory locations




-----------------------------
Chapter 10: Mass Storage
-----------------------------

We have discussed storage in the CPU, storage in RAM.
Now, we will discuss mass storage aka secondary storage.

Types:
 - Magnetic Tape (reel-to-reel diagram with heads)
   tape must be wound to the right place to read data
 - Optical Disk (CD, DVD, Blu-ray)
   disk must be spun to the right place to read data
 - Magnetic Disk
   platters on a spindle, tracks (cylinder=track across platters), sectors (part of a track)
```

```
   head on an arm reads from the disk
   disk must be spun to right place to read data
 - Solid State Drives (aka Flash)
   no moving parts --> faster access time
   higher cost per MB
   shorter lifespan

Addressing:
 - treated as one-dimensional array of logical blocks (usually 512 bytes)
 - logical block maps to cylinder-track-sector address
   (abstraction hides details of defective blocks, variable sectors per track)

Attachment to the machine:
 - Host-attached storage (local storage)
    - attached to an I/O bus
      examples: ATA, SATA, eSATA, USB, fiber channel
    - a disk controller built into each drive communicates with a host controller on the machine
 - Network attached storage:
   storage on a network, accessed with storage access protocol
   protocal examples Unix NFS, Windows CIFS, iSCSI (SCSI over IP)
 - Storage Area Network:
   separates communication with storage from other network traffic

Disk Scheduling (for magnetic disk)
 - FCFS
 - SSTF (shortest seek time first)
 - SCAN (elevator algorithm)
 - C-SCAN (circular SCAN) (only reads one direction)
 - LOOK / C-LOOK, consider requests before reverse or reset

SSDs: read time is uniform, but write time is not.
      reads are serviced FCFS, but adjacent writes are merged
```