Embedded Systems Programming

Linux Kernel Modules
(Module 4)

Yann-Hang Lee

Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014

What is “kernel”

- **The basic component of an operating system**
  - to provide the lowest-level abstraction layer for the resources (especially processors and I/O devices).
  - available to application processes through inter-process communication mechanisms and system calls
- **Kernel space and user space**
- **What are system calls and how many are they?**
  - Which are systems calls –

    cc, make, ls, cat, grep, read, open, printf, malloc, etc.
  - How can we set the baud rate of a serial port?
    - Configuration of a hyperterminal
    - stty -F /dev/ttyS2 ospeed 38400
    - Ioctl to get and set terminal attributes (defined in *struct termios)*
  - The mechanism of making system calls and passing parameters

Kernel Components

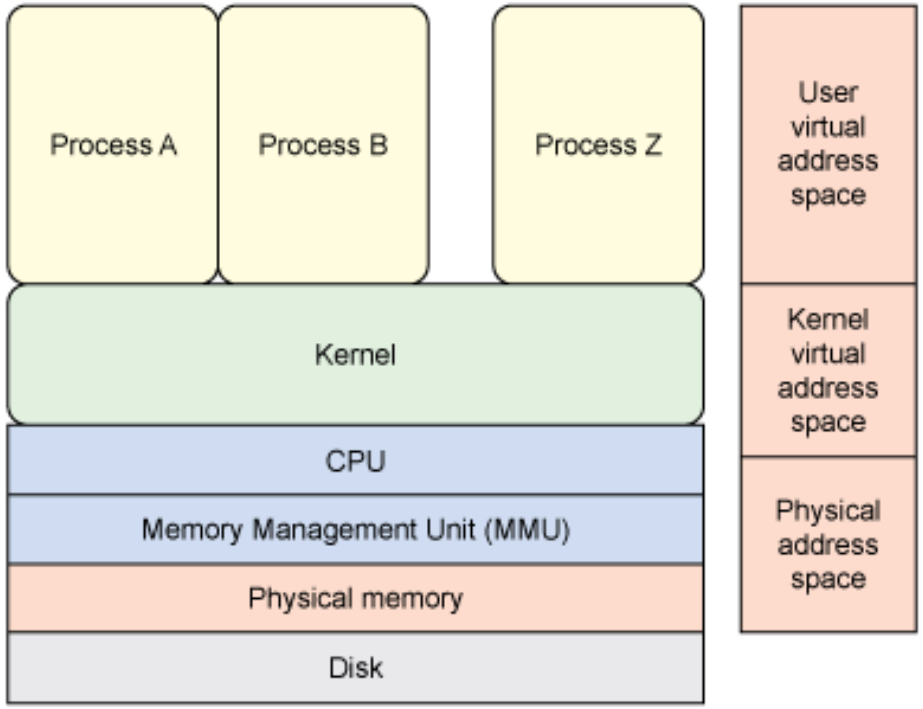
- **Process Management**
  - Process life cycle, Inter Process Communication, I/O
  - Scheduling (long-term, short-term)
- **Memory Management**
  - Virtual memory, management, security
- **File System**
  - File system tree, management, security
- **Device Control**
  - Almost every system operation maps to a physical device
  - The code used to do those operations is called Device Driver
- **Networking**
  - Collect incoming packets and De-Multiplexing them
  - Deliver outgoing packets

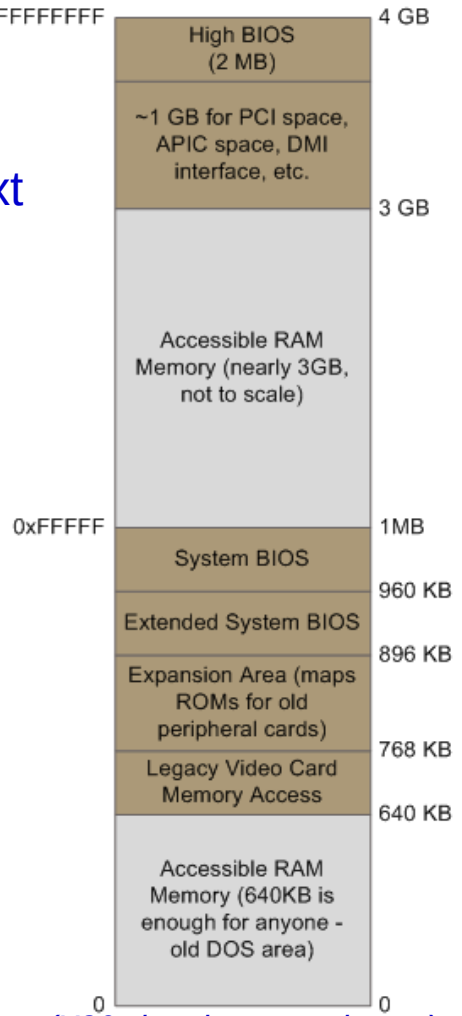
Memory Spaces (1)

- **The logical address space of a process is divided into two parts**
  - – 0x00000000 to PAGE_OFFSET-1 can be addressed in either user or kernel mode
  - – PAGE_OFFSET to 0xffffffff can be addressed only in kernel mode
  - – PAGE_OFFSET is usually 0xc00000000
- **Peripheral devices are controlled by writing and reading their registers. Where are these registers?**
- **I/O Port: request an IO port region and *inb (outb)* from *<asm/io.h>***
- **I/O Memory:**
  - Request a memory region
  - Devices at physical addresses which have to be mapped to virtual addresses for software to access.

Memory Spaces (2)

- **Physical memory**
- **Virtual memory (user and kernel space)**
  - Linux ARM -- Linux/Documentation/arm/memory.txt





(X86 virtual memory layout)

Linux Kernel Modules

- **Modules can be compiled and dynamically linked into kernel address space.**
  - Useful for device drivers that need not always be resident until needed. (why?)
  - Extend the functionality of the kernel without rebuilding and rebooting the system.
- **Kernel keeps a symbol table**
  - Symbols accessible to kernel-loadable modules appear in `/proc/kallsyms`.
  - EXPORT_SYMBOL(), which exports to any loadable module, or
  - EXPORT_SYMBOL_GPL(), which exports only to GPL-licensed modules.
- **Can call any functions exported by the kernel**
  - no library is linked to modules

Kernel Modules

- **Pieces of code that can be loaded and unloaded into the kernel.**
  - a module registers itself in order to serve future requests
  - Is a part of kernel – printf or printk

- **An example module**

```
#include <linux/module.h>        // Needed by all modules
#include <linux/kernel.h>        // Needed for KERN_ALERT
#include <linux/init.h>          // Needed for the macros
static int hello_2_init(void)  {
    printk(KERN_ALERT "Hello, world 2\n");
    return 0;   }
static void hello_2_exit(void) {
    printk(KERN_ALERT "Goodbye, world 2\n");          }

module_init(hello_2_init);
module_exit(hello_2_exit);
```

(Add code/Makefile example)

Programs for Linking and Unlinking Modules

- **insmod**
  - Invokes create_module( ) and query_module( ) system calls
  - Using the kernel symbol table, the module symbol tables, and the address returned by the create_module( ) system call, relocates the object code.
  - Allocates a memory area in the User Mode address space and loads with a copy of the module object
  - Invokes the init_module( ) system call, passing to it the address of the User Mode memory area
  - Releases the User Mode memory area and terminates
- **lsmod**
- **rmmod**
- **modprobe**
  - loads a module into the kernel.
  - check any module dependency and load any other modules that are required – stacking of modules

Module Implementation

- **The kernel considers only modules that have been loaded into RAM by the *insmod* program and for each of them allocates memory area containing:**
  - a module object
  - null terminated string that represents module's name
  - the code that implements the functions of the module
- **Building**
  - Linux kbuild
  - *obj-m*
  - *make -C ~/kernel-2.6 M=`pwd` modules* to build modules.ko
    - *"M="* is recognized and kbuild is used
    - *~/kernel-2.6* is the kernel source directory
    - *pwd* ??

Supplementary Slides

HelloWorld Kernel Module Example (1)

```
/* Hello World kernel module program */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/time.h>
#include <linux/delay.h>
static int hello_init(void)    {
      int i;
      struct timeval curTime;   // keeps the time since the Epoch
      struct tm bkd_time;       // containing a calendar date and time
      for (i=0; i<10; i++)  {
            do_gettimeofday(&curTime);  time_to_tm(curTime.tv_sec, 0, &bkd_time);
            printk(KERN_ALERT "Hello, world. The current time is: %d:%d:%d:%ld\n",
                bkd_time.tm_hour, bkd_time.tm_min, bkd_time.tm_sec, curTime.tv_usec);
            msleep(i*1000);   }
}
static void hello_exit(void)    {
   printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

HelloWorld Kernel Module Example (2)

```
# Makefile for Hello World kernel module

CC = i586-poky-linux-gcc
ARCH = x86
CROSS_COMPILE = i586-poky-linux-
SROOT=/opt/clanton-full/1.4.2/sysroots/i586-poky-linux/

obj-m = HelloModule.o

all:
        make ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) -C
                $(SROOT)/usr/src/kernel M=$(PWD) modules

----  copy to the target
scp   HelloModule.ko    root@10.218.101.25:/home/.....

----  install and remove the module
insmod  HelloModule.ko
rmmod HelloModule
```