Kernel

黄悦民 教授

2014/03/21

Outline

- Introduction
- User mode & kernel mode
- System call
- Interrupt
- Kernel module
- Related information about Linux kernel
- Lab guide

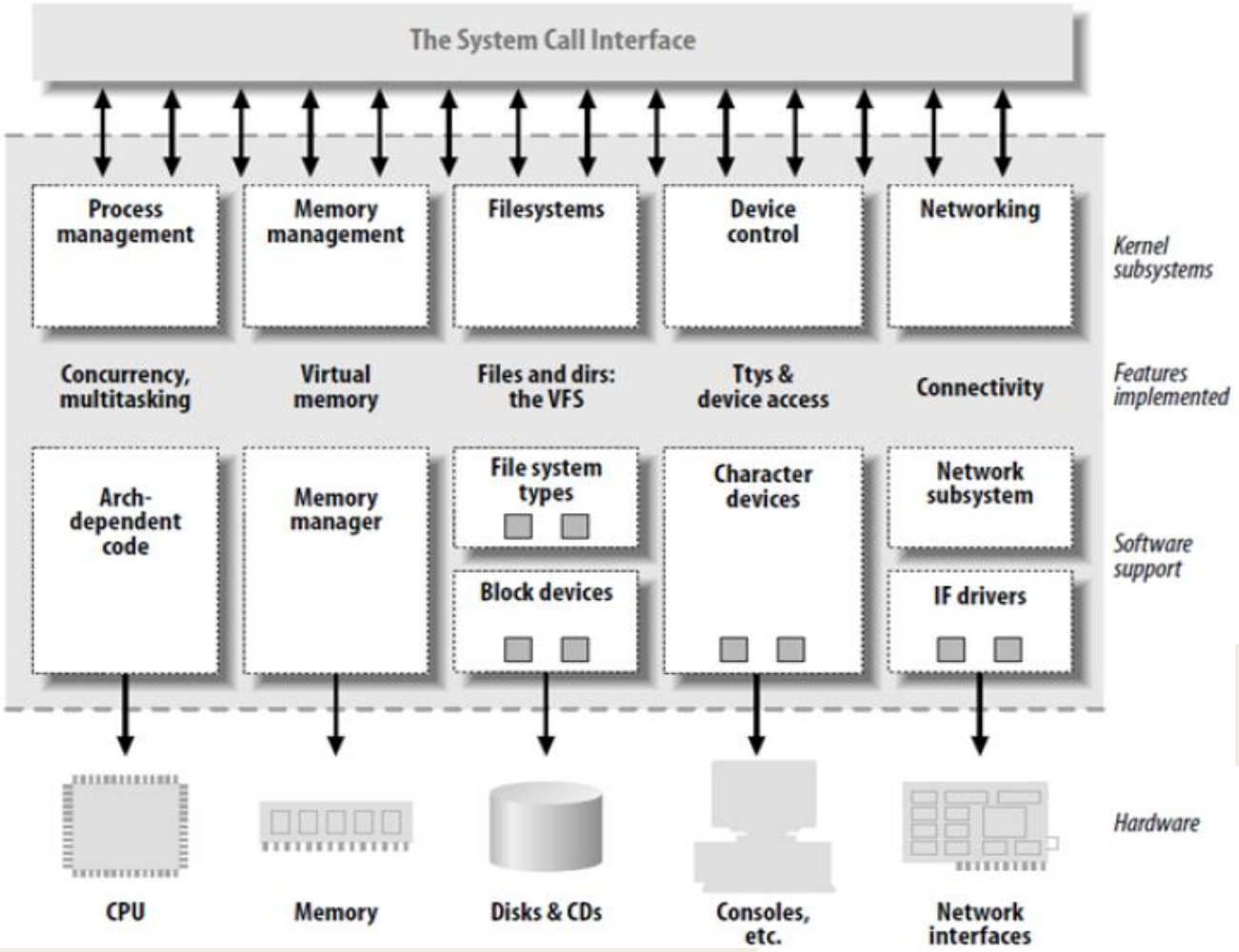
What is Kernel(1/2)

- The most important program of operating system, it’s contains many critical procedures that are needed for the system to operate
  - Kernel can be thought as the main software of OS, other progarms of OS are less crucial utilities

What is Kernel(2/2)

- Process management
  – Determinate process's life cycle, control process I/O and distribute CPU resource for scheduler
- Memory Management
  – Communicate with memory management subsystem ex: malloc/free
- File System
  – Kernel support multiple file system
- Device Control
  – System call will be mapping to corresponding hardware device, every device has unique code to operate, call device driver
- Networking
  – The kernel provide send/receive mechanism to handle incoming data

Kernel architecture



Referenced from **Building Embedded Linux Systems,** edited by Karim Yaghmour *et al*, O'REILLY

User mode & Kernel mode(1/3)

- Hardware(processor) provides two different operative modes
  - Kernel mode
  - User mode
- Except processor support, OS support is also needed
- This design allows the OS to run with more privileges than application program

User mode & Kernel mode(2/3)

- User mode
  - Restricted level, the executing code has no ability to directly access hardware or reference memory.
  - Code running in user mode to access hardware or memory must via system library(ex:glibc)

User mode & Kernel mode(3/3)

- Kernel mode
  - The most unrestricted level, allows access to all the computer's resources
    * Any CPU instruction can be executed and every memory address can be accessed
- System calls, interrupts, and exceptions are changed mode from user to kernel

System call(1/3)

- System call is a mechanism that is used by the application program to request a service from the OS
- Each system call provides a basic operation, such as:
  - Opening a file
  - Reading a character
  - etc.

System call(2/3)

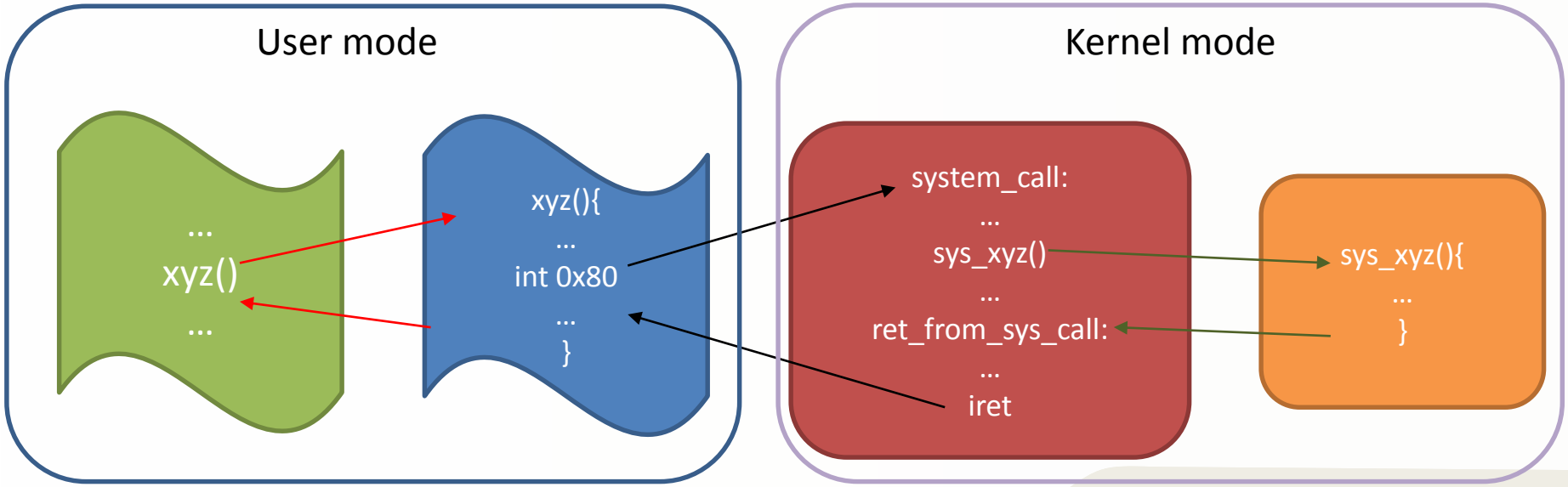
- Even for same OS version, different processor architectures make system calls in different ways
  - Using a machine-code instruction that causes the processor to change mode
  - System call numbers are different for various processor architectures

System call(3/3)

- System calls are generally not invoked directly, but rather via wrapper functions in system libraries, such as:
  - glibc(on Unix-like systems)
  - Native API (on Windows NT)
  - etc.

Accessing Kernel via Library interface

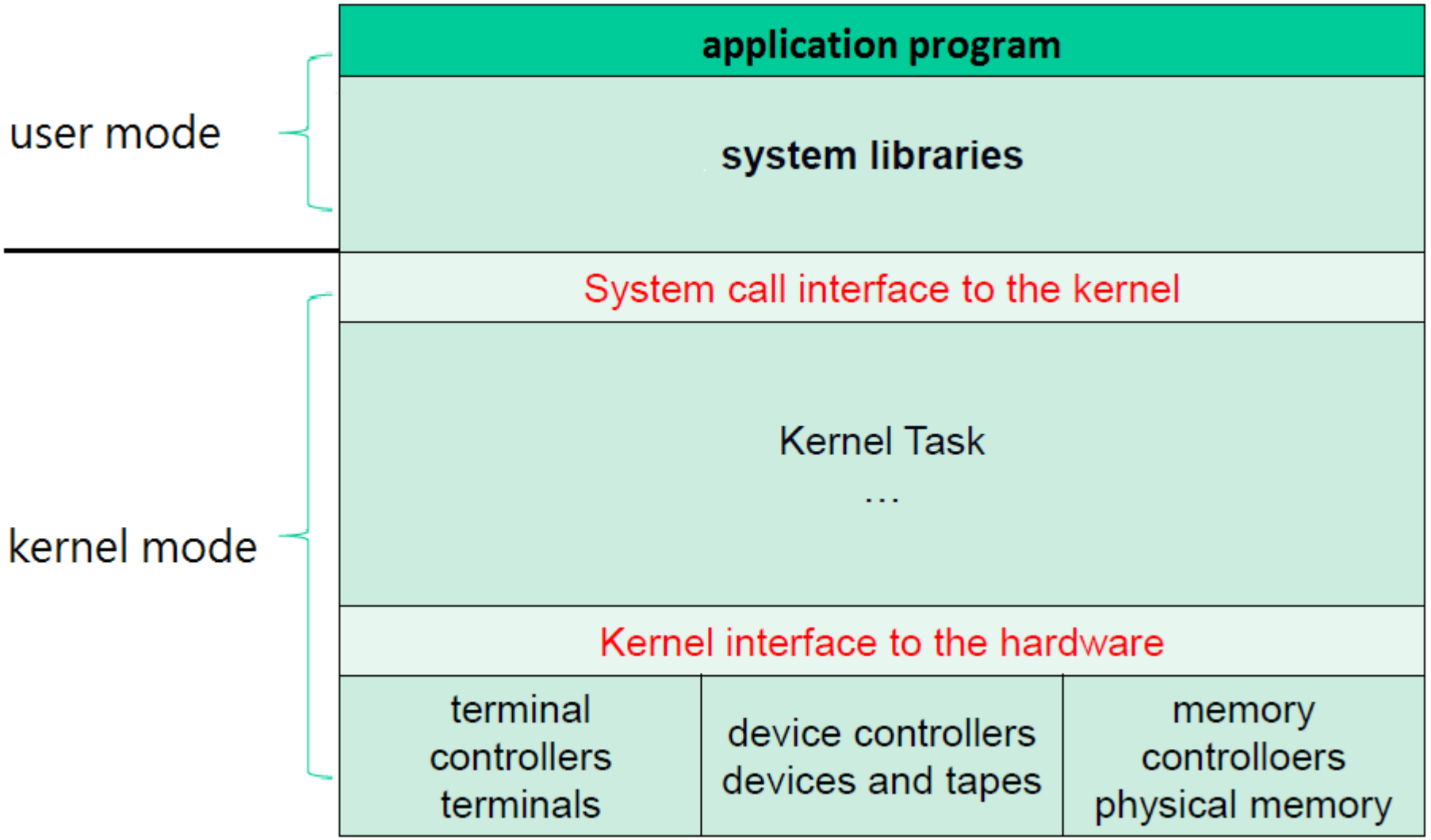
- Invoking a system call



Note: In Linux system, the system calls must be invoked by executing the **int 0x80** Assembly instruction, which raises the programmed exception having **vector 128**

Referenced from **Understanding the LINUX KERNEL**, edited by DANIEL P. BOVET & MARCO CESATI, O'REILLY

System Structure



What is Interrupt ?

- An interrupt is usually defined as a signal to the CPU emitted by hardware or software that stops the execution of a running program and perform another action(Interrupt Service Routine)

Interrupt classification(1/5)

- Interrupts are often divided into:
  - Synchronous interrupts
    * CPU control unit issues them only after terminating the excution of an instruction
  - Asynchronous interrupts
    * Generated by other external hardware devices at arbitrary times, it can occur in the middle of instruction execution

Interrupt classification(2/5)

- The Intel documentation(x86 manuals) designate asynchronous & synchronous interrupts as ***interrupts*** & ***exceptions*** and classifies them as:
  – Interrupts
    * Maskable interrupts
      - The signal sent to the INTR pin of CPU, they can be disable by clearing the flag. All IRQs issued by I/O devices give rise to maskable interrupts.

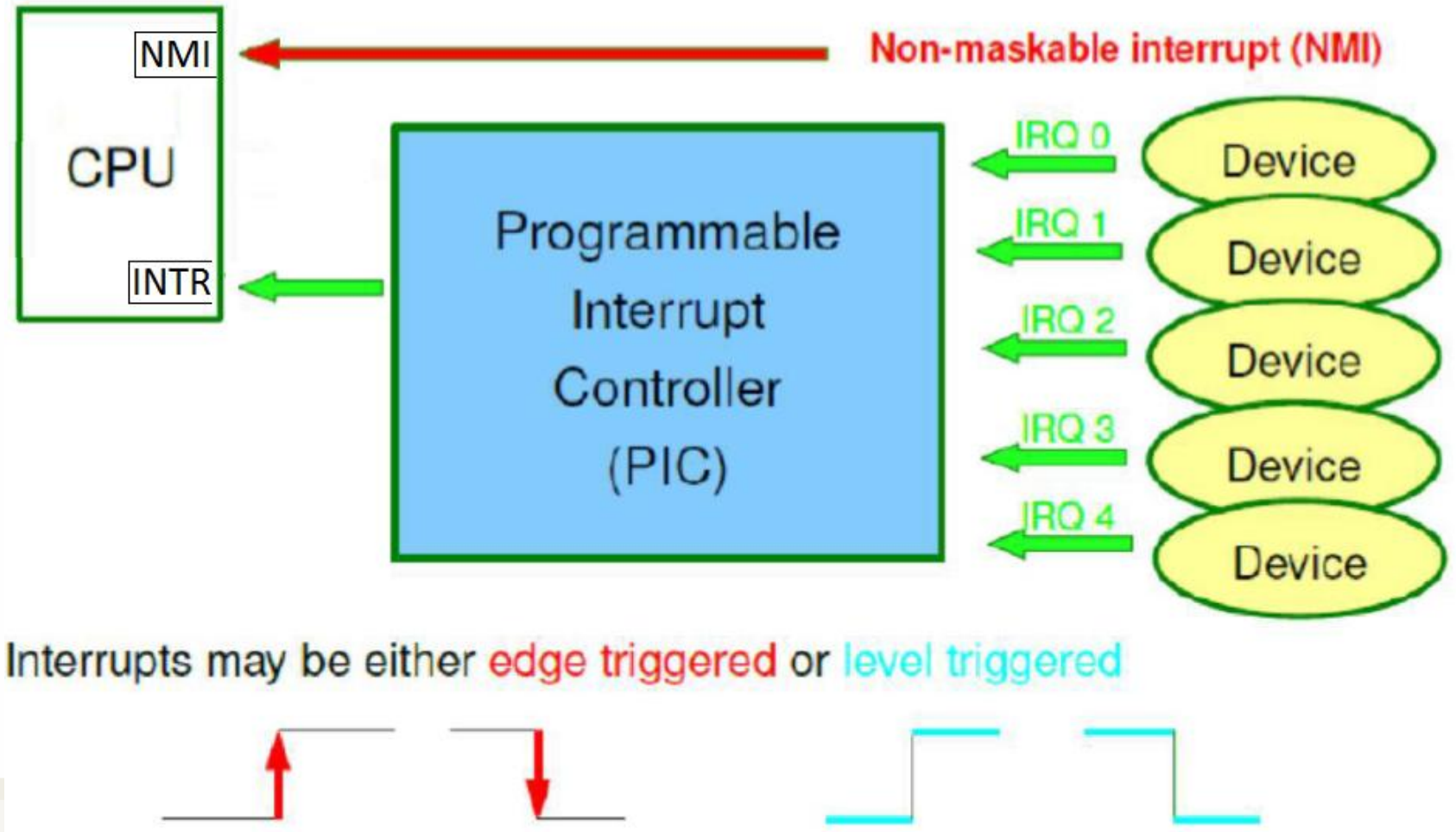
Interrupt classification(3/5)

– Interrupts(cont)

* Nonmaskable interrupts

- The signal sent to the NMI pin of CPU, they can not disable by clearing the flag. Only a few critical events(ex: hardware failures) give rise to nonmaskable interrupts

Interrupt classification(4/5)



Interrupt classification(5/5)

– Exceptions

* Processor-detected exceptions

- Generted when the CPU detects an anomalous condition (ex:divide by zero) while executing an instruction

* Programmed expections

- Occur at the request of the program(system call), they are trigged by **int** or **int3** assembly instructions

- They are often called ***software interrupts***

Kernel module

- An object file that can be inserted or removed dynamically to the running kernel
- Kernel modules are typically used to add support for new hardware or filesystem, or for adding system calls
  - USB
  - SATA
  - etc.

Related information about kernel

- Linux kernel source official website: www.kernel.org
- To date, the lastest stable version is 3.13.6

- Get the version number of your Linux
  - $ uname -r
  - ex:

```
jim@jim-BM6630-BM6330-BP6230:~$ uname -r
3.6.1
```

Reference

- DANIEL P. BOVET and MARCO CESATI, *Understanding the LINUX KERNEL*, O'EILLY.
- Linux kernel official web.,*https://www.kernel.org/*

Lab Guide

Today’s lab.

- Install related package
- Build kernel image file
- Verify on the board

Getting lab file

- Download lab file from course website:
  - http://mmn.twbbs.org
  - 102學年度上學期
    * [在職專班]網路嵌入式系統應用
    * Account => embedded102
    * Password => embedded102

Install related package

- Install mkimage package
  - $ sudo apt-get install uboot-mkimage -y

```
$ sudo apt-get install uboot-mkimage -y
```

Get kernel directory(1/2)

- Uncompress **linux-02.01.03.11.tar.gz**
  - $ tar -zxvf linux-02.01.03.11.tar.gz
- Enter the kernel-source directory
  - $ cd linux-02.01.03.11/

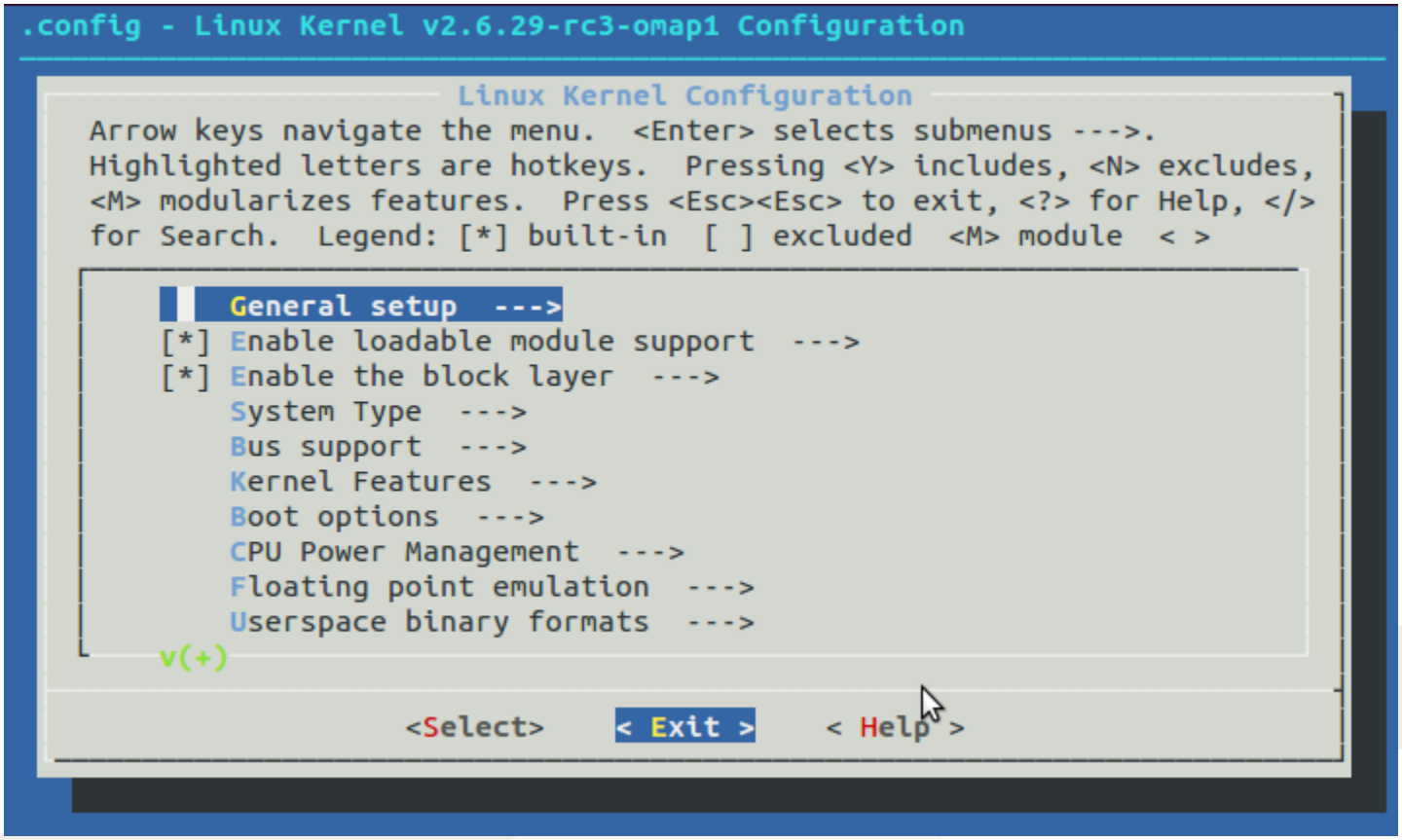
Get kernel directory(2/2)

- Patching
  - A patch is a modify technique to update/modify kernel
  - A patch file describes the changes (lines removed or lines added) to the kernel
  - Using the **patch** command
    - $ patch -p1 < ../linux-02.01.03.11-devkit8000-2.patch

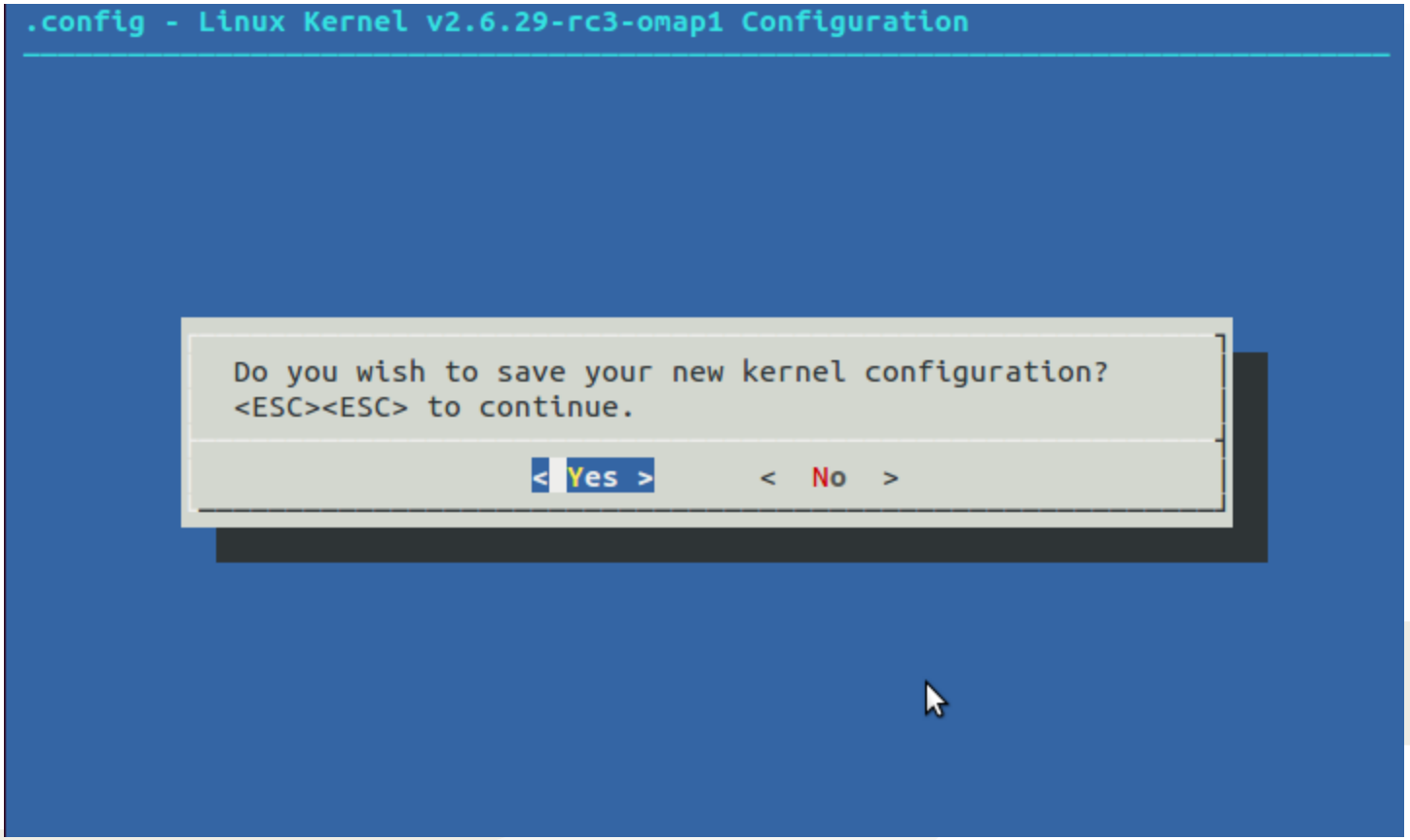
Compile Kernel steps(1/5)

- Copy config file
  - $ cp arch/arm/configs/omap3_devkit8000_defconfig .config
- $ make menuconfig

Compile Kernel steps(2/5)



Compile Kernel steps(3/5)



Compile Kernel steps(4/5)

- $ make
- $ make uImage

It is I, not l

Compile Kernel steps(5/5)

- When it is completed, you can see that:

```
  LD [M]  drivers/usb/serial/usbserial.ko
jim@jim:~/Downloads/linux-02.01.03.11$ make uImage
  CHK     include/linux/version.h
make[1]: `include/asm-arm/mach-types.h' is up to date.
  CHK     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-arm
  CALL    scripts/checksyscalls.sh
<stdin>:1097:2: warning: #warning syscall fadvise64 not implemented
<stdin>:1265:2: warning: #warning syscall migrate_pages not implemented
<stdin>:1321:2: warning: #warning syscall pselect6 not implemented
<stdin>:1325:2: warning: #warning syscall ppoll not implemented
<stdin>:1365:2: warning: #warning syscall epoll_pwait not implemented
  CHK     include/linux/compile.h
  Kernel: arch/arm/boot/Image is ready
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-2.6.29-rc3-omap1
Created:      Mon Mar 17 22:44:46 2014
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2210992 Bytes = 2159.17 kB = 2.11 MB
Load Address: 80008000
Entry Point:  80008000
  Image arch/arm/boot/uImage is ready
```

Put kernel image into SD card

- Copy **uImage** file from specific directory to SD card
  - $ cp arch/arm/boot/uImage /media/boot

Prepare for boot

- Safely remove SD card and insert it into pandaboard.
  - Note: Do not insert or remove the SD card when the board is powered on
- Connecting DevKit8000 to PC by using RS232 connector.
- Start serial communication software(C-Kermit)
  - $ sudo kermit -c
- Turn on the DevKit8000 board
  - Plug power connector into the power slot of DevKit8000.

Let’s boot

- Initial mmc controller.
  - # mmcinit
- Load u-boot.bin into the RAM.
  - # fatload mmc 0:1 0x80300000 uImage
- Booting from appropriate address of RAM.
  - # bootm 0x80300000

Verify

```
OMAP3 DevKit8000 # mmcinit
OMAP3 DevKit8000 # fatload mmc 0:1 0x80300000 uImage
reading uImage

2211056 bytes read
OMAP3 DevKit8000 # bootm 0x80300000
## Booting kernel from Legacy Image at 80300000 ...
   Image Name:   Linux-2.6.29-rc3-omap1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2210992 Bytes =  2.1 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ...
```

```
eth0: link down
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
     device=eth0, addr=192.168.0.10, mask=255.255.255.0, gw=255.255.255.255,
     host=192.168.0.10, domain=, nis-domain=(none),
     bootserver=192.168.0.9, rootserver=192.168.0.9, rootpath=
Waiting 1sec before mounting root device...
Looking up port of RPC 100003/2 on 192.168.0.9
rpcbind: server 192.168.0.9 not responding, timed out
Root-NFS: Unable to get nfsd port number from server, using default
Looking up port of RPC 100005/1 on 192.168.0.9
```