Linux Kernel Development

(Hands-on)

Kiran Divekar

About me:

- More than 16 years of experience in System programming, Linux Kernel, Wifi, embedded system.
- Worked in Multi-national companies in PMC, Nevis, Marvell, Ericsson, Calsoft.
- Vice president of GEEP.
- Delivered many talks, seminars on Linux Kernel.
- One of major contributors of Kernel 2.6.36
- Major contributor of Marvell mwifiex driver.

Linux distributions

- Set of packages for a class of applications – Desktop, server, router, embedded system.
- Desktop – kernel, shell, utilities, GUI environment, LibreOffice, compiler.
- Router – kernel, a small shell with built-in utilities (busybox), configuration scripts.
- Server – kernel, shell, utilities, server packages: apache, mysql, samba.
- Embedded:- Android
- Docker containers:- Tricky example…multiple VMs, single kernel

Kernel introduction

- Kernel located at www.kernel.org
- `uname -a`, `uname -r`
- Most common architecture x86_64. Others – x86, arm, powerpc, mips.
- Usually kernel versions are fixed by distribution makers.

Editor

```
 /* hello.c */
 # include <stdio.h>
int main()
{
    printf (“Hello world \n”);
    return 0 ;
}
```

→ gcc hello.o -o hello

Execution

./hello

---> Hello World

Power of GCC :

Not only a compiler for c. [c, c++, fortran, and binary ojbects]

Acts as linker as well.

Supports all CPU platforms.

Linux OS architecture

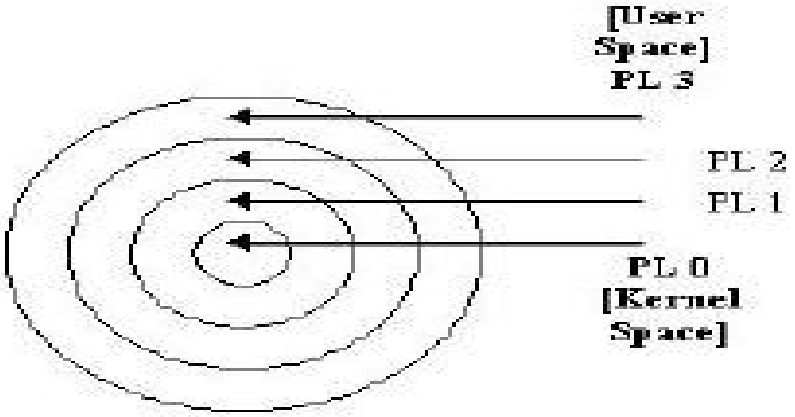
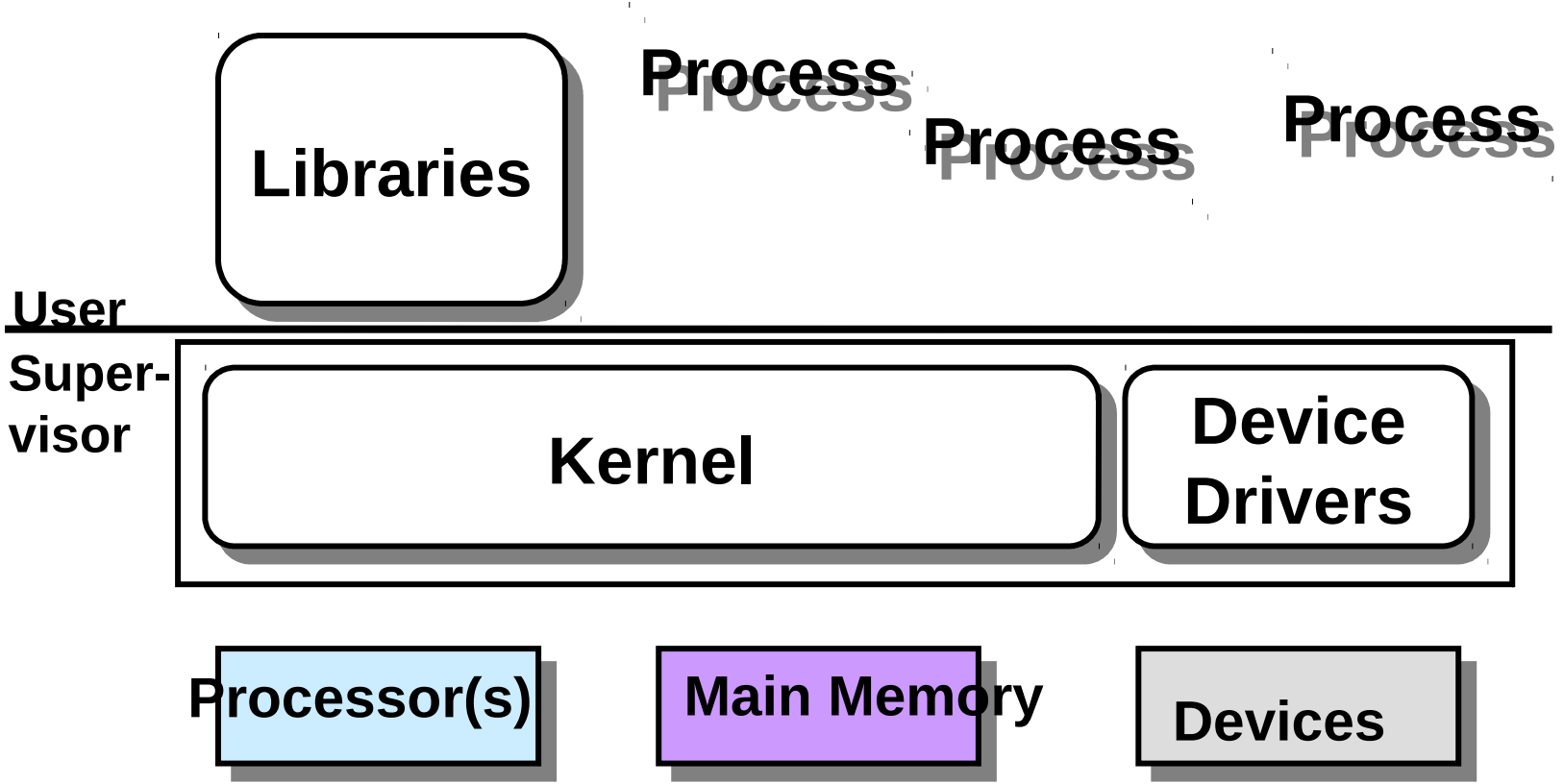


Figure 2.1: x86 Privilege Levels

Architecture



OS architecture

User Space :

- Less Privileged.
- All Processes exist.
- Swappable.

Kernel Space :

- More Privileged.
- Kernel and drivers exist.
- Non-Swappable

Compilation, Linking

Application Programs :

- Compiled in user space.
- Linked in user space.

Kernel Modules :

- Compiled in user space.
- Linked in kernel space.

Kernel source hierarchy

- arch – architecture specific
- kernel – core kernel
- mm – memory management
- net – networking stack (tcp/ip)
- fs – filesystems
- drivers:- device drivers
- block – block device driver framework

Kernel drivers

- scsi – scsi framework and drivers
- net/ethernet, net/wifi
- usb
- Block – replication(drbd)
- md – device mapper(dm), raid.

Kernel source browsing

- http://lxr.linux.no http://lxr.free-electrons.com/
- cscope – command line tool.
- vim/gvim/gedit.

Kernel compilation

- Different processes for modifying, building and releasing a kernel – ubuntu: dpkg, fedora: rpm, Openwrt: menuconfig.
- We'll look at compilation of a generic kernel: common.
- Modify-make-install-reboot-test cycle for base kernel.
- Modify-make-insmod-test cycle for modules.

Compile and load

- make menuconfig
- make
- sudo make modules_install
- sudo make install
- Reboot
- Choose new kernel at grub prompt.

Check config

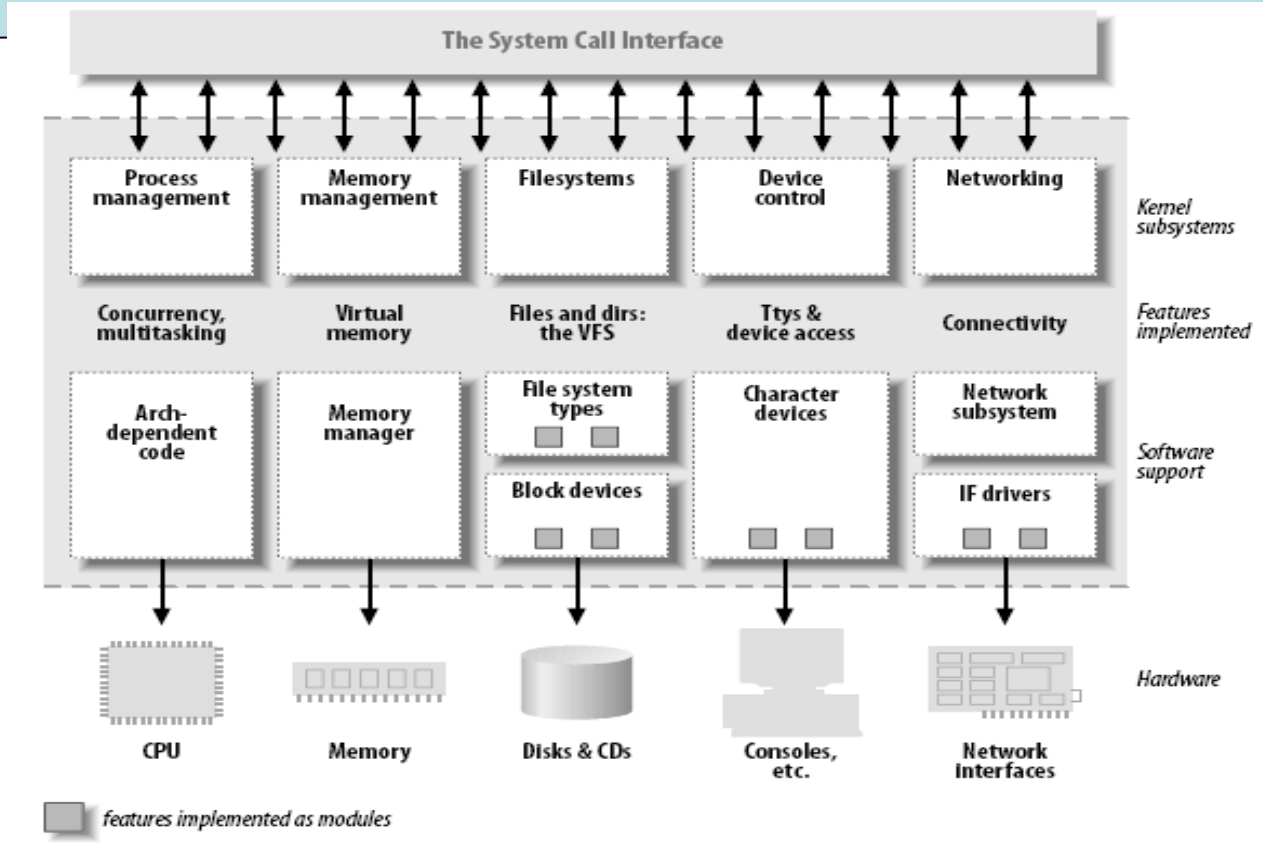
- uname -r
- cat /proc/cmdline, /proc/filesystems, /proc/devices, /proc/cpuinfo
- cat /sys/block/sda/size
- lspci (-v, -vv)

Lets do it.

Kernel Modules

- Device drivers
  - Black boxes to hide details of hardware devices
  - Use standardized calls
    - Independent of the specific driver
  - Main role
    - Map standard calls to device-specific operations
  - Can be developed separately from the rest of the kernel
    - Plugged in at runtime when needed

Splitting the Kernel



Loadable Modules

- The ability to add and remove kernel features at runtime
- Each unit of extension is called a *module*
- Use **`insmod`** program to add a kernel module
- Use **`rmmod`** program to remove a kernel module
-
- See: modinfo, depmod

Classes of Devices and Modules

- Character devices
- Block devices
- Network devices

Loadable Modules

Can include only those header files present under kernel source/headers:

Include/linux

Include/asm

of the kernel source tree (/usr/src/linux)

Kernel.h

Defines printk and other required functions

Module.h

Everything else needed for module

Resolves Kernel Module versioning issues

Loadable Modules

insmod

Command can be used to insert a driver module in the running kernel.

Beware of the kernel version, Linux does no allows modules compiled for different versions to be inserted.

rmmod

Command can be used to remove the driver module from the running kernel.

Module: testappHello3x

- Read source file.
- make -C <path_to_kernel_src> M=$PWD
- Run make.
- sudo insmod hello.ko
- Check dmesg for “Module initialization....” msg.
- sudo rmmod hello
- Check dmesg for “Module cleanup...” msg.

Character Devices

- /proc/devices
- Abstraction: a stream of bytes
  - Examples
    - Text console (**`/dev/console`**)
    - Serial ports (**`/dev/ttyS0`**)
  - Usually supports **`open`**, **`close`**, **`read`**, **`write`**
  - Accessed sequentially (in most cases)
  - Might not support file seeks
  - Exception: frame grabbers
    - Can access acquired image using **`mmap`** or **`lseek`**

Block Devices

- /proc/devices

- Abstraction: array of storage blocks
- However, applications can access a block device in bytes
  - Block and char devices differ only at the kernel level
  - A block device can host a file system

Network Devices

- ifconfig -a
- Abstraction: data packets
- Send and receive packets
  - Do not know about individual connections
- Have unique names (e.g., **`eth0`**)
  - Not in the file system
  - Support protocols and streams related to packet transmission (i.e., no **`read`** and **`write`**)

System Call

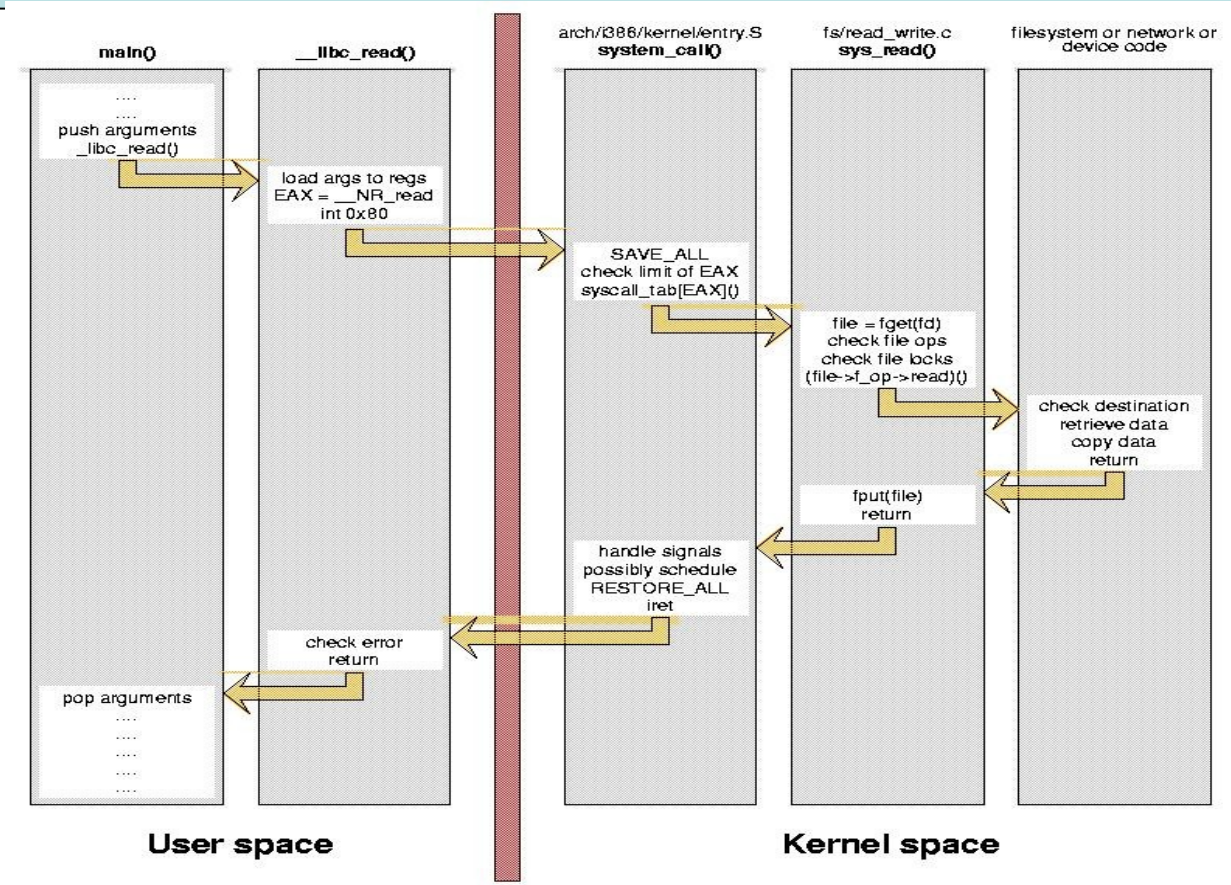
Invoked by executing int 0x80

- CPU switches to kernel mode and executes a kernel function
- Calling process passes system call number in eax register (on Intel processors)

Syscall handler is responsible for

- Saving reg. On kernel stack
- Invoking sys call service routine
- Exit by calling ret_from_sys_call ()

System call


main()
__libc_read()
arch/i386/kernel/entry.S
system_call()
fs/read_write.c
sys_read()
filesystem or network or device code
....
....
push arguments
_libc_read()
load args to regs
EAX = __NR_read
int 0x80
SAVE_ALL
check limit of EAX
syscall_tab[EAX]()
file = fget(fd)
check file ops
check file locks
(file->f_op->read)()
check destination
retrieve data
copy data
return
fput(file)
return
handle signals
possibly schedule
RESTORE_ALL
iret
check error
return
pop arguments
....
....
....
....
....
User space
Kernel space

Character driver file operations

File operations: open, release, read, write, ioctl <linux/fs.h>

Things to understand:

- Cdev: increasing the size of the dev_t device number type
- macros (defined in <linux/kdev_t.h>

- major = MAJOR(dev_t dev);
- minor = MINOR(dev_t dev);
- dev = MKDEV(int major, int minor);

Global data in test programs

Used for driver local storage.

Used to store hardware information.

```
struct kdev = {
   /* local bookkeeping information */
   /* hardware specific information */
   /* linked list pointer for minors */
};
```

Global data in test programs

init_module : Allocate the structure.

open driver function : set up file->private_data.

System calls (e.g read) : use file->private_data to access device memory, local data.

release driver funtion : file->private_data = NULL.

Cleanup_module : Free the structure.

Global data in test programs

<asm/uaccess.h>

copy_from_user(kptr, uptr, size) ;
get_user(kptr, uptr) ;

copy_to_user(uptr, kptr , size) ;
put_user(kptr, uptr) ;

Read file operation

- Validate function arguments.
- Wait for device data to be ready using interruptible_sleep_on
- Read from the device file.
- Use put_user function or using copy_to_user

Write file operation

- Validate function arguments.
- Write to device file from the user buffer.
- The user buffer is accessed using get_user function or using copy_from_user
- Wake up all waiting processes using wake_up_interruptible.

Lets do it.

Creating Processes

- Resources owned by a parent process are duplicated when a child process is created.

–*It is slow* to copy whole address space of parent and *unnecessary*, if child (typically) immediately calls **execve(),** thereby replacing contents of duplicate address space.

- Cost savers:

–*Copy on write* – parent and child share pages that are read; when either writes to a page, a new copy is made for the writing process.

–*Lightweight processes* – parent & child share page tables (user-level address spaces), and open file descriptors.

Process Descriptors

- The kernel maintains info about each process in a process descriptor, of type **`task_struct`**.

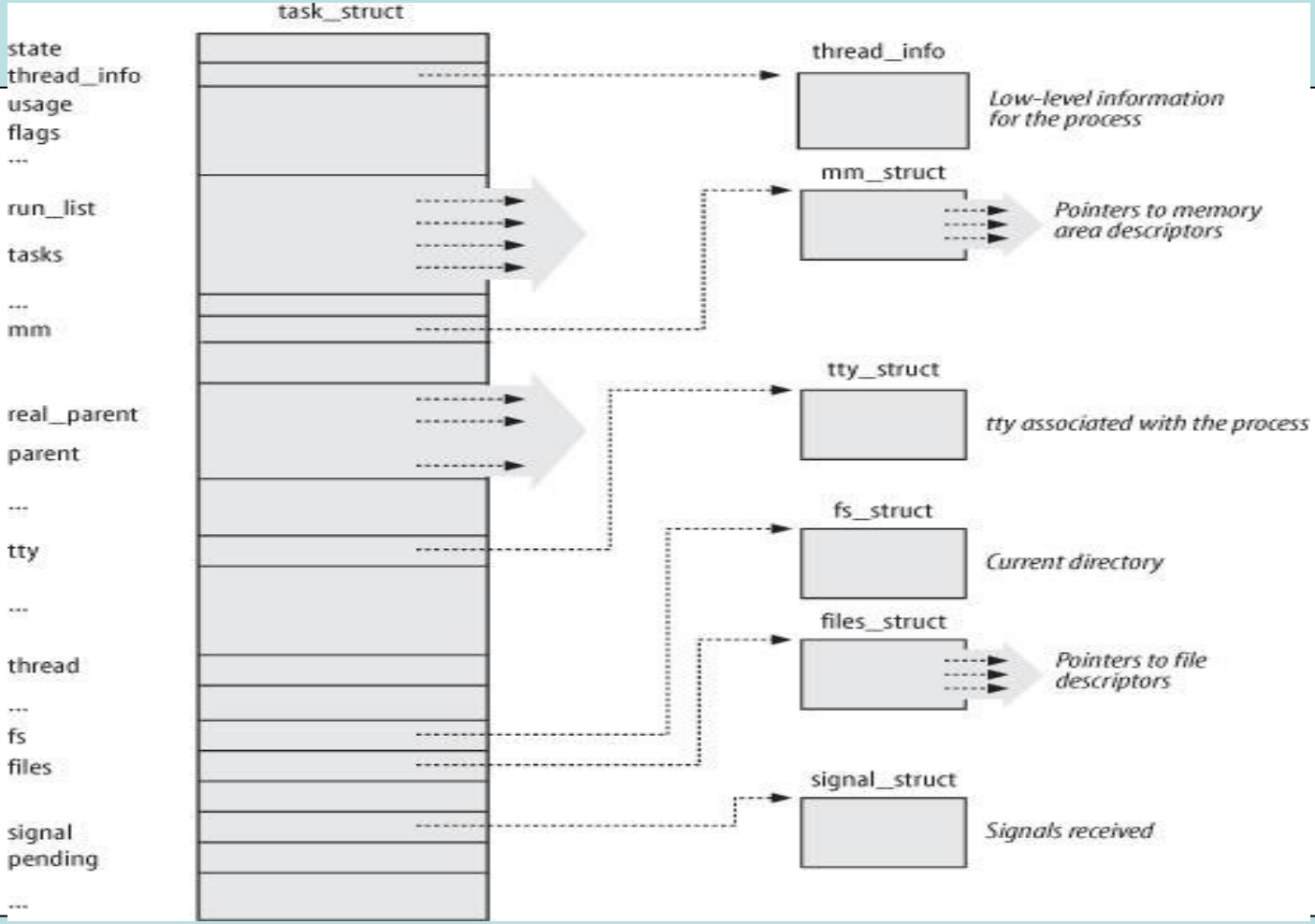
  See **`include/linux/sched.h`**

- Each process descriptor contains info such as run-state of process, address space, list of open files, process priority etc…

```
struct task_struct {
     volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    unsigned long flags; /* per process flags */
     mm_segment_t addr_limit; /* thread address space:
                                 0-0xBFFFFFFF for user-thead
                                 0-0xFFFFFFFF for kernel-thread */
   struct exec_domain *exec_domain;
  long need_resched;
  long counter;
  long priority;
   /* SMP and runqueue state */
       struct task_struct *next_task, *prev_task;
       struct task_struct *next_run,  *prev_run;
    ...
  /* task state */
  /* limits */
   /* file system info */
  /* ipc stuff */
   /* tss for this task */
    /* filesystem information */
    /* open file information */
    /* memory management info */
   /* signal handlers */
     ...
};
```

Contents of process descriptor

Linux Process Descriptor



Process State

- **`state`** values:
  - **`TASK_RUNNING`** (executing on CPU or runnable).
  - **`TASK_INTERRUPTIBLE`** (waiting on a condition: interrupts, signals and releasing resources may “wake” process).
  - **`TASK_UNINTERRUPTIBLE`** (Sleeping process cannot be woken by a signal).
  - **`TASK_STOPPED`** (stopped process e.g., by a debugger).
  - **`TASK_ZOMBIE`** (terminated before waiting for parent).

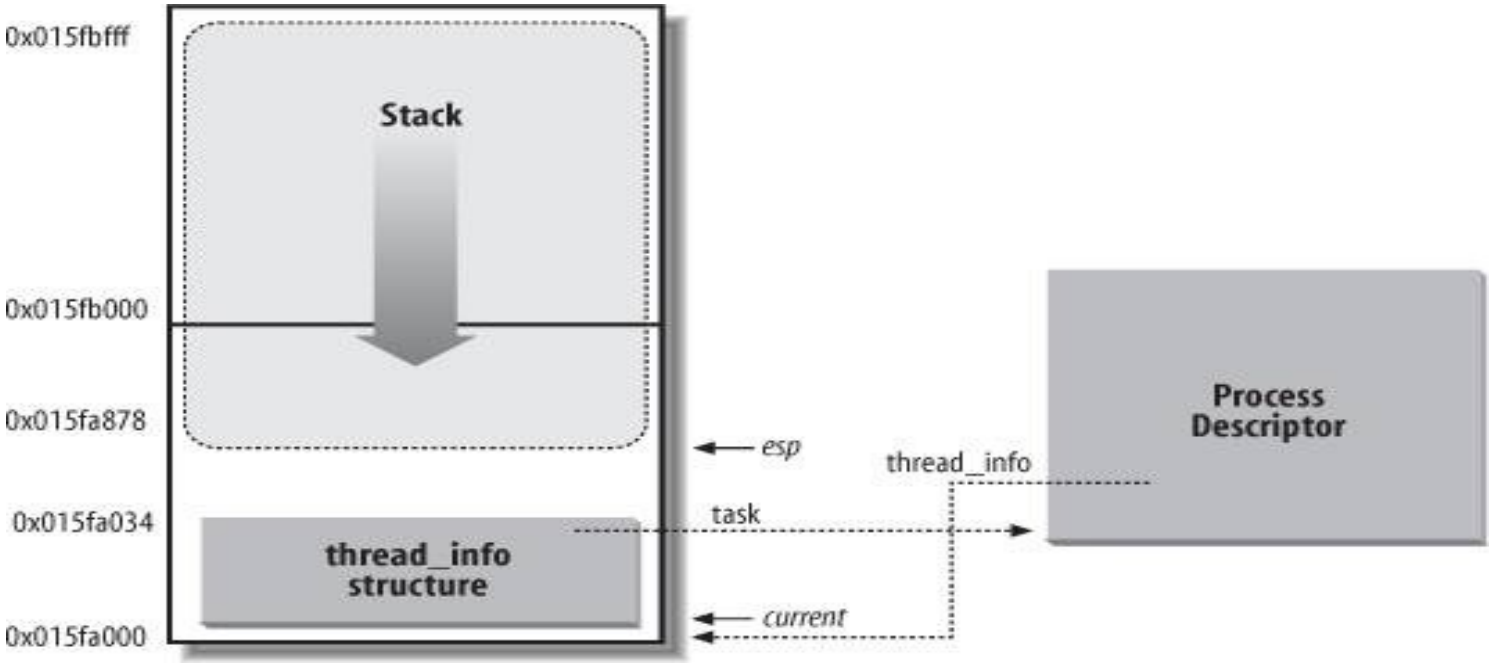
Process Identification

- Each process, or independently scheduled execution context, has its own process descriptor.
- Process descriptor addresses are used to identify processes.
  - Process ids (or **PID**s) are 32-bit numbers, also used to identify processes.
  - For compatibility with traditional UNIX systems, LINUX uses PIDs in range 0..32767.

Process Descriptor Storage

- Processes are *dynamic*, so descriptors are kept in dynamic memory.
- An 8KB memory area is allocated for each process, to hold process descriptor *and* kernel mode process stack.
  - **Advantage**: Process descriptor pointer of `current` (running) process can be accessed quickly from stack pointer.
  - 8KB memory area = 2^{13} bytes.
  - Process descriptor pointer = **esp** with lower 13 bits masked.

thread_info Structure



The Process List

- The *process list* (of all processes in system) is a doubly-linked list.
  - **`prev_task`** & **`next_task`** fields of process descriptor are used to build list.
  - **`init_task`** (i.e., swapper) descriptor is at head of list.
- **`prev_task`** field of **`init_task`** points to process descriptor inserted *last* in the list.
  - **`for_each_task()`** macro scans whole list.

Wait queues

- Required when processes wait for a resource and enter TASK_INTERRUPTIBLE state.
- Wait queues are defined per resource.
- Kernel keeps a queue of kernel tasks waiting for a resource

```
struct __wait_queue {
    unsigned int flags;
    struct task_struct * task;
    struct list_head task_list;
};
```

Wait Queues

- **`TASK_(UN)INTERRUPTIBLE`** processes are grouped into classes that correspond to specific events.
  - –e.g., timer expiration, resource now available.
  - –There is a separate wait queue for each class / event.
  - –Processes are “woken up” when the specific event occurs.

Lets do it.

Execution Context

- PROCESS CONTEXT
- INTERRUPT CONTEXT
- SOFTIRQ CONTEXT

Overview

- The Hardware Part
  - Interrupts and Exceptions
  - Exception Types and Handling
  - Interrupt Request Lines (IRQs)
  - Programmable Interrupt Controllers (PIC)
  - Interrupt Descriptor Table (IDT)
  - Hardware Dispatching of Interrupts
- The Software Part
  - Nested Execution
  - Kernel Stacks
  - SoftIRQs, Tasklets
  - Work Queues
  - Threaded Interrupts

Interrupt architecture (top half & bottom half)

- Interrupts remain disabled for a very less amount of time.
- It minimizes overall interrupt latency.
- The interrupt controller chip is told to disable the specific IRQ be serviced while the kernel is executing the top half.
- Bottom half consists of things that don t need to be done on ever single interrupt.

Interrupt architecture (top half)

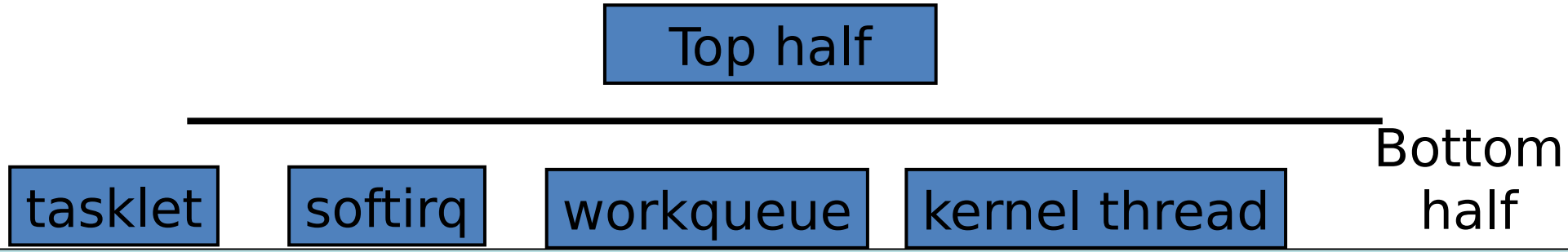
- Validate function arguments.
- Check the dev_id,for shared interrupts.
- Perform bookkeeping, like storing interrupt information in driver memory.
- Queue the task to immediate task queue.
- [queue_task(&task_queue, &tq_immediate);]

Interrupt architecture (bottom half)

- Validate function arguments.
- Read the driver memory for stored interrupt information.
- Take appropriate action as per the interrupt source

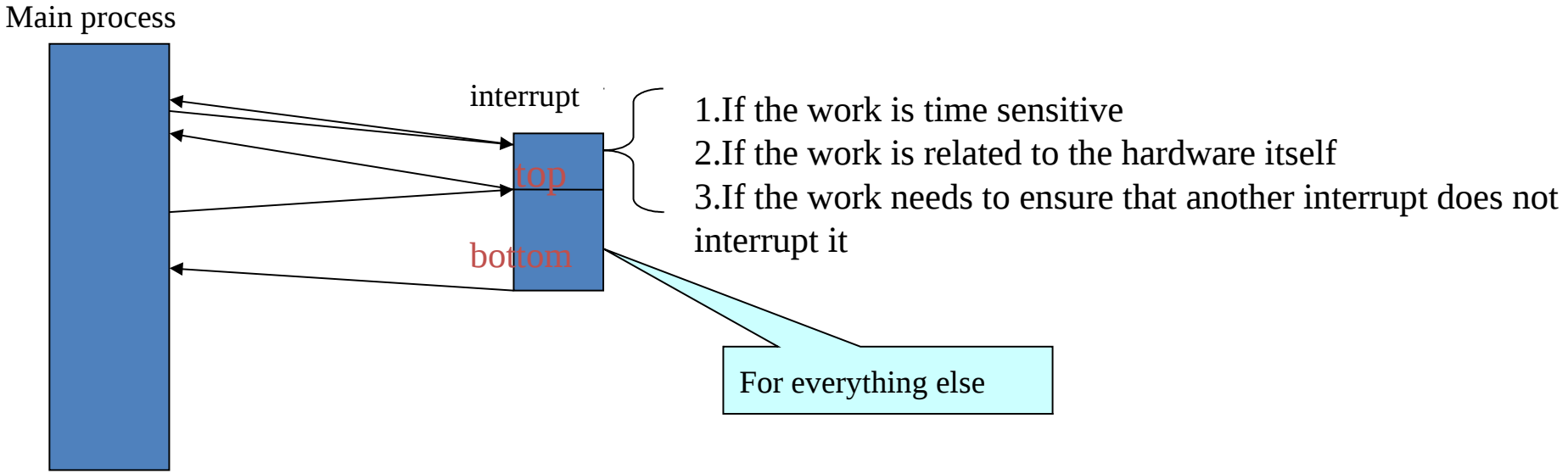
Bottom Half: Do it Later!

- Mechanisms to defer work to later:
  - *softirqs*
  - *tasklets* (built on top of softirqs)
  - *work queues*
  - *kernel threads*
- All can be interrupted



Bottom Halves

- The job of bottom halves is to perform any interrupt-related work not performed by the interrupt handler itself.



Interrupt architecture

- Interrupt handlers run asynchronously and thus interrupt other potentially important code, including other interrupt handlers. Therefore, to avoid stalling the interrupted code for too long, interrupt handlers need to run as quickly as possible.
- We hope to limit the amount of work you perform in an interrupt handler, so handle interrupt time is the less the better.
- So in an interrupt handler only do some necessary work, and defer some of the work until later
- Three methods : softirqs, tasklets, and work queues
  - All part of Kernel.
- Device drivers use tasklets/work queues.

Softirqs

- Softirqs are raised at the end of interrupt handler. They run as BH with interrupt enabled.
  - They are fixed in number.
  - Used by kernel code, but NOT available for driver writers.
  - Same type of softirq can run on multiple processors.
  - e.g. HiPrio, Timer, Tx, Rx, block devices, Tasklet softirq, scheduler, HRTimer, RCU
  - **Ksoftirqd** handles throttling.

When Softirqs Run?

- Run at various points by the kernel:
  - After system calls
  - After exceptions
  - After interrupts (top halves/IRQs, including the timer intr)
  - When the scheduler runs ksoftirqd
- Softirq routines can be executed simultaneously on multiple CPUs:
  - Code must be re-entrant
  - Code must do its own locking as needed
- Hardware interrupts are enabled when softirqs run.

Task queues

- Executed in the context of HiPrio or tasklet softirq.
- Same type of tasklet can not run simultaneously on multiple processors.
- tasklet_schedule() or tasklet_hi_schedule()
  - tq timer The timer task queue, run on each timer interrupt.
  - tq scheduler The scheduler task queue, consumed by the scheduler.
  - tq immediate bottom half IMMEDIATE BH task queue
  - tq disk Used by low level block device access.

Work queues

- Work queues are used when defered work needs to sleep.e.g.

Continous while(1) loop

  - schedule_work
  - schedule_delayed_work
  - flush_delayed_work

Lets do it.

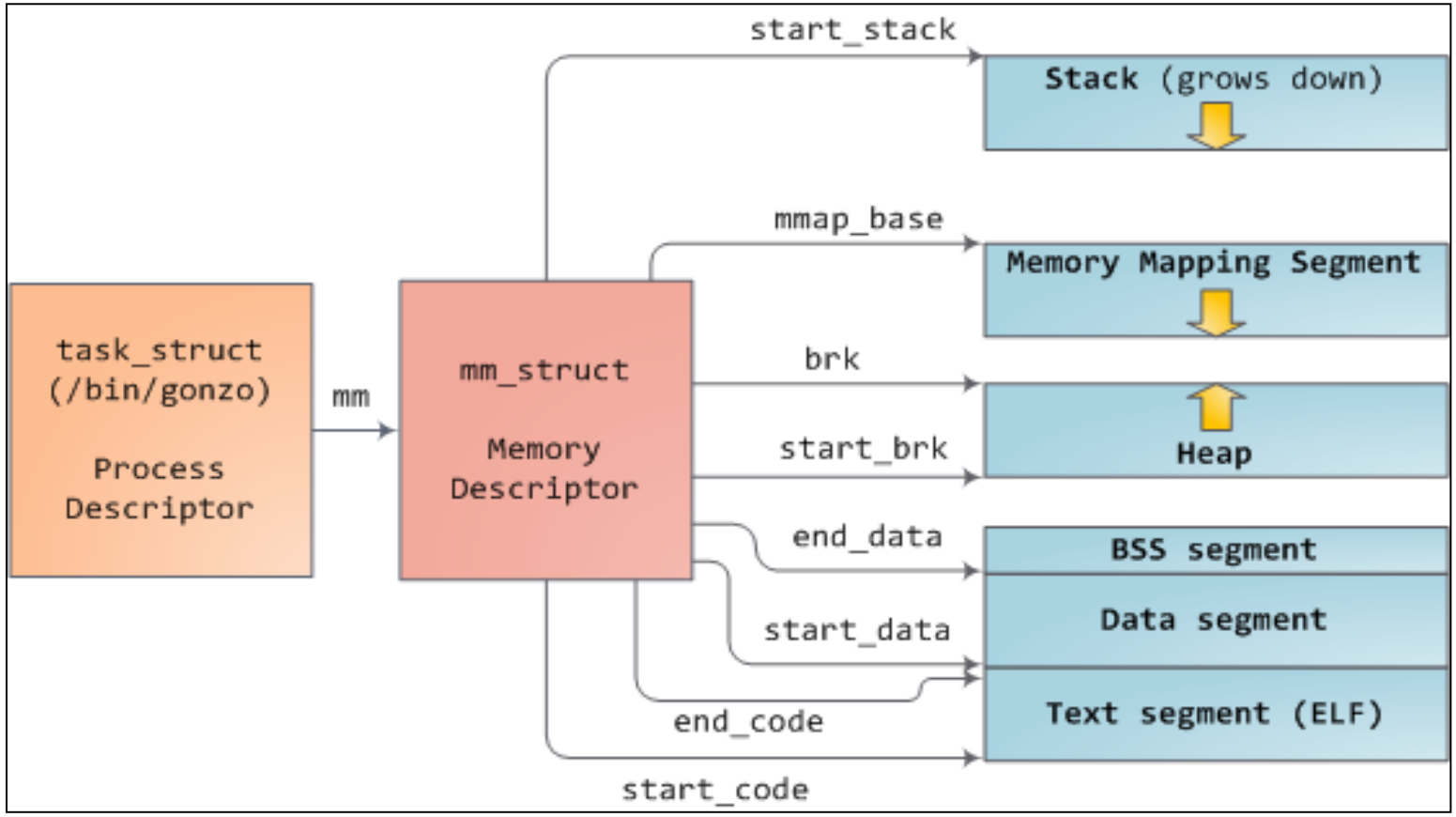
Kernel Threads

Some (background) system processes run only in kernel mode. e.g., flushing disk caches, swapping out unused page frames. (bdflushd)

–Can use *kernel threads* for these tasks.

- Kernel threads only execute kernel functions – normal processes execute these fns via syscalls.
- Kernel threads only execute in kernel mode as opposed to normal processes that switch between kernel and user modes.
- Kernel threads use linear addresses greater than PAGE_OFFSET – normal processes can access 4GB range of linear addresses.

Address space C program



Address space (mm_struct)

mm_struct defined in the process descriptor. (in linux/sched.h)

This is duplicated if CLONE_VM is specified on forking.

```
struct mm_struct {
int count;  // no. of processes sharing this descriptor
pgd_t *pgd;   //page directory ptr
unsigned long start_code, end_code;
unsigned long start_data, end_data;
unsigned long start_brk, brk;
unsigned long start_stack;
...
unsigned long rss;  // no. of pages resident in memory
unsigned long def_flags;  // status to use when mem regions are created
struct vm_area_struct *mmap;  // ptr to first region desc.
struct vm_area_struct *mmap_avl;  // faster search of region desc.
}
```

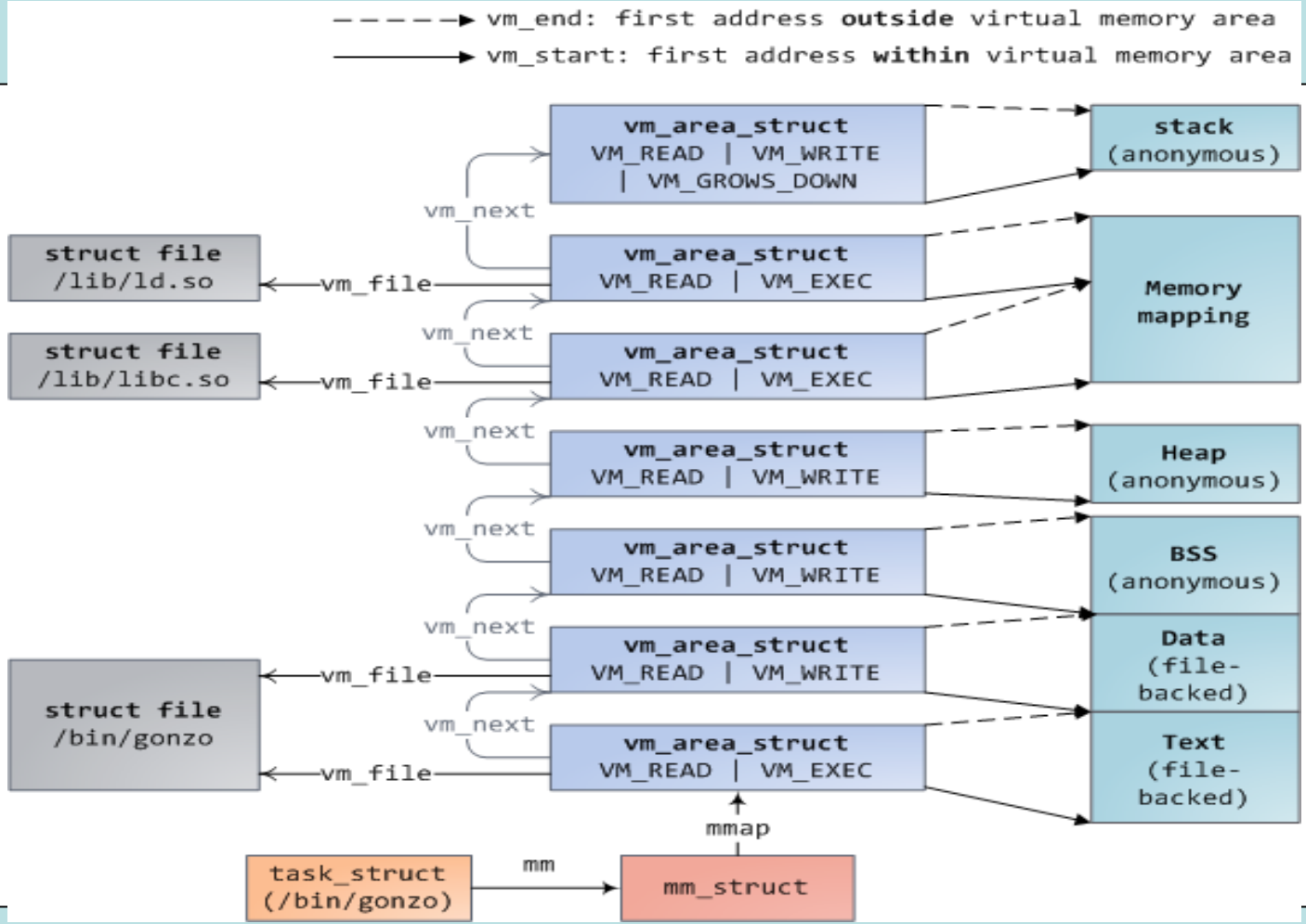
Virtual address space

- The User Segment
- Virtual Memory Areas
- The System Call ***brk***
- Mapping Functions
- The Kernel Segment
- Static/Dynamic Memory Allocation in the Kernel Segment

Address space (vma_struct)

```
struct vm_area_struct {
        struct mm_struct * vm_mm;       /* The address space we belong to.
*/
        unsigned long vm_start;         /* Our start address within vm_mm. */
        unsigned long vm_end;           /* First byte after our end address
                                           within vm_mm. */
        /* linked list of VM areas per task, sorted by address */
        struct vm_area_struct *vm_next;
        struct file *vm_file;
        ....
}
```

Virtual address space (/proc/%d/maps)



sudo pmap -X

Address: start address of map

Kbytes: size of map in kilobytes

RSS: resident set size in kilobytes

Dirty: dirty pages (both shared and private) in kilobytes

Mode: permissions on map: read, write, execute, shared, private (copy on write)

Mapping: file backing the map, or '[anon]' for allocated memory, or '[stack]' for the program stack

Offset: offset into the file

Device: device name (major:minor)

Lets do it.

Kernel Memory Zones

Zones of Memory (32-bit)

- Zone DMA : 0-16MB
- Zone Normal : 16 – 896 MB
- Zone Highmem : Above 896 MB

Zones of Memory (64-bit) /proc/buddyinfo, zoneinfo

- Zone DMA, DMA32 :
- Zone Normal :

Types of Memory

- Physically contiguous and Virtually contiguous

Physically Contiguous memory

- Custom slabs </proc/slabinfo> created by the kernel.
- Size is power of 2 upto 128KB
- Header file having declaration : include/linux/slab.h
- static __always_inline void *kmalloc(size_t size, gfp_t flags)
    - Physical memory for device drivers
    - DMA-able memory.

The Memory allocation flags

Header file having declaration : include/linux/gfp.h

- #define GFP_ATOMIC (__GFP_HIGH)
- #define GFP_NOIO (__GFP_WAIT)
- #define GFP_NOFS (__GFP_WAIT | __GFP_IO)
- #define GFP_KERNEL (__GFP_WAIT | __GFP_IO | __GFP_FS)

Freeing Physically contiguous memory

- Header file having declaration: include/linux/slab.h
- void kfree(const void *objp);
  - Internals:-
    - kmem_cache_create
    - kmem_cache_alloc
    - kmem_cache_free
    - kmem_cache_destroy

Allocating Virtual contiguous memory

- Memory is not physically contiguous
- Addresses are in kernel address space
- Idea of continuity is provided by stitching the page tables to create illusion.
- Header file having declaration : include/linux/vmalloc.h
- void *vmalloc(unsigned long size)
    - Performance hit at MMU.

Freeing Virtually contiguous memory

- Header file having declaration : include/linux/vmalloc.h

- void vfree(const void *addr);

The /proc filesystem

- A pseudo file system
- Real time, resides in the virtual memory
- Tracks the processes running on the machine and the state of the system
- A new /proc file system is created every time your Linux machine reboots
- Highly dynamic. The size of the proc directory is 0 and the last time of modification is the last bootup time.

Other features

- /proc file system doesn't exist on any particular media.
- The contents of the /proc file system can be read by anyone who has the requisite permissions.
- Certain parts of the /proc file system can be read only by the owner of the process and of course root. (and some not even by root!!)
- The contents of the /proc are used by many utilities which grab the data from the particular /proc directory and display it.

  eg : top, ps, lspci ,dmesg etc

Tweak kernel parameters

- /proc/sys : Making changes in this directory enables you to make real time changes to certain kernel parameters.

  eg : /proc/sys/net/ipv4/ip_forward
- It has default value of "0" which can be seen using 'cat'.
- This can be changed in real time by just changing the value stored in this file from "0" to "1", thus allowing IP forwarding

Files in /proc

- buddyinfo
- cmdline
- cpuinfo
- devices
- filesystems
- interrupts
- iomem
- ioports
- meminfo
- misc
- modules
- mounts
- pci
- self
- slabinfo
- stat
- swaps
- uptime
- version
- vmstat

/proc/sys subdirectories

- /proc/sys/dev : provides parameters for particular devices on the system
  - cdrom/info : many important CD-ROM parameters
- /proc/sys/fs
- /proc/sys/net: networking
- /proc/sys/vm: virtual memory
- /proc/sys/kernel
  - acct — Controls the suspension of process accounting based on the percentage of free space available on the filesystem containing the log

/proc File System Entries

- Header file: #include <linux/proc_fs.h>

- struct proc_dir_entry* **create_proc_entry**(const char* *name*, mode_t *mode*, struct proc_dir_entry* *parent*);
  - This function creates a regular file with the name *name*, the mode *mode* in the directory *parent*

- void **remove_proc_entry**(const char* *name*, struct proc_dir_entry* *parent*);
  - This function removes the proc entry

/sysfs : Linux Device Model

- **/sys/block :** Contains known block devices
- **/sys/bus :** Contains all registered buses.
- **/sys/class :** Contains Devices
- **/sys/device :** All devices known by the kernel organized by the bus that they connect to.
- **/sys/firmware :** Contains firmware files for some devices
- **/sys/fs :** Contains files to control filesystems
- **/sys/kernel :** Various kernel related files
- **/sys/module :** Loaded kernel modules. Each module is represented by a directory of the same name
- **/sys/power :** Various files to handle power state of system

Device Model Functionality

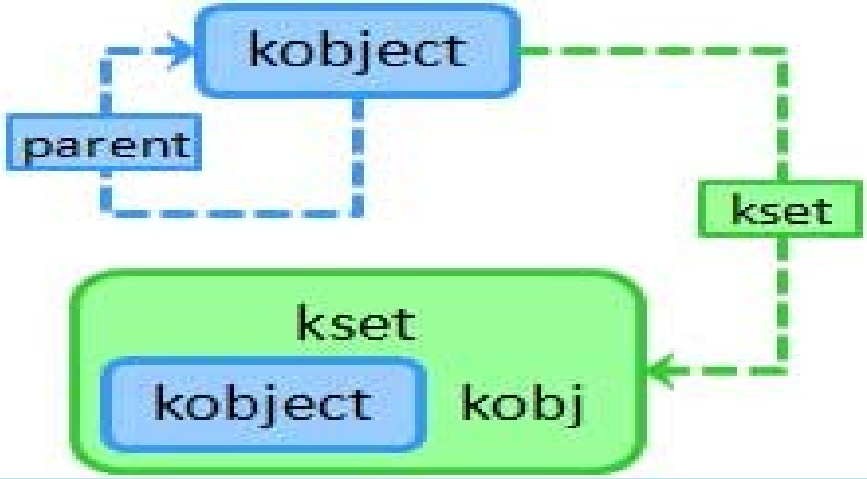
- Demands of newer systems with more complicated topologies and power management motivated construction of the Linux Unified Device Model
- Power management and system shutdown
  - Knowing the ordering of when to shut down components.
  - E.g., shut down USB mouse before USB controller
- Communication with user space
  - Sysfs virtual file system tightly tied into device model and exposes structure and device tuning

Device Model Functionality

- Hotpluggable devices
  - Used to handle and communicate the plugging and unplugging of devices
- Device classes
  - Allows system to discover types of devices that are related
- Other
  - Common facilities such as reference counting
  - Capability to enumerate all devices and status

Kobject, kset and all

- struct kobject is used for:
  - reference counting
  - sysfs representation
  - "data structure glue" - representing relationships between devices
  - OO-like programming
  - hotplug event handling



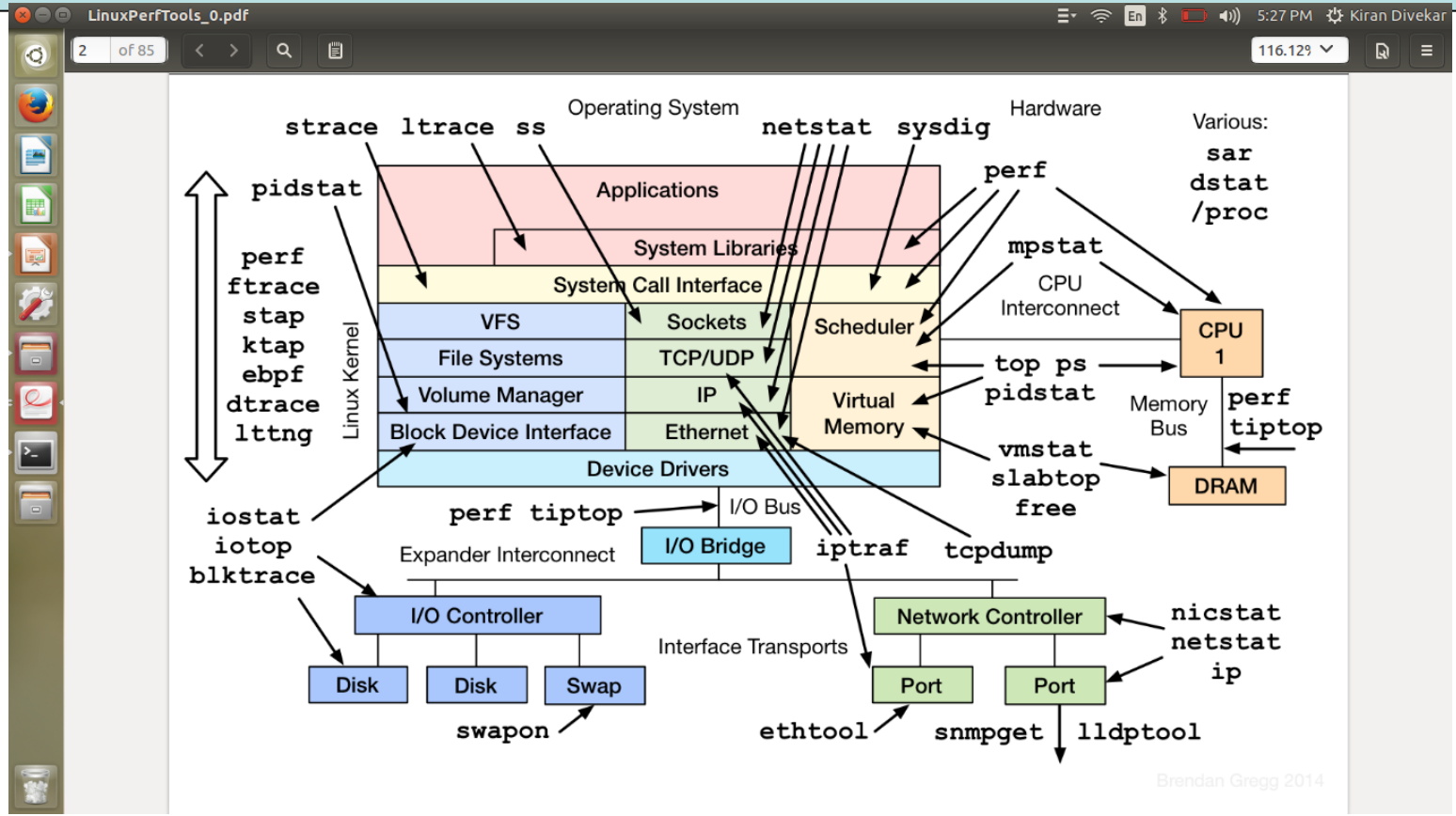
Kernel dis-section

- /proc interface
- strace
- top, htop, ps, pstree
- ulimit -a
- ipcs
- lsof -p <pid>
- ldd
- ifconfig, ping
- netstat -pantu

Kernel dis-section

- vmstat 1 : virtual memory
- iostat -xmdz 1 : IO disk statistics
- mpstat -p ALL 1 : multiple processor statistics
- pidstat -p 1
- sar -n TCP,DEV 1
- free -m

Kernel dis-section



Trace setup at run-time

- Pseudo-files in debugfs
  - e.g. mount debugfs –t debugfs /sys/kernel/debug
- Select a tracer
  - e.g. echo function_duration >current_tracer
- Set tracing parameters
  - e.g. echo 100 >tracing_threshhold
  - echo duration-proc >trace_options

Example of Use

```
$ mount debugfs -t debugfs /sys/kernel/debug
$ cd /sys/kernel/debug/tracing
$ cat available_tracers
function_graph function_duration function sched_switch nop
$ $ echo 100 >tracing_thresh
$ echo function_duration >current_tracer
$ echo 1 >tracing_on ; do \
   ls /bin | sed s/a/z/g ; done ; echo 0 >tracing_on
$ echo duration-proc >trace_options
$ cat trace >/tmp/trace.txt
$ cat /tmp/trace.txt | sort –k3 > /tmp/trace.txt.sorted
```

Tracer Comparison Table

	KFT	LTTng	SystemTAP
Target user	embedded developer	system admin	system admin
Instrumentation	every function, by compiler	static definitions in source	programmable, using external definition
Control interface	echo and cat, using /proc	lttctl and lttd, using netlink	stap, using insmod and kprobe
Overhead	medium	low	high
Trace format	ascii, fixed	binary, with xml schema	programmable, printf and ascii-art
Post-processing	kd, text-oriented	lttv, graphical	systemtapgui??

Many more:---

- Debugging:- kgdb
- Interrupt kernel drivers.
- GNU binary utilities.
- Bus drivers (pci, usb)
- Block, Network drivers
- Procfs, sysfs, configfs, systemtap

Thank You.