8 – Memory Management

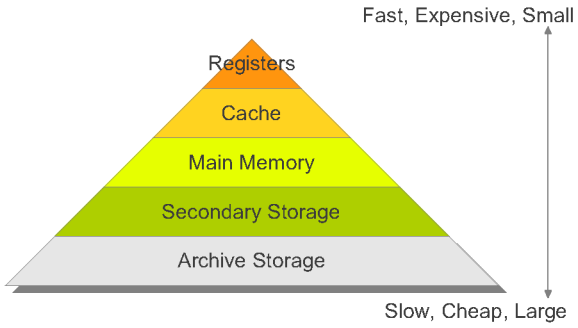
8 – Memory Management

- Various techniques to manage memory

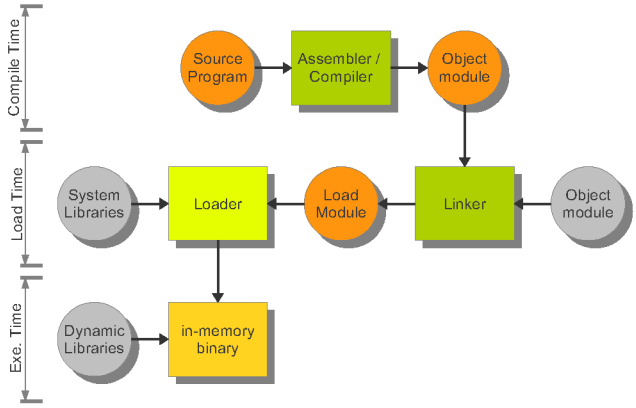
Background

- Memory and registers are only storages CPU can access directly.
  - Program must be loaded into memory to run.
- Registers access in one clock (or less).
- Main memory access can be many more cycles.
- Caches sit between memory and registers
  - Ease differences of speed
- Memory protection using base and limit registers.
- Code and data must be addressable.

Memory Hierarchy



Address Binding



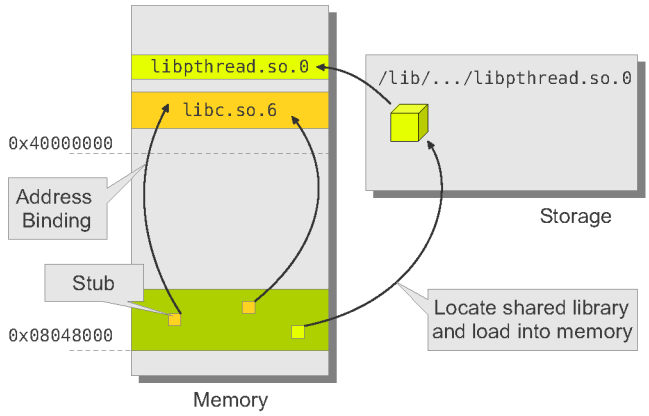
(cont'd.)

- **Compile time**: if memory location is known priori, *absolute code* can be generated.
  - Must be recompiled if starting location is changed.
  - e.g., MS DOS
- **Load time**: must generate *relocatable code* if memory location is not known at compile time.
  - Delayed address binding until load time.
  - Just reload if starting location is changed.
- **Execution time**: delayed memory binding until run time.
  - Need hardware support for address map, e.g., base and limit registers.

Dynamic Linking and Shared Libs.

- Delay address binding until execution time.
  - Because addresses of share library routines are unknown at compile/link time.
- Put a *stub* to locate the routine of the library.
  - The stub checks whether the library is in the memory, and load the library if it is necessary.
  - The stub replaces itself with address of the routine, then execute it.
- Many programs may share the same library.
  - This requires helps from the operating system since processes may be protected from one another.

(cont'd.)



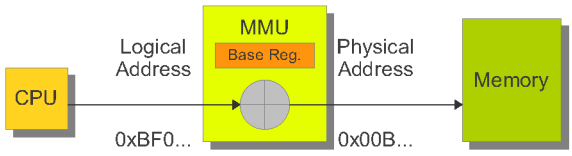
(cont'd.)

- On Linux, you may try
  - `readelf` to find dynamic symbols in an executable
  - `ldd` to find address binding of each shared library

Logical and Physical Address Spaces

- Logical address is generated by CPU to MMU.
- Physical address is addressable by MMU to refer to physical location of data in the memory.
- In compile-time and load-time address binding, logical = physical.
- For execution-time scheme, logical ≠ physical.
  - In this case, logical address is referred to as *virtual address*.

(cont'd.)



- The concept of a logical-address space that is bound to a separate physical-address space is central to proper memory management.

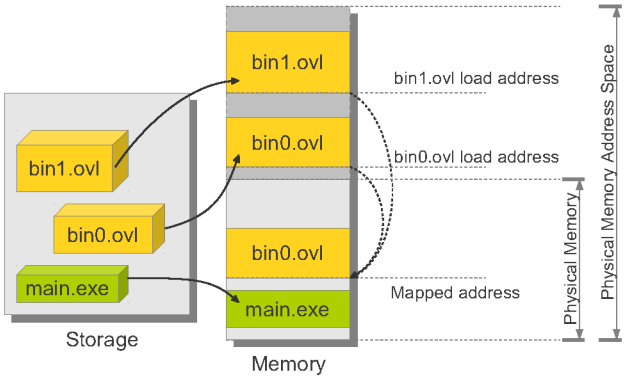
The Real Mode

- 8086 has 20-bit address space
  - 2^{20} = 1 MB addressable memory.
- 8086 has 16-bit index registers. So, it could address only 2^{16} = 64 kB.
- To address larger memory, segment registers (`CS`, `DS`, `SS`, `ES`) are used.
- Logical address = segment:offset ~ 32 bits.
- Physical address = (16 × segment) + offset
  - So, each physical address could be referenced by 2^{12} segment:offset pairs.
  - Small program (<= 64kB) does not need to be relocated.

Overlays

- In DOS era, physical memory is relatively small.
  - e.g., original IBM PC equipped with 64 kB RAM.
- Programs are larger than available memory and an entire program cannot be loaded.
- Overlay is a technique to split code into several objects, keep only those are required in the memory at any given time.
- When the others are needed, load them to ones that are no longer needed.

(cont'd.)



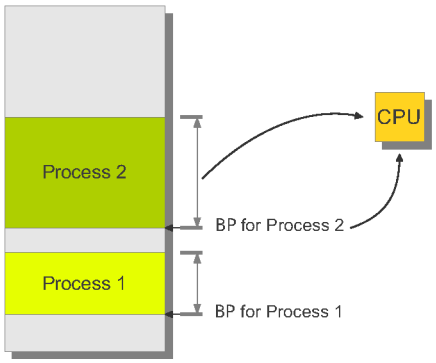
(cont'd.)

- The operating system has nothing to do with this.
  - So, programmers have to do this by themselves.
  - Linkers may provide support for overlays.
- Although this technique is old, it is still useful for today's programming, especially in embedded systems.
  - Mainly because such systems have small amount of memory.

Contiguous Memory Allocation

- A process must be loaded into available linear address space.
  - Imply contiguous memory allocation.
- For multiple processes, the system must track memory usage.
  - For each process, two registers are required.
    - **Base address register** indicates start of the address space for the process.
    - **Limit register** indicates size of the space allocated.

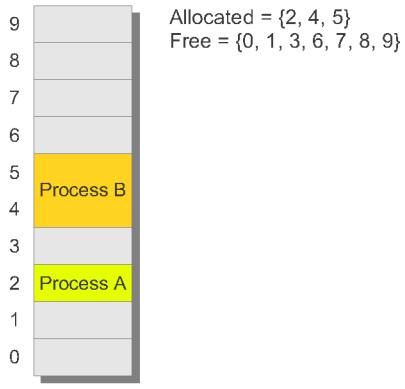
(cont'd.)



(cont'd.)

- Multiple-partition method
  - Divide memory into a number of fixed-size partitions
  - Each partition may contain exactly one process
    - Degree of multiprogramming is bound at the number of partitions
  - A process may occupy more than one partition.
  - When a process terminates, the partition is available.
    - Holes
  - When creating a new process, it has to find a hole large enough to accommodate it.
  - The operating system must maintain ...
    - Allocated partitions
    - Free partitions

(cont'd.)

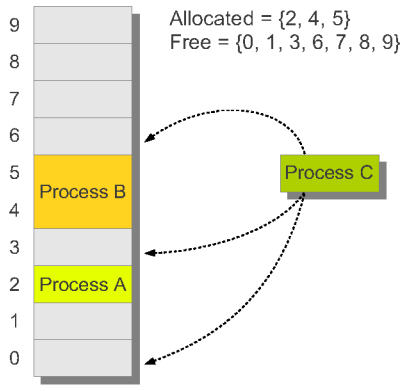


Allocated = {2, 4, 5}
Free = {0, 1, 3, 6, 7, 8, 9}

(cont'd.)

- The problem is that how to find a list of free holes that could accommodate a process size n
  - Dynamic storage allocation problem.
- **First-fit**: the first hole that is big enough.
- **Best-fit**: the smallest hole that is big enough.
  - Smallest leftover hole
- **Worst-fit**: the largest hole that is big enough.
  - Largest leftover hole

(cont'd.)



Allocated = {2, 4, 5}
Free = {0, 1, 3, 6, 7, 8, 9}

Fragmentation

- **External fragmentation**: total available memory is large enough, but it is not contiguous.
  - Compaction – shuffle, relocate, to create a large hole. This is possible only if relocation is dynamic, and is done at execution time.
- **Internal fragmentation**: allocated memory is (slightly) larger than requested.
  - e.g. similar to sector/cluster of disk

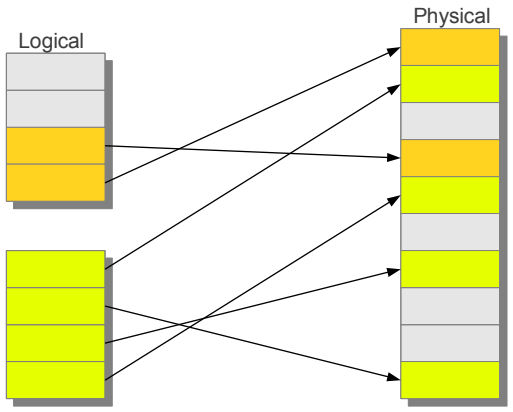
The Protected Mode

- Introduced in x86 architecture by the release of 80286 processor in 1982.
  - 24-bit address space.
  - Not widely used since the protected mode is not backward compatible with the real mode.
- i386 improvements in memory management
  - 32-bit address space.
  - Paging.
  - Virtual memory.

Paging

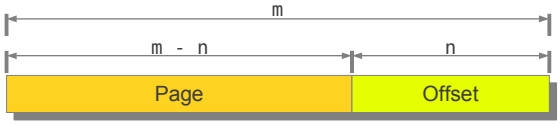
- Paging allows the physical-address space of a process to be non-contiguous.
- Divide physical memory into fixed-size blocks called *frame*
  - Frame size is power of 2; 512 bytes – 16 MB.
  - Imply internal fragmentation
- Divide logical memory into blocks of the same size of frame called *page*.
- A program that requires n pages needs to find n frames.
  - The n pages **must** be contiguous but **not** necessary for the n frames.

(cont'd.)

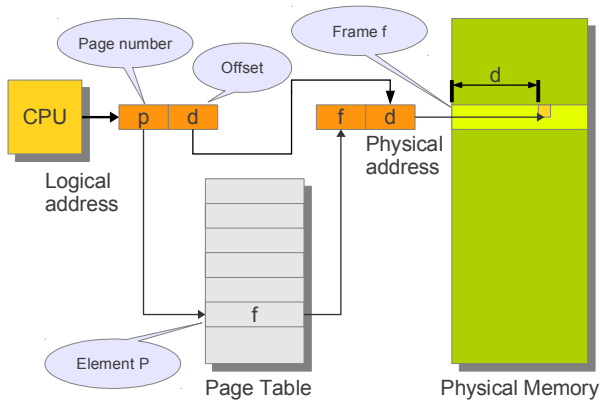


(cont'd.)

- Let logical address space = 2^m and page size = 2^n addressable units (i.e., byte), then
  - The high-order $m - n$ bits designate the page number.
  - The low-order n bits designate the page offset.

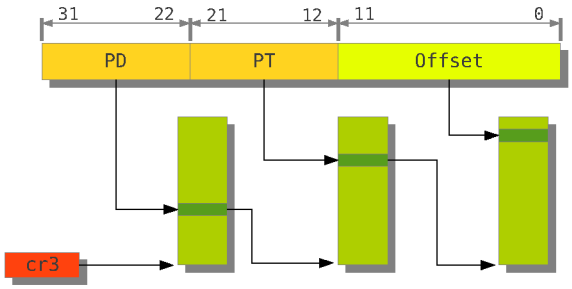


(cont'd.)



Paging in IA-32 Architecture

- Since i386, 4-kB pages can be handled.

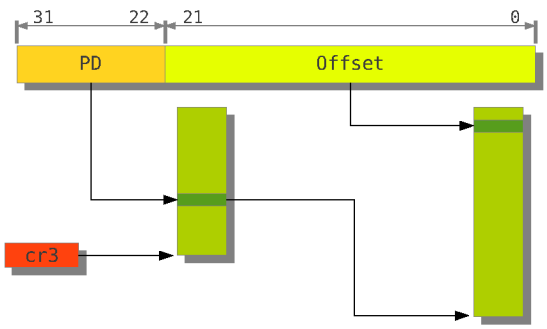


(cont'd.)

- CR3 register selects currently-executing process's page directory.
  - This is also called *Page Table Base Register* (PTBR).
- A page directory table entry selects a page table.
- A page table entry selects a 4-kB page.
- This is called multilevel paging.
  - Why multilevel ?
  - Why 10 + 10 + 12 bits ?

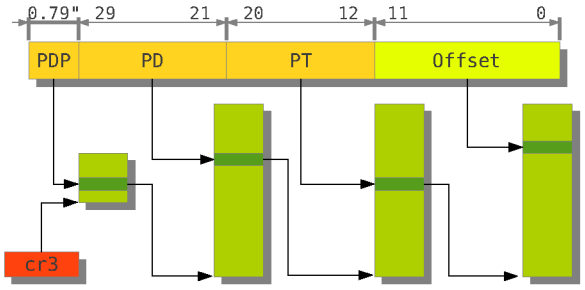
(cont'd.)

- Intel introduced *extended paging* since Pentium.
  - 4-MB page size



(cont'd.)

- Intel increased address pins to 36 pins since Pentium Pro. This allows to address up to 64 GB by the *Page Address Extension* (*PAE*).



(cont'd.)

- In PAE mode, a process is still use 32-bit logical address.
  - This means a process cannot see beyond 4 GB.
  - Only kernel could modify the page table and exploit up to 64 GB.
- There is also the *Page Size Extension* (*PSE-36*) in Pentium III and later.
  - Alternative to PAE, to address up to 64 GB.
  - Map 4-MB pages into 36-bit physical address space.

Paging in x86-64 Architecture

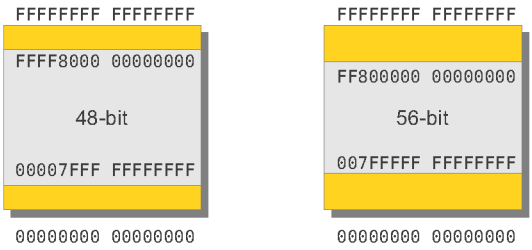
- x86-64 ~ IA-32 architecture with 64-bit memory management capability.
  - And additional 64-bit features, e.g. 64-bit GPRs.
- Currently, there are three variants
  - **AMD64** (formerly x86-64)
    - Athlon 64, Turion 64, Opteron, Sempron, Phenom
  - **Intel 64** (formerly EM64T and then IA-32e)
    - NetBurst: Celeron-D, Pentium-D/4/XE, Xeon
    - Core: Core 2, Core i7, Pentium-Dual Core, Xeon
    - Atom: 200 series
  - **VIA Nano** (formerly Isaiah)

(cont'd.)

- Theoretically, x86-64 can use 64-bit address space. However, at present and near future, we **do not** need full 64-bit address space.
  - Larger address space implies more complexity and cost of address translation.
  - Early AMD64 implementations (K8 series) provides 48-bit logical address space (256 TB) and 40-bit physical address space (1 TB).
    - Later, in K10 series, physical address space is also 48 bits.
  - AMD64 architectural limitation is 64-bit for logical address space (16 EB) and 52-bit for physical address space (4 PB).

(cont'd.)

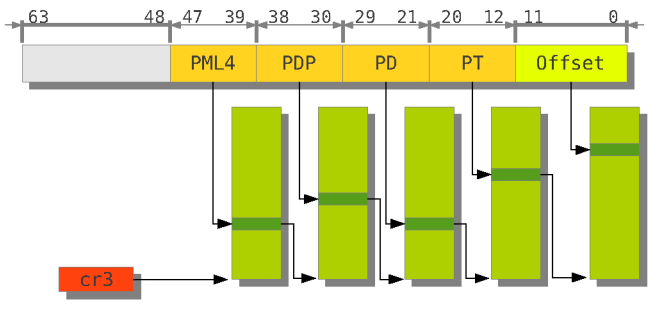
- AMD use *canonical form* address.
  - Only 48 LSBs can be used. The 48th - 63rd bit must be copies of the 47th bit.



  - This is good for splitting user/kernel address spaces.

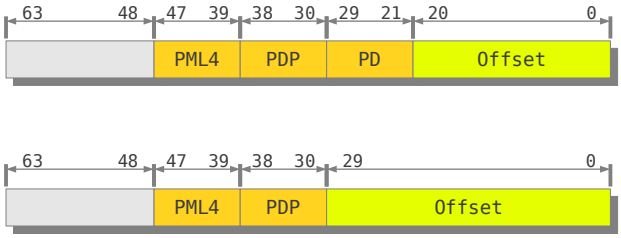
(cont'd.)

- x86-64 provides the *long mode*.
  - 4-level paging



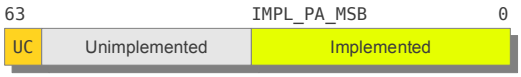
(cont'd.)

- In the long mode, the page size can also be 2 MB, and 1 GB.

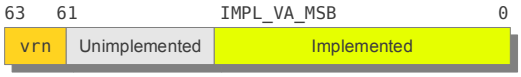


Paging in IA-64 Architecture

- IA-64 has 63-bit physical address space (8 EB)



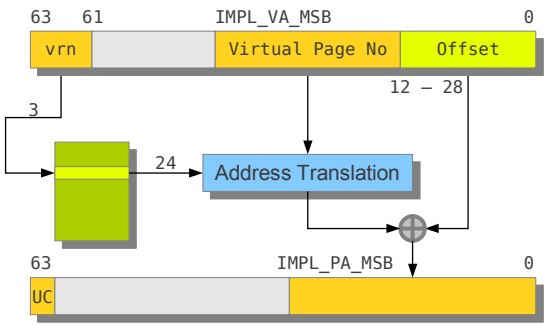
  - UC indicates an uncacheable memory.
  - Intel Itanium uses `IMPL_PA_MSB` = 43.
- And, 64-bit logical address space (16 EB).



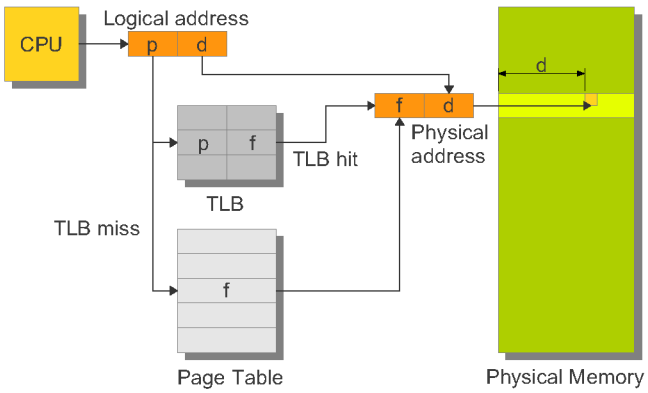
  - `vrn` indicates a virtual region number.
  - Intel Itanium has `IMPL_VA_MSB` of 50 bits.

(cont'd.)

- Each `vrn` is associated with a region register that specifies a 24-bit region identifier.



Translation Lookaside Buffers



(cont'd.)

- Translation Lookaside Buffer (TLB) is associative, high-speed memory in CPU.
  - It is essentially cache memory ~ fast access
  - Expensive hardware, size ~ 64 – 8192 entries
    - Try `cat /proc/cpuinfo` or `x86info`
- Effective Access Time
  - Lookup = ε time unit
  - Access memory = μ time unit
  - Hit ratio = α
  - EAT = $(\varepsilon + \mu)\alpha + (\varepsilon + 2\mu)(1 - \alpha)$

(cont'd.)

- Access time ? .. let's see ..
  - Intel® Core™ 2 processors
    - L1 Cache = 3 cycles
    - L2 Cache = 14 cycles
  - Intel® Itanium® processors
    - L1 Cache = 1 cycle
    - L2 Cache = 5+ cycles
    - L3 Cache = 14+ cycles
  - DDR-400 ~ min. 4 cycles @ 200 MHz
  - DDR2-800 ~ min. 10 cycles @ 200 MHz
  - DDR3-1600 ~ min. 14 cycles @ 200 MHz

Virtual Memory

- With hardware supported, a process may be swap temporarily out of (physical) memory to a backing store.
  - Backing store – a fast storage (disk), keep memory images for all users, must provide direct access to these memory images.
  - This provide rooms for other processes to load/swap in and run.
- This technique use the backing store as if it is a part of physical memory. This is sometimes called *virtual memory*.

Linux Implementation ~ Dynamic Lib.

- Shared libraries can be relocated, this causes great performance penalty.
  - Loader must find all libraries required.
  - 5 years ago, 10 libraries was rare. Now, 40 libraries are common, especially those with GUI.
- For faster library loading, Linux can employ *prelink*.
  - Introduced in 2004 by Jakub Jelinek of Red Hat.
  - Binds each library to a specific address so an executable would know exactly where the library is.
    - Eliminate those library searching.
    - Base addresses can be randomized.
  - Must prelink every time the library has changed.

(cont'd.)

- Nowadays, many Linux distros use `DT_GNU_HASH`
  - Mainly by Jakub Jelinek.
  - It is an algorithm to hash and chain symbols in the symbol table for faster lookup.
    - Done only once when building a share library.
  - ~50% improvement.
  - Ubuntu, Gentoo, Fedora, ...

Linux Implementation ~ Paging

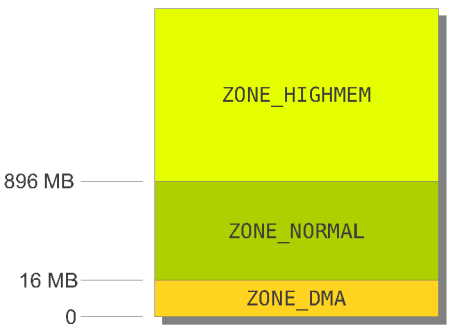
- Linux supports 32/64-bit address space, PAE, virtual memory, etc...
- Linux < 2.6.11 uses 3-level paging.
- In 2001, Intel released the first 64-bit Itanium architecture processor. Linux is the first OS to support IA-64/Itanium architecture.
  - Still use 3-level paging.
- In Oct 04, Andi Kleen and Nick Piggin introduced 4-level paging to support x86-64.
  - Merged in 2.6.11 since rc1 (~Jan 05).

(cont'd.)

- It is true that paging could avoid external fragmentation. However,
  - Contiguous frames are necessary in some cases, e.g., DMA ignores paging circuitry, buffers for DMA must be located in contiguous frames.
  - With non-contiguous frames, kernel often update page table, and CPU will flush TLBs. This cause larger EAT.
  - Kernel may access 4 MB pages, which reduces TLB misses, and thus reduces EAT.
- So, instead using paging circuitry, Linux keeps track of available frames by itself.

Linux Implementation ~ Zone

- Linux 2.6 partitions the physical memory into 3 zones:



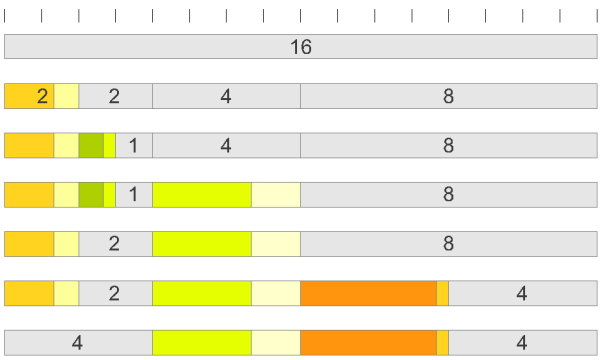
(cont'd.)

- The `ZONE_DMA` is for ISA/PCI to perform DMA.
- The `ZONE_DMA` and `ZONE_NORMAL` can be accessed through 4 GB linear mapping.
  - Kernel virtual address is 1 GB (i.e., `0xc0000000` – `0xffffffff`). This area will never be swapped out.
  - 128 MB is reserved for memory map and page tables.
  - 1 GB – 128 MB = 896 MB.
- Kernel cannot access `ZONE_HIGHMEM` directly and must be mapped to `ZONE_NORMAL`.
  - Frames in each zones are allocated by the buddy system allocation.

Buddy System Allocation

- Invented by Harry Markowitz 1963.
- Allocate blocks (or frames) in power of 2
  - If n blocks is required, it will allocate 2^i blocks, where $2^{i-1} < n < 2^i$.
- If 2^i blocks are not available, it will allocate 2^i and split it to 2 of 2^{i-1} blocks until $2^{i-1} < n < 2^i$ is met.
- After free blocks, if 2 contiguous blocks of the 2^i are available, they are merged into one 2^{i+1} block.

(cont'd.)



(cont'd.)

- Imply internal fragmentation.
  - Worst case when size is a little bit more than 2^i.
- Also imply external fragmentation.
  - But the algorithm attempt to make larger blocks available. So, it has little external fragmentation.
- Linux adopted SLAB allocation to reduce external fragmentation.
  - First introduced in Solaris 2.4 by Jeff Bonwick
  - Deallocation does not actually free blocks, but marks as unused.
  - Allocating the same size will reuse the blocks immediately without searching for one that fits.
    - Less holes, less fragmentation.

(cont'd.)

- Many kernel objects created and destroyed frequently. With SLAB, performance can be increased.
- In 2007, Christoph Lameter introduced SLUB allocation.
  - It is basically SLAB, but reduce management overhead and complex queues of the original SLAB.
  - 5 – 10 % performance increased.
  - Original SLAB allocation is still an option.

(cont'd.)

- In November 2010, Matt Mackall introduced Simple List of Block (SLOB) allocator to Linux.
  - First fit
  - Small footprint, good for embedded system
  - Internal fragmentation, not scale.
  - Used in DSLinux, a Linux distribution for Nintendo DS.