Linux Kernel Memory Subsystem and Facilities: From A Novice Programmer's Perspective

Shyam Saini
What happen after Legendary Power on button ?
After power on button (on x86)

➔ BIOS operations
➔ BIOS loads bootloader
➔ Bootloader loads Kernel
➔ Kernel bootstrap code start to run
➔ Memory is accessed in real mode with backward compatibility curse
  ◆ i.e, only small portion of the memory could be accessed
➔ Linux queries and retrieves memory from bios by detect_memory();
➔ At last kernel switches from cpu’s real mode to protected mode by go_to_protected_mode();
After power on button (on x86), Contd..

/* arch/x86/boot/main.c */

void main(void) {

    ....

    /* Detect memory layout */
    detect_memory();

    ....

    /* Do the last things and invoke protected mode */
    go_to_protected_mode();

}
Real Mode

Memory is viewed as array of offsets

Protected Mode

Application/Kernel

MMU

Virtualized view

Page Page Page Page Page Page Page Page
Memory organisation

- Physical memory is divided into different nodes, which are represented by `struct pglist_data`, typedefed as `struct pg_data_t`.
- Each memory node is split into zones list which is represented by `struct zone`.
- Each zone holds list of pages which represented by `struct page`. 
Memory Zones

For 32 bit machines

For 64 bit machines
Physical memory nodes

Each node has zone list

Each zone has pagelist

Zone list

Page list
Main Page allocators routines

```c
struct page *alloc_pages(gfp_t gfp_mask, unsigned int order);
struct page *alloc_page(gfp_t gfp_mask);
unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order);
unsigned long __get_free_page(gfp_t gfp_mask);
```
Slab Allocators and Slab cache allocators

➔ kalloc family of routines
  ◆ `kmalloc(size_t size, gfp_t flags)`
  ◆ Kmalloc, kzalloc, krealloc and kcalloc etc.

➔ Slab caches reserves a pool of free pages during initialisation and creates list of free pages
  ◆ Free list is further divided into full, partial and empty list

➔ Slab cache allocator `kmem_cache_create` is frequently used
Fragment allocators

➔ Used for random and large size allocations.
➔ Returns virtually contiguous memory and physical fragmented memory
➔ Vmalloc, vzalloc and vfree etc are routines
Boot time memory reservation

➔ Using **reserved-memory** device tree nodes (mostly used on arm and powerpc arch)
➔ Using **mem** and **memmap**, kernel boot argument
➔ Using **cma** (contiguous Memory Allocator), kernel boot argument
Physical Memory Layout

```
shyam@debian:~$ sudo cat /proc/iomem
00000000-00000fff : reserved
00001000-00009fff : System RAM
0009d000-00099fff : reserved
000a0000-000b0fff : PCI Bus 0000:00
000c0000-000cffff : Video ROM
000d0000-000d3fff : pnp 00:0a
000d8000-000dffff : pnp 00:0a
000e0000-000fffff : reserved
  000f0000-000ffffffff : System ROM
00100000-cca09fff : System RAM
cca0a000-cf89cfff : reserved
cf89d000-cf89dff : ACPI Non-volatile Storage
cf89e000-db569fff : reserved
db56a000-db599fff : ACPI Non-volatile Storage
db59a000-db5fefff : ACPI Tables
db5ff0000-df7ffffff : reserved
```
Accessing Physical Memory from user space

➔ By `mmap()` system call or library routines for memory management

➔ Accessing using `/dev/mem, devmem2/devmem` tool can be used to read and write on physical memory
  ◆ To use this, `CONFIG_STRICT_DEVMEM` kernel config option must be disabled
  ◆ This is can be mostly used by system developers for debugging drivers and other stuff, reversing engineering, etc.
Any Questions?

Feel free to reach me either at mayhs11saini@gmail.com or at shyam@amarulasolutions.com for queries or any feedback.
Thanks a lot !!