Introducing the Linux Kernel Programming Assignments

- All assignments implemented as kernel loadable modules.
  - NO kernel source modifications or kernel rebuilding
  - "make" with specialized kernel module Makefile
    - Access to source #include files for kernel build
    - Many kernel "helper" functions generated as in-line code from includes
  - Module inserted or removed from kernel with commands 'insmod' 'rmmod' (as root)
    - Dynamic linkage with symbols EXPORTED from kernel
    - Initialization function in module executed during insmod
    - Termination (cleanup) function in module executed during rmmod
  - Module can have both static and dynamic state variables
- Kernel loadable modules are used for device drivers, file systems, protocol stacks, etc.
  - Execute as if built directly from kernel source

Introducing the Linux Kernel Programming Assignments

- Modules loaded and executed in Red Hat Enterprise Linux 6.4 (kernel 2.6.32)
- RHEL 6.4 is to be run in virtual machine on ***your*** Windows, Mac OS-X, or Linux laptop/desktop
- Running in VM ***is essential*** because ***you will*** cause kernel 'panics' which hang the system and require a power cycle.
- The ***good*** news is the VM will reboot the kernel cleanly (you did not modify it permanently).
- The ***bad*** news is there are no effective debugging tools for the kernel so you have to be creative.

Linux 2.6.32-431 Source Space

3.4M	./tools	
224K	./ipc	
4.6M	./kernel	
40K	./usr	
2.1M	./scripts	
144K	./init	
17M	./Documentation	
20M	./net	
19M	./sound	
61M	./firmware	(loaded into programmable adapters)
1.5M	./security	
2.3M	./mm	
20M	./include	
2.0M	./crypto	
96K	./samples	
1.5M	./lib	
111M	./arch	(machine specific)
156K	./virt	
227M	./drivers	
672K	./block	
31M	./fs	
520M	.	

About 80% of source is hardware specific

Some Linux C Examples (like code you will need to understand)

```
struct pid *get_task_pid(struct task_struct *task, enum pid_type type)
{
        struct pid *pid;
        rcu_read_lock();
        if (type != PIDTYPE_PID)
                task = task->group_leader;
        pid = get_pid(task->pids[type].pid);
        rcu_read_unlock();
        return pid;
}
EXPORT_SYMBOL_GPL(get_task_pid);
```

Some Linux C Examples (like code you will need to understand)

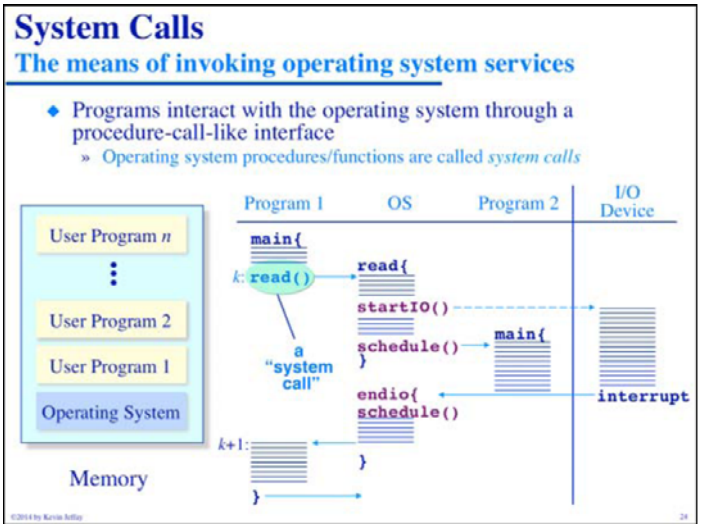
```
pid_t pid_vnr(struct pid *pid)
{
return pid_nr_ns(pid, task_active_pid_ns(current));
}
EXPORT_SYMBOL_GPL(pid_vnr);

 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
 {
         struct upid *upid;
         pid_t nr = 0;
         if (pid && ns->level <= pid->level) {
                 upid = &pid->numbers[ns->level];
                 if (upid->ns == ns)
                         nr = upid->nr;
         }
         return nr;
 }
```

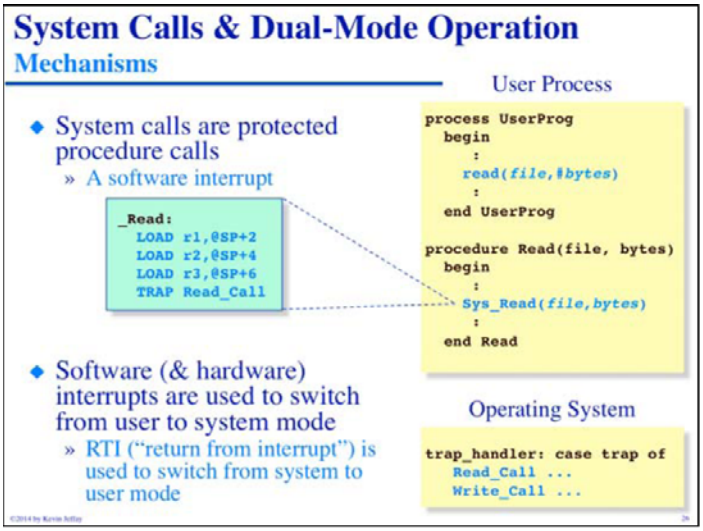

Some Linux C Examples (like code you will need to understand)

```
char *d_path(const struct path *path, char *buf, int buflen)
{
         char *res;
         struct path root;
         struct path tmp;
         read_lock(&current->fs->lock);
         root = current->fs->root;
         path_get(&root);
         read_unlock(&current->fs->lock);
         spin_lock(&dcache_lock);
         tmp = root;
         res = __d_path(path, &tmp, buf, buflen);
         spin_unlock(&dcache_lock);
         path_put(&root);
         return res;
}
EXPORT_SYMBOL(d_path);
```

From 530 Lecture: System Calls



From 530 Lectures: System Calls

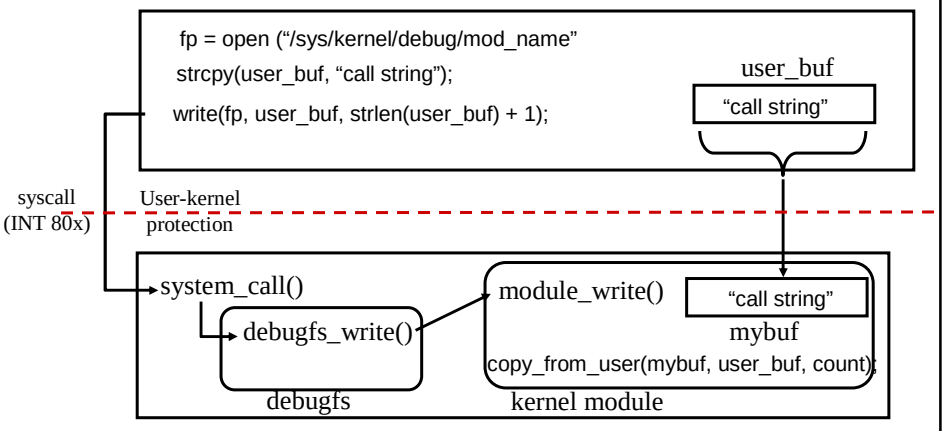


Introducing the Linux Kernel Programming Assignments

- Your modules will ***emulate*** a system call interface for user programs.
- A 'pseudo' file system, debugfs, will be used for communication between user space and kernel space.
  - User programs use read() and write() system calls
  - Kernel modules 'hook' the debugfs file system so reads and writes are passed to the module for execution.
- debugfs is mounted at /sys/kernel/debug
  - Kernel modules can create debugfs directories/files that can be opened and read/written by user programs
  - Kernel modules use kernel-provided functions to copy data between user space buffers and their kernel space buffers.

Introducing the Linux Kernel Programming Assignments

- Emulating a system call with debugfs (making the call)



Introducing the Linux Kernel Programming Assignments

- Emulating a system call with debugfs (returning result)