What is a page table?

And why should we care about it?

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS
SEMINAR SERIES

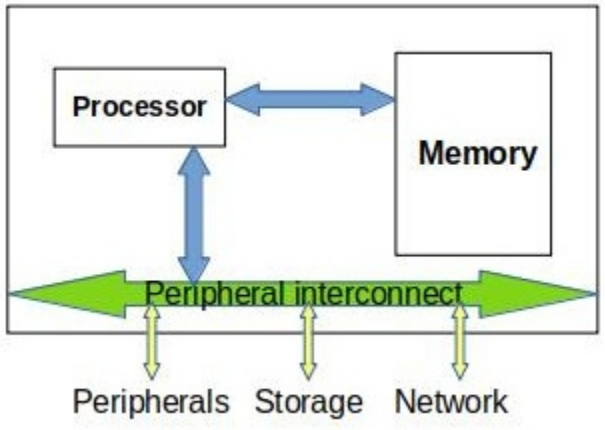
Khalid Aziz
Senior Software Engineer,
Oracle

Agenda

- Sharing resources on a computer
- Physical memory management
- Isolating memory between tasks
- Page table and MMU
- Memory allocation
- Sharing memory
- Sharing page tables

Shared Resources

- Running multiple tasks on a single computer requires sharing resources
- Core shared resources – (1) Processor, (2) Memory, (3) Network, (4) Storage
- Additional shared resources – peripherals, sensors, actuators, etc.



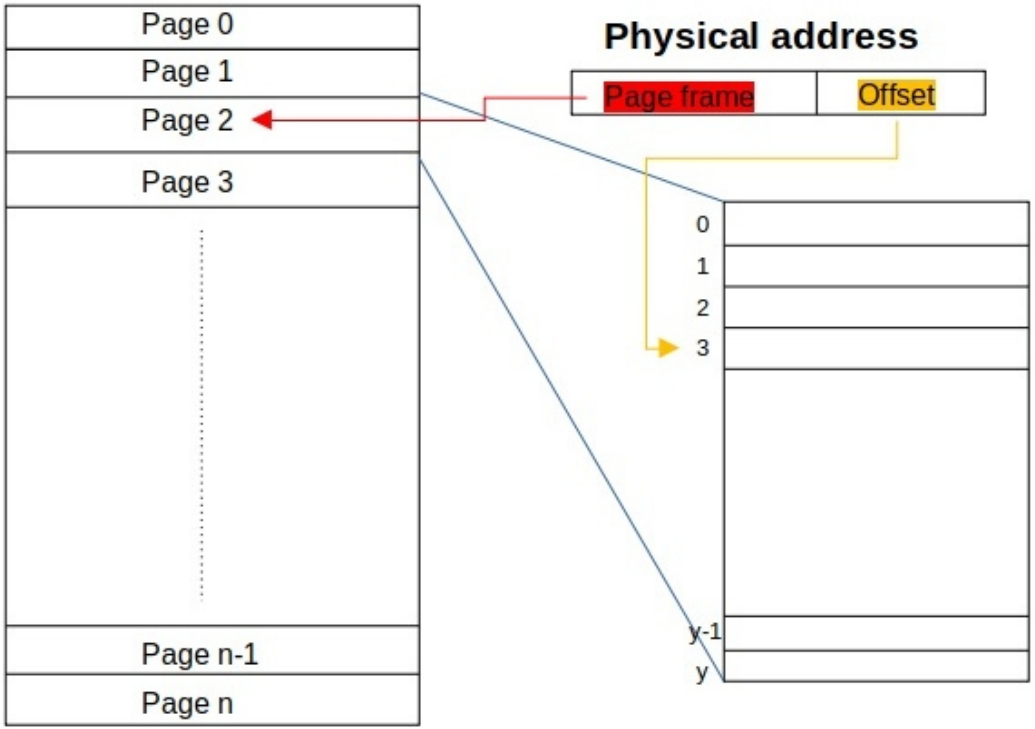
Resource Sharing Philosophy

- Allow multiple tasks to make forward progress
- Make sharing of resources fair
- Access to shared resources should be:
  - Accurate: same result every time
  - Reliable: available when needed
  - Safe: No unknown tasks/events can manipulate the resource
- Support designed-in unfairness and sharing

Physical Memory

- Physical memory on a system is typically a set of contiguous storage slots
- Each slot has an address, typically starting at 0
- For ease of physical memory management, storage slots are grouped together into a base group called a page
- Page size varies from architecture to architecture and many architectures can support multiple page sizes. On x86, base page size is 4K
- A page becomes the basic unit of allocation

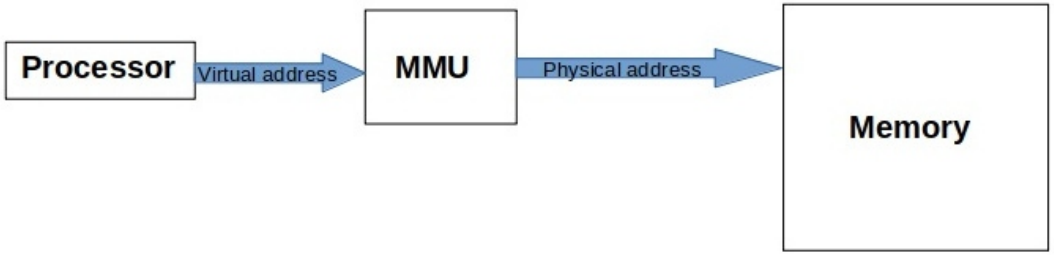
Physical Page

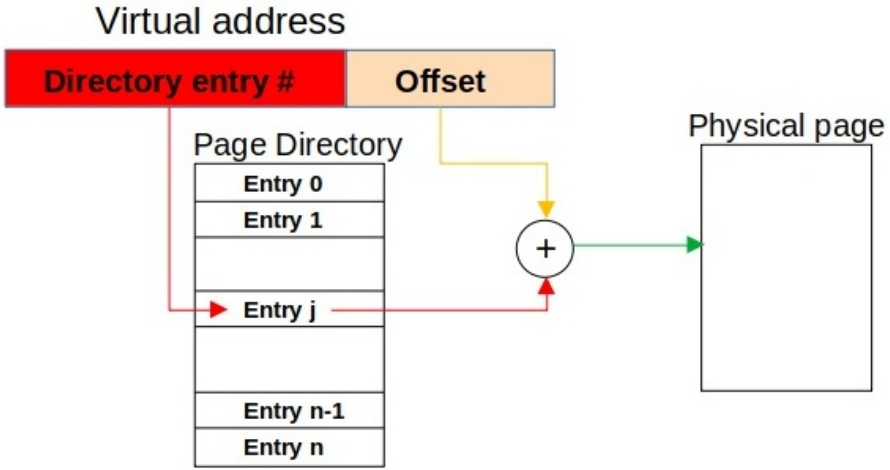


Virtual Address

- For multiple tasks to use physical addresses, each must use a unique range of physical addresses. This is not practical.
- Solve the problem by using virtual addresses instead. Virtual address refers to a storage slot on a virtual page
- Memory Management Unit (MMU) provides translation from virtual address to physical address in hardware
- A virtual page may not be backed by a physical page until needed. As a matter of fact, large address ranges of a tasks may not be backed by physical pages at any given time.

Address Translation

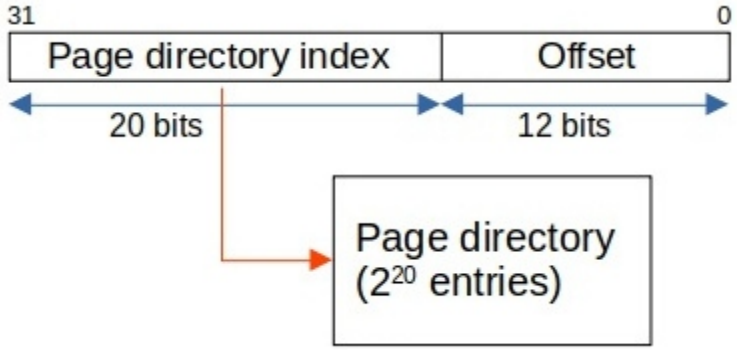




Page Table

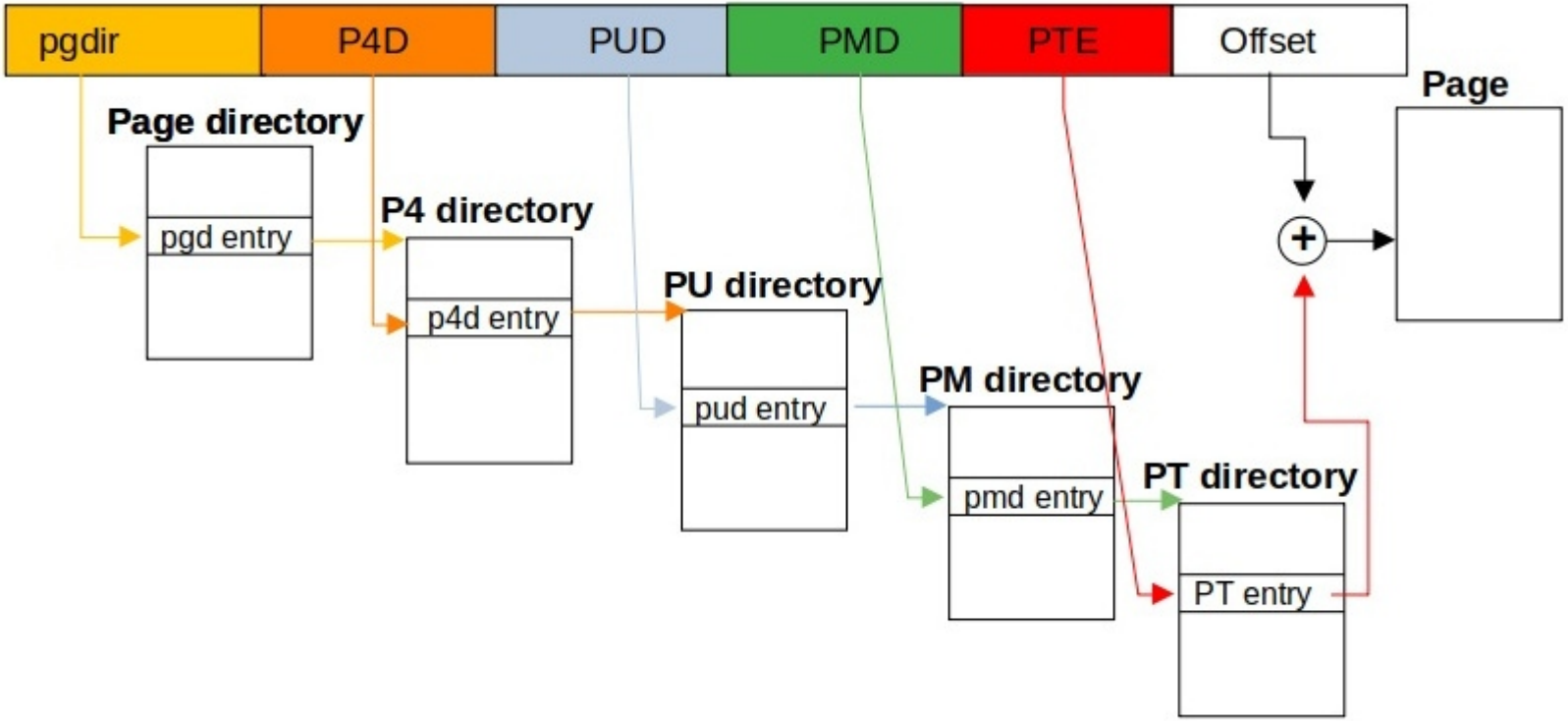
- A Page Table Entry (PTE) directs how a virtual address gets translated to a physical address
- PTE contains the physical page frame number for a corresponding virtual address and various attributes for the page that control access to that page. For example, read/write/execute permissions, present/swapped status, cacheability, possible security keys
- A collection of PTEs that cover the address range for a task constitutes the Page Table for that task

Multi-level Page Table



- 1 million entries needed for a 1-level page table
- At just 4 bytes per entry and 100s of tasks, it adds up to large numbers
- Tasks have sparse address space and not all pages are populated
- Adding multiple levels helps break address space up into smaller chunks requiring less space

5-level Page Table



Page Allocation

- Core allocator is buddy allocator. It manages pages in groups of number of pages that are powers of 2
- Order 0 page is 2^0 i.e. 1 page. Order 1 page is 2^1 i.e. 2 pages. Order 2 page is 2^2 i.e. 4 pages, and so on
- `cat /proc/buddyinfo` shows the number of free pages of various orders from order 0 – order 10:

```
Node 0, zone   Normal   4455   2454   1520    983   1629    643    238    104     42     10     33
```

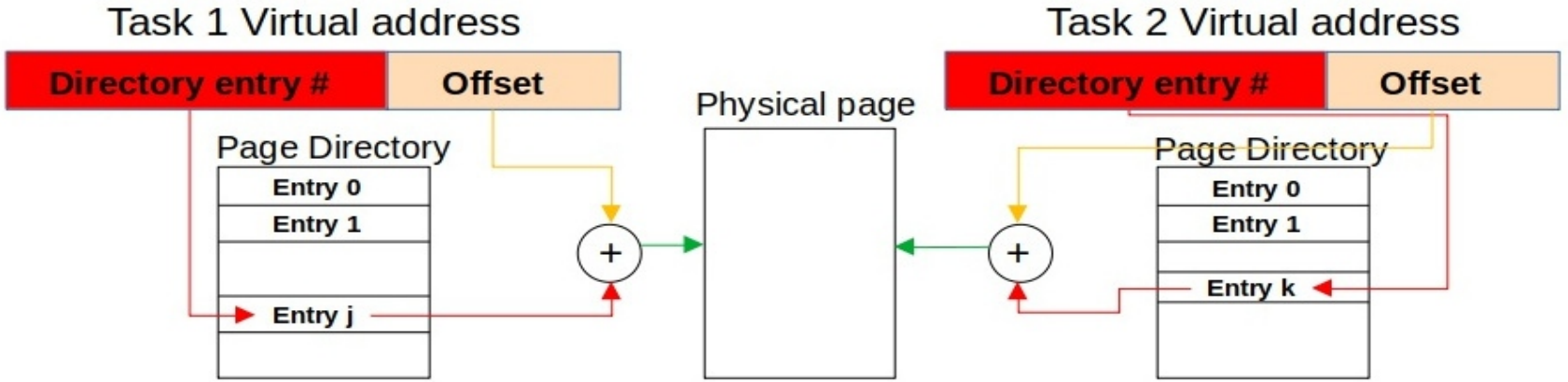
- Pages are allocated for application use as well for data caches
- Page cache holds cached data to speed up access
- Kernel will try to use any unused memory as page cache

Memory Clean up Threads

- MM subsystem employs two important kernel threads to reclaim and manage reclaimable pages – *kswapd* and *kcompactd*
- *kswapd* monitors dirty pages in the page cache and it (I) swaps out pages to swap device as needed, (ii) flush dirty disk pages, (iii) discard old pages and reclaim
- *kcompactd* scans the free pages and moves them around to create as many contiguous pages as possible. This helps create an inventory of higher order free pages

Sharing Pages

- Unique page table per task ensure address space protection for each task
- What happens when we want the same memory page to be visible to more than one task?
- Page table entry can map the same page in two address spaces



How big is a Page Table Entry

- We will use x86-64 as example here
- PTE for a 4K page is 8-bytes
- So not very big…….unless we have lots of them
- When a physical page is shared between tasks, each task has its own copy of PTE
- With 1000 tasks sharing a single page (not an uncommon scenario in large real-world deployments), we need 8000 bytes to map a single 4K page. That is twice the size of the page already, and it can get worse

Do we really need multiple copies of PTEs?

- Each task maintaining its own copy of PTE for a shared physical page gets expensive
- Threads within the same task already share PTEs and number of threads has no effect on number of PTEs needed
- Why not allow tasks to share PTEs as well?
- With the right requirements met, it is possible and can save significant amount of memory

A Proposal to Share Page Tables: mshare

- An opt-in mechanism to enable page table sharing between unrelated tasks
- A task choosing to share its pages and page tables informs the kernel which pages to share
- Any task that wants to use the shared pages and page tables maps those pages in its address space as long as it has the right permissions
- The tasks sharing page tables must be able to trust each other since pages will be shared with same permissions across all tasks

mshare Components

- mshare adds a new filesystem, *msharefs*, to facilitate sharing
- A process choosing to share its pages creates a new file on mounted msharefs, mmap's the resulting file descriptor in its address space and then maps any objects in mmap'ed address range
- Client processes open the file on msharefs and mmap the file descriptor they get in their address space. This adds the shared address pages and page tables to the client process
- A call to unlink() on the file under msharefs marks the mshare'd region for removal once no more processes have it mapped

mshare Status

- A set of RFC patches was sent upstream earlier this year
- A v1 patch series was posted in early April containing refinements from RFC review
- V1 patch series has resulted in further refinement of the API
- V2 patch series is close to being released


Questions?
ELISA
ENABLING LINUX IN SAFETY APPLICATIONS
SEMINAR SERIES