Modules & Proc Filesystem

/proc File System

- A virtual filesystem that permits a novel approach for communication between the kernel and user space.
- Virtual files can be read from or written
  - The content of these virtual files is dynamically created.

/proc File System

- Originally developed to provide information on the processes in a system
- Contains directories and virtual files.
- A virtual file:
  - can present information from the kernel to the user
  - can serve as a means of sending information from the user to the kernel

/proc File System

```
giorgio@GRik:~$ ls /proc
1       176    3475  7069  7846  8194  8609       cpuinfo      meminfo
10      177    3477  7070  7860  8245  8690       crypto       misc
10178   178    3749  7076  7865  8246  8729       devices      modules
10450   1788   3751  7082  7866  8258  8731       diskstats    mounts
10464   1872   4     7086  7869  8260  8733       dma          mtrr
10465   2      5     7089  7889  8264  8853       dri          net
10695   2008   6     7106  7901  8265  8988       driver       partitions
11      2024   6049  7121  7912  8273  8993       execdomains  scsi
133     2070   6657  7138  7983  8285  9          fb           self
14      2138   6684  7155  7984  8287  9040       filesystems  slabinfo
15      21817  6686  7170  7985  8289  9179       fs           stat
16      23129  6724  7188  7986  8290  9277       ide          swaps
1690    23173  6725  7239  7987  8339  9310       interrupts   sys
1691    29250  6747  7256  7988  8344  9360       iomem        sysrq-trigger
1696    2932   6854  7260  8     8350  9406       ioports      sysvipc
1697    3      6931  7280  8083  8355  9451       irq          tty
1698    3067   6973  7344  8101  8477  9494       kallsyms     uptime
1699    31172  6977  7665  8138  8491  acpi       kcore        version
17      3212   7     7731  814   8498  asound     key-users    version_signature
174     3217   7018  7732  8141  8499  buddyinfo  kmsg         vmnet
175     3472   7039  7818  8142  8514  bus        loadavg      vmstat
17580   3473   7065  7834  8145  8601  cmdline    locks        zoneinfo
```

/proc File System

```
root@GRik:~# ls /proc/1
attr      cpuset   exe   mem          oom_adj    seccomp  statm   wchan
auxv      cwd      fd    mounts       oom_score  smaps    status
cmdline   environ  maps  mountstats   root       stat     task
root@GRik:~# cat /proc/1/cmdline
init [2]
[root@GRik]# cat /proc/sys/net/ipv4/ip_forward
0
[root@GRik]# echo "1" > /proc/sys/net/ipv4/ip_forward
[root@GRik]# cat /proc/sys/net/ipv4/ip_forward
1
[root@GRik]#
```

Modules

- Loadable Kernel Modules (LKM) are a way to dynamically add or remove code from the Linux kernel.
- LKMs are also a popular mechanism for device drivers and filesystems in the Linux kernel.

Modules

- When a driver is compiled directly into the kernel, its code and static data occupy space even if they're not used.
- If the driver is compiled as a module, it requires memory only if memory is needed and subsequently loaded, into the kernel.
- Modules are a powerful means of creating a lean kernel that adapts to its environment based upon the available hardware and attached devices.

Modules

- Managed using:
  - insmod <modname>, rmmod <modname>
  - modinfo <modname>
  - lsmod <modname>
  - modprobe

Compiling Modules

- Note: this applies to 2.2 kernels
- Add:

```
#define __KERNEL__
#define MODULE
#include <linux/modversions.h>
#include <linux/module.h>
```

- Compile with:

```
cc -Wall -c -I/usr/src/linux-2.2.26/include [-nostdinc]
```

Compiling Modules

2.6 kernels:

- Typical Makefile:

```
obj-m += <name>.o

all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
   modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Compiling Modules

Compile as usual:

```
hostname:# make
make -C /lib/modules/2.6.11/build M=/root/lkmpg-examples/02-
   HelloWorld modules
make[1]: Entering directory `/usr/src/linux-2.6.11'
  CC [M]  /root/lkmpg-examples/02-HelloWorld/hello.o
 Building modules, stage 2.
  MODPOST
  CC      /root/lkmpg-examples/02-HelloWorld/hello.mod.o
  LD [M]  /root/lkmpg-examples/02-HelloWorld/hello.ko
make[1]: Leaving directory `/usr/src/linux-2.6.11'
hostname:#
```

Modules

```
/* Entry and exit functions */

init_module(void);

cleanup_module(void);
```

Functions

```
struct proc_dir_entry *create_proc_entry ( const char *name, mode_t
   mode, struct proc_dir_entry *parent );

void remove_proc_entry( const char *name, struct proc_dir_entry *parent
   );

struct proc_dir_entry {
        …
        int (*read_proc)(char *page, char **start, off_t off,
                         int count, int *eof, void *data);
        int (*write_proc)(struct file *file, const char *buffer,
                          unsigned long count, void *data);
        int (*readlink_proc)(struct proc_dir_entry *de, char *page);
        …
};
```

Other Useful Functions

```
unsigned long copy_to_user(
   void __user *to,const void *from,unsigned long n );

unsigned long copy_from_user(
   void *to,const void __user *from,unsigned long n );

void *vmalloc( unsigned long size );
void vfree( void *addr );

void *kmalloc( unsigned long size );
void kfree( void *addr );
```

kmalloc() vs. vmalloc()

- kmalloc() allocates physically contiguous memory, memory which pages are laid consecutively in physical RAM.
- vmalloc() allocates memory which is contiguous in kernel virtual memory space (that means pages allocated that way are not contiguous in RAM, but the kernel sees them as one block).

- In order to use DMA from/to some hardware device, you'll need to use kmalloc