Contents

[ Interview questions: 1](#_Toc88317072)

[1. Why do you think you are a good fit for the job position you’re applying for? 1](#_Toc88317073)

[2. Working experience for all companies: 1](#_Toc88317074)

[**~~1)~~** **~~三一重工：~~**~~SANY Heavy Industry Company: automobile body structure design for bus.~~ 1](#_Toc88317075)

[2) 上富Shangfu Technologies 1](#_Toc88317076)

[3) 优思Uniscope Technologies 1](#_Toc88317077)

[4) 共进Shanghai Gongjin Communications Technology: 1](#_Toc88317078)

[5) 龙爵Zhuhai Longjue Technology: 1](#_Toc88317079)

[3. What are the most challenging problems/issues you debugged before? 1](#_Toc88317080)

[1) BSP: 1](#_Toc88317081)

[2) Network driver: 1](#_Toc88317082)

[3) Deadlock issue: 1](#_Toc88317083)

[ BSP (priority): 1](#_Toc88317084)

[4. Summarize the UBOOT Board bring-up? 1](#_Toc88317085)

[5. How does the Bootloader pass the parameters to the kernel? 1](#_Toc88317086)

[6. Memory Data segment 1](#_Toc88317087)

[7. ARM RISC architecture? 1](#_Toc88317088)

[8. 7 CPU modes for ARM? 1](#_Toc88317089)

[9. ARM CPU registers? 1](#_Toc88317090)

[10. Current Program Status Register 1](#_Toc88317091)

[11. CPU I used in previous projects? 1](#_Toc88317092)

[12. What’s the vector table? 1](#_Toc88317093)

[13. Explain ARM interrupt switch (register level)? 1](#_Toc88317094)

[14. 中断嵌套 interrupt nest 1](#_Toc88317095)

[15. What is mmu and what does it do? 1](#_Toc88317096)

[16. Describe the kernel starts up? 1](#_Toc88317097)

[17. File system customization? 1](#_Toc88317098)

[ Drivers 1](#_Toc88317099)

[18. Device driver 1](#_Toc88317100)

[19. SDRAM driver 1](#_Toc88317101)

[20. NAND flash: 1](#_Toc88317102)

[1) NAND flash driver summary 1](#_Toc88317103)

[2) For the operations on NAND flash: 1](#_Toc88317104)

[21. NOR Flash driver summary: 1](#_Toc88317105)

[22. Ethernet Network driver summary 1](#_Toc88317106)

[23. Router vs switch 1](#_Toc88317107)

[24. Touch panel driver 1](#_Toc88317108)

[25. Description on the input subsystem in Linux? 1](#_Toc88317109)

[26. LCD driver 1](#_Toc88317110)

[27. Camera - Video stream (V4l2 under Linux) 1](#_Toc88317111)

[28. ALSA 1](#_Toc88317112)

[29. UART: 1](#_Toc88317113)

[30. SPI: 1](#_Toc88317114)

[31. 485 1](#_Toc88317115)

[32. CAN 1](#_Toc88317116)

[33. I2C 1](#_Toc88317117)

[34. SCCB 1](#_Toc88317118)

[35. DMA 1](#_Toc88317119)

[36. Bluetooth 1](#_Toc88317120)

[37. PCI/PCIe 1](#_Toc88317121)

[38. PWM(Pulse Width Modulation) 1](#_Toc88317122)

[39. USB 1](#_Toc88317123)

[40. how does Jtag work 1](#_Toc88317124)

[41. Objdump 1](#_Toc88317125)

[ interrupt 1](#_Toc88317126)

[42. Why should we avoid the context switching during interrupt? 1](#_Toc88317127)

[43. Request\_irq 1](#_Toc88317128)

[44. Explain software interrupt? 1](#_Toc88317129)

[45. Top and Bottom Halves 1](#_Toc88317130)

[46. To deploy a module inside kernel, what are the possible methods? 1](#_Toc88317131)

[47. In how many ways we can assign a major minor number to any device? 1](#_Toc88317132)

[48. When is that we want to use "user virtual address" instead of "kernel virtual address"? List some situations when we cannot go with kernel virtual address. 1](#_Toc88317133)

[49. how does kernel space communicate with user space? 1](#_Toc88317134)

[ synchronization 1](#_Toc88317135)

[50. What are the best synchronization techniques used in linux kernel? 1](#_Toc88317136)

[51. IPCs in the Linux user space? 1](#_Toc88317137)

[52. How function pointers are shared across different processes? Using which IPCs? 1](#_Toc88317138)

[53. Difference between semaphore and mutex 1](#_Toc88317139)

[54. Difference between binary semaphore and mutex 1](#_Toc88317140)

[ kernel 1](#_Toc88317141)

[55. As kernel can access user space memory, why should copy\_from\_user is needed? 1](#_Toc88317142)

[56. System call 1](#_Toc88317143)

[57. What’s the PCB? 1](#_Toc88317144)

[58. Thread vs Process? 1](#_Toc88317145)

[59. Explain the Linux task scheduler? Is Linux RTOS? 1](#_Toc88317146)

[60. Difference between Linux and other RTOS (e.g. VxWorks)? 1](#_Toc88317147)

[61. Priority inversion? 1](#_Toc88317148)

[62. fopen vs open 1](#_Toc88317149)

[63. kernel thread vs user thread 1](#_Toc88317150)

[64. Critical section 1](#_Toc88317151)

[65. Deadlock 1](#_Toc88317152)

[66. Circular Wait 1](#_Toc88317153)

[67. What’s Starvation? 1](#_Toc88317154)

[68. MMAP vs ioremap 1](#_Toc88317155)

[69. MMU? 1](#_Toc88317156)

[70. Zombie process and orphan process: 1](#_Toc88317157)

[71. What are the process items that are inherited from the process’s parent? 1](#_Toc88317158)

[72. VFORK () vs fork () 1](#_Toc88317159)

[73. What is the difference between a process and a thread? 1](#_Toc88317160)

[74. Do you know the segments in which a program gets divided? 1](#_Toc88317161)

[75. which segments do threads share? 1](#_Toc88317162)

[76. Reentrant function 1](#_Toc88317163)

[77. Exec and system 1](#_Toc88317164)

[78. Linux的system() 和popen()差异 1](#_Toc88317165)

[1) summarize 1](#_Toc88317166)

[2) 执行流程 1](#_Toc88317167)

[3) 信号处理 1](#_Toc88317168)

[4) 4. 功能 1](#_Toc88317169)

[5) NOTE 1](#_Toc88317170)

[ command 1](#_Toc88317171)

[ C++ 1](#_Toc88317172)

# Interview questions:

## Why do you think you are a good fit for the job position you’re applying for?

NIO:

First, thank you very much for spending time to interview me. My name is Shawn.

* My last job was working for NIO as a senior system software engineer.
  + I am responsible for the manufacture test. Including porting device drivers under Linux/QNX/Android OS and developing the manufacture test application.
  + Developed the diagnostics software applications for diagnosing hardware failures.
  + Responsible for developing Secure boot and DoIP features.
  + Worked closely with hardware design team and Chip vendors on debugging device drivers and troubleshooting software and hardware issues.
* I worked on the automobile industry as a vehicle engineer for 2 years. At that time, I was a hardware engineer, my job was designing the new schematic of the Parking system and doing the PCB layout.
* And then I started to work as an embedded software engineer in the telecommunication industry. My working experience includes bootloader development, porting and debugging different kinds of drivers such as network device/touch panel/LCD/Bluetooth/camera and so on and designing software applications under Linux OS. Also, I need to resolve the system problems. For example, if the system crashed, I need to locate the causes and fixed it.
* ~~I have rich practical experience on Linux bootloader development and the system integration on different kinds of drivers and applications under Linux.~~

So, I think I am a good fit for this job position I am applying for. **I’m fully charged!**

Pure Storage:

Thank you very much for spending time to interview me. My name is Shawn.

I have about 10 years’ working experience on the embedded software development in the telecommunication industry. I have rich practical experience on Linux bootloader development and the integration on different kinds of drivers such as NAND flash, NOR flash, PCI-e, network device, i2c, camera and so on. And I also worked on the software automation design on hardware diagnosis of different device drivers before.

I am very interested in this embedded software engineer position. And I think I have the key skills required by the job position such as C and python experience, strong problem-solving skills etc. I believe I am a good fit for this job position, I would appreciate it if you would consider my application.

What makes you a hands-on embedded software engineer?

I have about 10 years’ working experience in embedded software development in the telecommunication industry. I have rich practical experience in Linux bootloader development and the integration of different kinds of drivers such as UART, I2C, SPI, NAND/NOR Flash, SDRAM, Camera, USB, PCI, Ethernet MAC&PHY, xDSL DSP and so on.

Moreover, the projects I worked on before allowing me to gain practical skills on using different kind of tools including JTAG, multi-meters, oscilloscopes, logic analyzer, Trace32, BDI3000, GDB, Smartbits.

Furthermore, I have strong troubleshooting and problem-solving skills on software of Linux system.

Your programming experience in any IoT related device and IoT Security…

I worked on Uboot, device driver and software application design based on the Linux OS by using C programming language on GPON Gateway product and Set-top box product before.

## Working experience for all companies:

### 上富Shangfu Technologies

* Hardware engineer

The product is a Parking system which is designed for vehicles to alert the driver to obstacles while parking. These systems use ultrasonic sensors to detect the distance between the car and obstacles and beeps different sounds according to different distance.

I was a hardware engineer; my job was designing the new schematic of the system and doing the PCB layout.

### 优思Uniscope Technologies

* Embedded software engineer
* When I worked on a new project:
  + The first thing I need to do is go through the customer requirement and port the drivers, such as LCD driver, touch panel driver, camera and so on under MTK platform.
  + And I also worked on writing the automation test code for the drivers. After downloading the test code to the cell phone, we can press specific keys to start the automation test.
  + The test code calls the drivers to execute all kinds of tests. For example, the test codes let the camera uses LCD to show different colors, and lets camera do the recording and then play back.

### 共进Shanghai Gongjin Communications Technology:

* Products:
* GPON router, GPON switch and STB products.
* Job position (Embedded software engineer)
* I worked on integrating boot loader, Linux kernel bring-up, file system customization.
* And I also worked on integrating Linux device drivers such as SDRAM, Nand/nor flash, USB, ethernet driver and so on.
* Moreover, I participated in the software development and troubleshooting on the different products.
* When I work on a new project:
* firstly, I go through the hardware schematic to confirm how should I modify the bootloader and what are the new device driver I need to port.
* For the bootloader, I usually modify the start.s for the CPU and SDRAM configuration.
* I also port the NAND driver/network driver in the bootloader if we use new flash/network mac&phy chip in the new board.
* And then I’ll do some functionality test on the bootloader and drivers.

### 龙爵Zhuhai Longjue Technology:

* Products:
* Video door phone system
* when people press a button on the door, then we can see them through the camera, and we can communicate with them via microphone and speaker.
* Job position
* I worked on the BSP development and Linux bring-up, file system.
* I also worked on the trouble-shooting of the product and the software maintenance.
* For the video, the Chip manufacturer provided driver, we implemented the software application for the video.

## What are the most challenging problems/issues you debugged before?

### BSP:

I encountered a problem during the debugging of bootloader before. The board starts to run from a new NAND flash, but I observed there is NO console output after I reprogrammed the flash and power on the board. This issue can be caused by software or hardware. To confirm if it’s hardware related issue, I need a JTAG Debug tool, like BDI. But I have not gotten the tool from CPU manufacturer, and the project schedule is urgent, so I soldered a LED on the board instead, and I tried to trace down the code by controlling the LED on/off to debug the issue.

I observed we can NOT control the LED in the assembly code in the start.s successfully, so I suspect it is hardware issue related the NAND flash.

Then I checked the schematic and found out the electric capacity which is soldered on the power pin for NAND flash is incorrect. It causes the unsuccessful power up sequence on the NAND flash.

After we replaced the electric capacity, the board can start from NAND flash successfully.

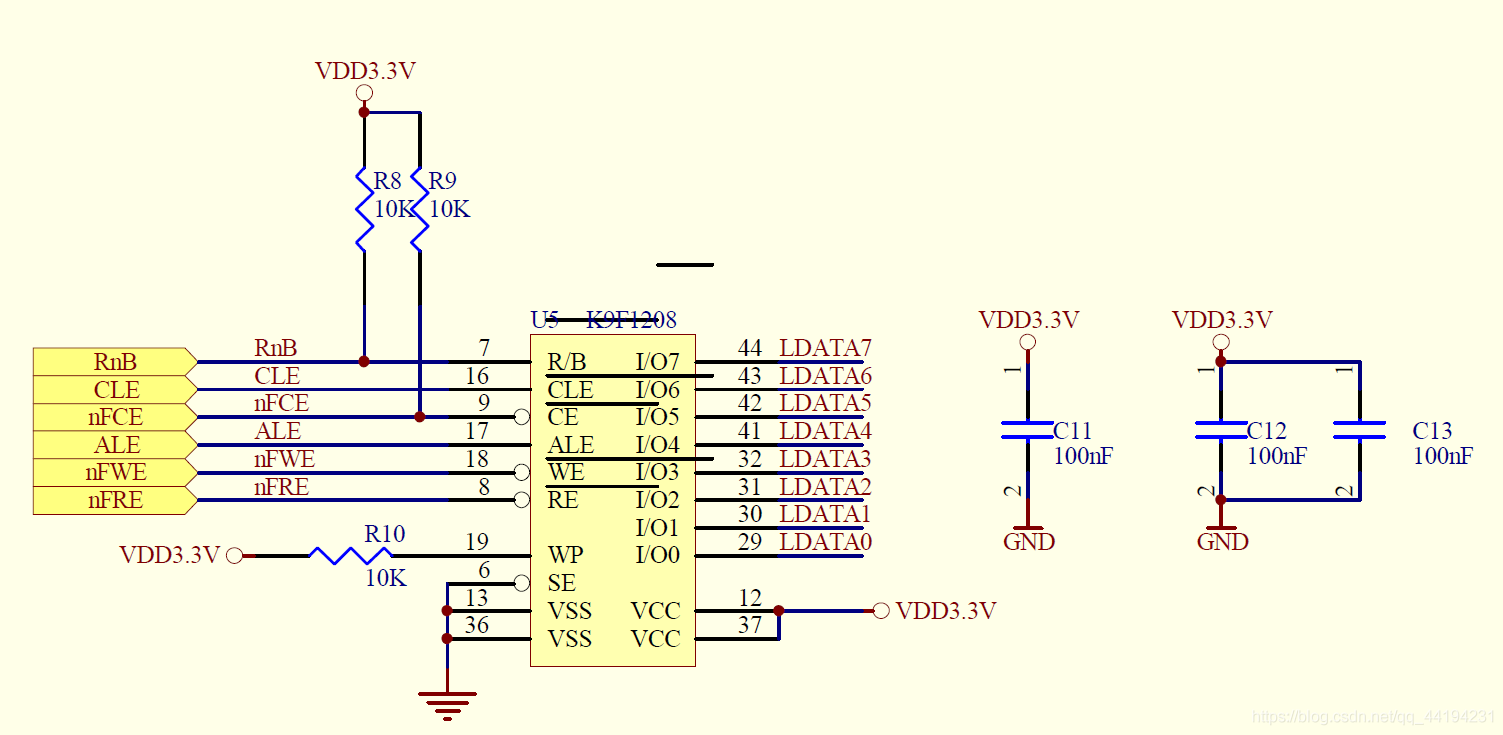
More detailed explanation:

The value of electric capacity is greater than the needed value. So, the NAND flash powers up was delayed, and the power up sequence is slower than what is expected. When steppingstone **mechanism**[ˈmɛkəˌnɪzəm] is **executed**[ˈɛksɪˌkjut] at CPU side to copy the first 4k file from NAND flash to CPU, the NAND flash is NOT power up yet, so the code copying process is failed.

We soldered an adapter on the board, then we used the programmer to program the NAND, and put the NAND on the adapter.

A screenshot of a computer

Description automatically generated with low confidence



A picture containing text, screenshot, number, font

Description automatically generated

**[Julie’s version] One of the challenging issues that I debugged before is debugging new NAND flash for bootloader. After I programmed the bootloader image into the new NAND flash, there was no console output after I powered up the board.**

**It could be a hardware issue or a software issue.**

**To confirm whether it’s a hardware issue, usually I need to check board hardware status - power supply, nand flash chip selection, WP(write protect), clock signal and data signal timing diagram（时钟和数据的时序图）.**

**For software debugging, usually I use Jtag tool like BDI to debug the bootloader code, but I did not get the tool. I came up an idea by soldering a LED in the board, I think I can move forward the debugging by controlling the LED on and off.**

**Then, I observed I was not able to control the LED on or off in the start.S code, I suspect it is a hardware issue.**

**After I checked the schematic document, I found out the electric capacity soldered to the power pin of the NAND flash is incorrect which caused the issue. After we replaced the electric capacity, then the board can boot up from NAND flash successfully.**

**If the interviewer shows his interest on our talking, then we can explain more which shows our deep understanding about what happened:**

**The value of the electric capacity is bigger than the required value which makes the power-up sequence of the NAND flash slower than what is needed.**

**During the steppingstone process, the first 4K code will be copied from NAND flash to CPU, since the NAND flash is up slower than expected, the copying process fails.**

### **Network driver:**

**To-do: I need to summarize the network driver.**

**Issue1:**

We connect the ethernet interface from our board to another device. The LED shows the link is up, but there’s no data packets are transmitted between the connected devices.

During the debugging, we enabled **loopback** in the MAC layer to narrow down the root cause. We also can use smartbits to debug the issue.

We found out the auto-negotiation is enabled at our board, but it’s disabled on another device.

We can see there is a register shows the auto-negotiation is not completed.

So, after we enable the auto-negotiation on another device, the issue is resolved.

A picture containing text, screenshot, font, design

Description automatically generated

I have an ethernet protocol book talking about the auto-negotiation and other important details about ethernet MAC layer, we need to find out that book.

**Issue2:**

**The resistor on the GMII interface is incorrect causing a lot of error frames which has CRC failure.**

During the debugging, we enabled loopback in MAC layer to narrow down the root cause. We also can use smartbits to debug the issue.

* Details:

**EECS** --> 控制总线: high-->8bit; low-->16bit

this pin connects with a resistor to configure the internal memory data bus width.



**Network driver hardware interface (PHY/MAC):**

I think we need to understand below items:

What’s PHY layer? What does the PHY layer do?

What’s MAC layer? What does the MAC layer do?

What’s the MII interface for? The high-level pin connection between PHY🡨-🡪MAC? Why do we have MII interface?

**We can refer to below documents:**

https://blog.csdn.net/u013372900/article/details/123194467

### **Deadlock issue:**

I worked on a software application layer issue for the GPON product. The issue is the system seems like get stuck after running our application after 8 hours, and the console does not print out any information. And we also cannot input any debug command via the console. I suspect we may have deadlock in the kernel layer.

First at all, I tried to reduce the issue happening time to 5 minutes(how??).

Then, I used magic key which provided by Linux kernel to debug the issue.

I used menuconfig to enable the magic key option in the kernel and rebuild the kernel image.

When the issue happens, then I can input some specific key to print out the CPU stack and the backtrace.

By analyze the stack/backtrace address, and used the symbol table on the vmlinux, we can locate the deadlock in the code.

Finally, we figured out the specific line in the code which causes the issue. It was an infinite loop in driver code.

# interview questions:

## The differences between thread and process(done in kernel section)?

The common characteristic of thread and process is they are both running tasks in the Linux system.

The

## Hash table: how does it work?

1). What is it?(Talk about category)

**A hash table is a data structure.**

2) How does it work?

**It works by using a hash function to map the key to an index in an array, and then stores the value at that index or retrieve data from that index.**

3). It is used to store keys/values in a way that allows for efficient data insertion and retrieval.

Diagram

Description automatically generated

## Priority Que: how to get the node asap? Binary heap ( I found an article, need to learn it)

## Interrupt: top halves and bottom halves （summarized in kernel section）

Softirq vs tasklet vs irq vs work queue (will-do)

工作队列的执行上下文是内核线程，因此可以调度和睡眠。为什么？

Tasklet是基于软中断实现的，因此也可以运行于软中断上下文。

## Usb hot plug原理 ( I found an article)

## Linux device driver flow ( see driver section)

## Tcp udp packet format (I found an article)

Beacon frame vs tcp (I’ll summarize in WiFI document)

## Strcpy vs memcpy (done)

Strcpy will stop at ‘\0’(NULL character)

Memcpy will copy by length.

## How does free() know the size of malloc (copied)

Most implementations of C memory allocation functions will store accounting information for each block, either in-line or separately.

One typical way (in-line) is to actually allocate both a header and the memory you asked for, padded out to some minimum size. So for example, if you asked for 20 bytes, the system may allocate a 48-byte block:

16-byte header containing size, special marker, checksum, pointers to next/previous block and so on.

32 bytes data area (your 20 bytes padded out to a multiple of 16).

The address then given to you is the address of the data area. Then, when you free the block, free will simply take the address you give it and, assuming you haven't stuffed up that address or the memory around it, check the accounting information immediately before it. Graphically, that would be along the lines of:

\_\_\_\_ The allocated block \_\_\_\_

/ \

+--------+--------------------+

| Header | Your data area ... |

+--------+--------------------+

^

|

+-- The address you are given

Keep in mind the size of the header and the padding are totally implementation defined (actually, the entire thing is implementation-defined (a) but the in-line accounting option is a common one).

The checksums and special markers that exist in the accounting information are often the cause of errors like "Memory arena corrupted" or "Double free" if you overwrite them or free them twice.

The padding (to make allocation more efficient) is why you can sometimes write a little bit beyond the end of your requested space without causing problems (still, don't do that, it's undefined behaviour and, just because it works sometimes, doesn't mean it's okay to do it).

## System design (copied)

A server connects to 10000 clients,

Server checking client status: alive?

Server sends inquiry commands to clients: if 3 times no response, then delete the client.

List those clients who are NOT alive.

Server deletes client.

design data structure for these requirements.

Answer: TODO

* Hash table to save 10000 clients: 1. Status; 2.
* A linked list to save all alive clients

# ToDO

1. Process status
2. Ulimited -c command: what is this command for?

A screenshot of a computer screen

Description automatically generated

# BSP (priority):

## (done) Summarize the UBOOT Board bring-up?

The Uboot mainly includes two stages.

During the first stage:

* Nand flash, CPU will copy 4Kb to Steppingstone, which is a Boot RAM, and run bootloader in the Boot RAM.
* Run from start.S which is written in assembly language, since the stack is not set up.
* Initialize CPU including setting the CPU mode to SVC mode; turn off the cache/watchdog/interrupt/MMU, set Clock and so on.
  + Why turn off cache? Because there is no data in cache, turn it on will cause fetch data exception.
  + Why turn off MMU? Because now accessing physical address, don’t need map physical address to virtual address.
* And then we flush I/D cache.
* And then we initialize the SDRAM.

How to initialize it? Just like other devices,

(1). Power on.

(2). CS on/chip select on.

(3). set up clock.

(4). Setup data transmission mode, rate.

(5). Latency…

* And then set up the memory map including the stack section, heap section, BSS section and so on.
* Nor Flash: Copy the bootloader program from FLASH to SDRAM.
* Then we jump into C program.

During the second stage:

* Initialize the hardware devices we want to use such as serial port, flash, Network PHY and MAC.
* Copy the kernel and rootfs image from FLASH to SDRAM.
* Set the boot parameters to boot up kernel.
* Start the kernel.

## (done)How does the Bootloader pass the parameters to the kernel?

tagged list

The boot loader provides a tagged list for passing configuration data to the kernel

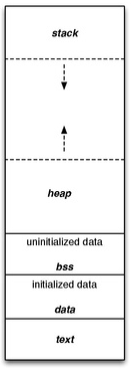
* the configuration data address and size;
* memory starts address and size;
* and the information on which mtd partition has the root filesystem;
* console kernel can use.

Bootargs

#define CONFIG\_BOOTARGS "noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0 mem=128M"

DTS

## Memory Data segment(done)



The .data segment contains any global or static variables.

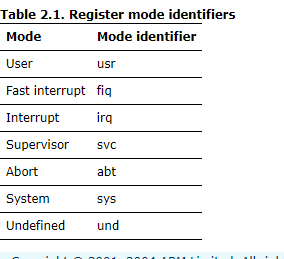
Static variables are allocated memory in data segment. It remains in memory while the program is running

## ARM RISC architecture?

RISC architecture provides powerful instructions that execute within in a single cycle.

(The RISC design concentrates on reducing the complexity of instruction performed by hardware)

## 7 CPU modes for ARM?



User mode: It is used for executing application programs

System mode: It is a privileged user mode for the operating system.

Supervisor mode: It is a protected mode for operating system. When the system resets, and SWI instructions is executed, CPU enters SVC mode.

Fast interrupt (FIQ): Can be used for data transfer or channel process. CPU enters FIQ when an external device sets the FIQ pin.

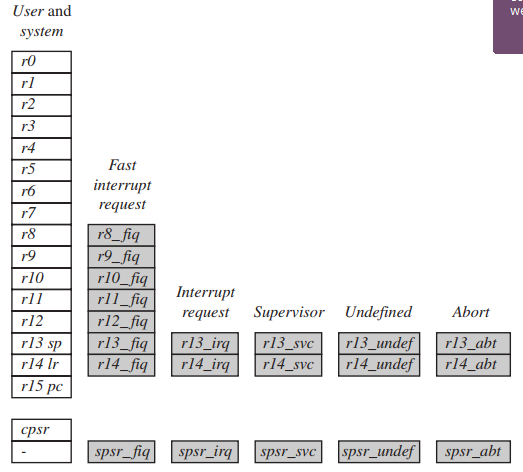
IRQ mode: It is used for general-purpose interrupt handling. CPU enters IRQ when an external device sets the IRQ pin.

Abort mode: It is entered after a data or instruction Prefetch Abort. (e.g., Try to access an invalid physical address or write the memory address without the access permission)

Undefined mode: It is entered when an undefined instruction is executed.

Note: Useful link: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/Cihhcjia.html

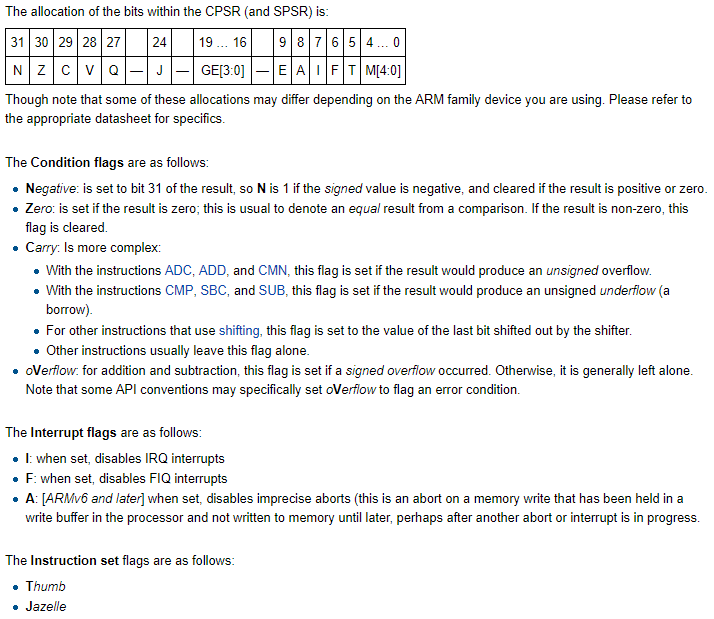
## ARM CPU registers(done)?



## Current Program Status Register(done)

The Current Program Status Register (CPSR) holds:

* the APSR flags
* the current processor mode
* interrupt disable flags
* current processor state (ARM, Thumb, ThumbEE, or Jazelle®)
* endianness state (on ARMv4T and later)
* execution state bits for the IT block (on ARMv6T2 and later).



## CPU I used in previous projects?(done)

BCM5871X Quad-core 64-bit 2GHz ARM Processor

BROADLIGHT BL2348 GPON MIPS Processor

SUMSUNG S3C6410 ARM11 Processor

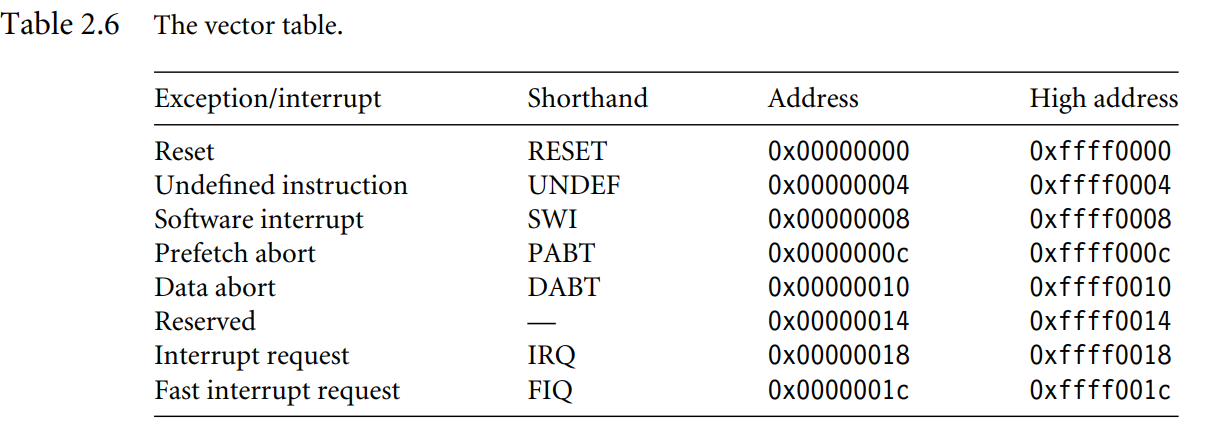
## What’s the vector table? (done)

The interrupt vector points to the ISR (developer writes the interrupt hander).

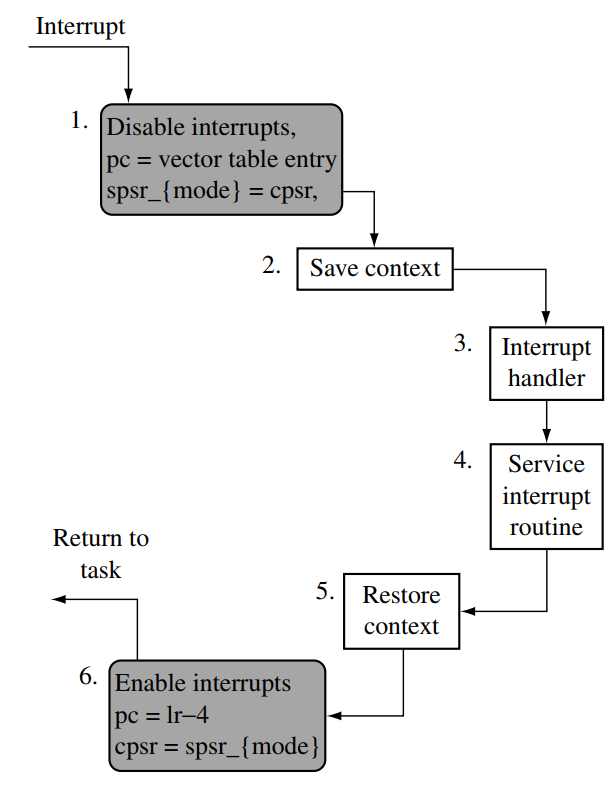
The vector table includes exception vectors for all exception handlers.

Each entry in the vector table has the instruction that branches to specific interrupt handler routine which handles specific exception or interrupt.

The address 0x00000000 in the memory map is reserved for the vector table.



## Explain ARM interrupt switch (register level) (done)?



Save context:

* Save the current **PC** into lr.
* Store the value of **r0-r12** and lr into stack.

Restore context:

* Pop out the **r0-r12** and lr from stack.
* **PC** = lr -4

## 中断嵌套 interrupt nest (done)

Linux is capable of handling nested interrupts.

In general, when a processor receives an interrupt request, it stops executing the current instruction and begins executing an interrupt service routine (ISR) associated with the interrupt. Once the ISR has completed, the processor returns to the instruction that was interrupted and continues execution.

## Sleep in Interrupt? （done）

**NO!**

**In general, Linux kernel does not allow sleeping in an interrupt context, because sleeping requires releasing the CPU and scheduling other tasks, which is not allowed in an interrupt context.**

会导致无法唤醒：

中断处理的时候,不应该发生进程切换，因为在中断context中，唯一能打断当前中断handler的只有更高优先级的中断，它不会被进程打断，如果在中断context中休眠，则没有办法唤醒它，因为所有的wake\_up\_xxx都是针对某个进程而言的，而在中断context中，没有进程的概念，没有一个task\_struct（这点对于softirq和tasklet一样），因此真的休眠了，比如调用了会导致block的例程，内核几乎肯定会死。

会导致上下文错乱：

睡眠函数会调用schedule()，切换进程时，保存当前的进程上下文（CPU寄存器的值、进程的状态以及堆栈中的内容），以便以后恢复此进程运行。中断发生后，内核会先保存当前被中断的进程上下文（在调用中断处理程序后恢复）；但在中断处理程序里，CPU寄存器的值肯定已经变化了吧（最重要的程序计数器PC、堆栈SP等），如果此时因为睡眠或阻塞操作调用了schedule()，则保存的进程上下文就不是当前的进程context了，所以不可以在中断处理程序中调用schedule()。

## What is mmu and what does it do? (done)

Basic concepts:

* Logical address (virtual address): generated by the CPU
* Physical address: address seen by the memory unit
* MMU: hardware device that maps virtual address to physical address.
* Logical and physical addresses are the same in the compile-time and load-time address-binding schemes.
* Logical(virtual) and physical addresses differ in execution-time address-binding scheme.
* Each process behaves as if it has free use of the entire CPU and 4G Bytes memory
* The drawback that there is an overhead – the MMU remapping – on every context switch.

The Memory Management Unit (MMU) is required by the Linux operating system for several reasons:

* Memory Protection: The MMU provides a mechanism for protecting the memory of one process from being accessed by another process. This is essential for maintaining the isolation and security of individual processes running on a Linux system.
* Virtual Memory: The MMU provides the mechanism for implementing virtual memory, which allows the Linux operating system to provide each process with the illusion of having a large, private memory space. Virtual memory allows processes to access memory beyond the physical memory available on the system, by swapping out pages of memory that are not currently in use to disk.
* Paging and Swapping: The MMU is responsible for translating virtual memory addresses used by a process into physical memory addresses. This is done through a process called paging, which divides the virtual memory space into fixed-size blocks called pages and the physical memory into fixed-size blocks called page frames. The MMU uses a page table to map each virtual page to a physical page frame.
* Address Space Layout Randomization (ASLR): The MMU provides a way of randomizing the memory layout of a process, which makes it more difficult for an attacker to predict the location of sensitive data in memory and therefore makes it harder for them to exploit vulnerabilities.
* Memory sharing: The MMU allows multiple processes to share the same memory regions, which is beneficial for resource sharing and for memory usage efficiency.
* Lazy allocation: The MMU allows the kernel to implement a technique called 'lazy allocation', which is a technique for allocating memory to a process on an as-needed basis, rather than allocating all the memory to the process at the time it starts. This reduces the amount of physical memory that is required to run a process and improves memory usage efficiency.

All these features provided by the MMU are essential for the proper functioning of a Linux operating system, which allows to run multiple concurrent processes, provide isolation and security, manage memory resources efficiently, as well as providing a better performance and stability.

## Describe the kernel starts up? (done)

First stage – head.S:

* Check whether kernel supports the specific **CPU architecture**
* Check whether kernel supports the specific **machine type** (board type)
* 🡪 **Create page tables**🡪 disable Data/instruction cache🡪 enable MMU🡪 set up stack pointer -> call start\_kernel()

Second stage – start\_kernel():

* Print out Linux version🡪 call **setup\_arch**() to configure the environment related to the specific architecture🡪initialize the console
* 🡪 start **init** process(linuxrc)

Modify the MTD partition (not sure it belongs to the first stage?)

Modify the mtd\_partition data structure by specifying the name, size, offset of each file system.

1. **BIOS**: The first stage of the boot process is the BIOS (Basic Input/Output System). The BIOS performs a series of basic hardware checks and initializations and then locates and loads the boot loader.
2. **Boot loader**: The boot loader's main function is to load the Linux kernel into memory and then transfer control to it. Some popular boot loaders in Linux include GRUB (GRand Unified Bootloader) and LILO (LInux LOader). The boot loader is responsible for allowing the user to select which operating system or kernel to boot and also allows to pass options to the kernel.
3. **Kernel initialization**: Once the kernel is loaded, it begins its initialization process. The kernel performs a series of low-level hardware detection and configuration steps and then proceeds to mount the root filesystem.
4. **Initial RAM disk**: The kernel may use an initial RAM disk (initrd) which is a small filesystem that is loaded into memory at the beginning of the boot process. The initrd contains drivers that are needed to mount the real root filesystem, so it can provide the kernel with the necessary kernel modules to be able to access the root filesystem.
5. **Root filesystem**: Once the kernel has determined the location of the root filesystem, it mounts it and switches the root filesystem to it. The root filesystem contains all the necessary system files, including the programs and scripts that are required to start the system.
6. **Init process**: Once the root filesystem is mounted, the kernel starts the init process, which is the first process that is started on the system. The init process is responsible for starting all other processes, including system services and daemons. The init process reads the configuration files located in /etc/inittab, /etc/init.d/ and other directories.
7. **System initialization**: Once the init process starts, it begins the process of initializing the system, by reading configuration files, starting system services and daemons. Services such as Network, SSH, and other daemons are started, and the system is ready for user login.
8. **User login**: Once the system initialization is complete, the system prompts the user to login and start using the system.

## File system customization? (done)

We use busybox to create different directories such as the executable programs under /bin, /sbin and so on.

Then we manually create other directories;

Then we use mkyaffsimage tool to generate the yaffs file system image; then we can download the yaffs file system image into Flash; we can mount the specific mtd partition from the flash and use the file system.

## The differences between stack and heap(done)

①管理方式：栈由编译器自动管理；堆由程序员控制，使用方便，但易产生内存泄露。

②生长方向：栈向低地址扩展(即”向下生长”)，是连续的内存区域；堆向高地址扩展(即”向上生长”)，是不连续的内存区域。这是由于系统用链表来存储空闲内存地址，自然不连续，而链表从低地址向高地址遍历。

③空间大小：栈顶地址和栈的最大容量由系统预先规定(通常默认2M或10M)；堆的大小则受限于计算机系统中有效的虚拟内存，32位Linux系统中堆内存可达2.9G空间。

④存储内容：栈在函数调用时，首先压入主调函数中下条指令(函数调用语句的下条可执行语句)的地址，然后是函数实参，然后是被调函数的局部变量。本次调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的指令地址，程序由该点继续运行下条可执行语句。堆通常在头部用一个字节存放其大小，堆用于存储生存期与函数调用无关的数据，具体内容由程序员安排。

⑤分配方式：栈可静态分配或动态分配。静态分配由编译器完成，如局部变量的分配。动态分配由alloca函数在栈上申请空间，用完后自动释放。堆只能动态分配且手工释放。

⑥分配效率：栈由计算机底层提供支持：分配专门的寄存器存放栈地址，压栈出栈由专门的指令执行，因此效率较高。堆由函数库提供，机制复杂，效率比栈低得多。Windows系统中VirtualAlloc可直接在进程地址空间中分配一块内存，快速且灵活。

⑦分配后系统响应：只要栈剩余空间大于所申请空间，系统将为程序提供内存，否则报告异常提示栈溢出。

     操作系统为堆维护一个记录空闲内存地址的链表。当系统收到程序的内存分配申请时，会遍历该链表寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点空间分配给程序。若无足够大小的空间(可能由于内存碎片太多)，有可能调用系统功能去增加程序数据段的内存空间，以便有机会分到足够大小的内存，然后进行返回。，大多数系统会在该内存空间首地址处记录本次分配的内存大小，供后续的释放函数(如free/delete)正确释放本内存空间。

     此外，由于找到的堆结点大小不一定正好等于申请的大小，系统会自动将多余的部分重新放入空闲链表中。

⑧碎片问题：栈不会存在碎片问题，因为栈是先进后出的队列，内存块弹出栈之前，在其上面的后进的栈内容已弹出。而频繁申请释放操作会造成堆内存空间的不连续，从而造成大量碎片，使程序效率降低。

     可见，堆容易造成内存碎片；由于没有专门的系统支持，效率很低；由于可能引发用户态和内核态切换，内存申请的代价更为昂贵。所以栈在程序中应用最广泛，函数调用也利用栈来完成，调用过程中的参数、返回地址、栈基指针和局部变量等都采用栈的方式存放。所以，建议尽量使用栈，仅在分配大量或大块内存空间时使用堆。

     使用栈和堆时应避免越界发生，否则可能程序崩溃或破坏程序堆、栈结构，产生意想不到的后果。

# Drivers

## Device driver (done)

Character device driver

* 1. We use register\_chrdev function to link the driver with its corresponding /dev file in kernel space. It’s called with three arguments: 1) major number, 2) a string of characters showing the module name, 3) a file\_operations structure which links the call with the file functions it defines.
* 2. Using ioremap function to map the hardware address from physical address to virtual address.
* 3. Then we can write read(), write(), ioctl() functions by using copy\_from\_user/copy\_to\_user functions to access these ioremaped hardware address.

code

struct file\_operations mydev\_fops = {

owner : THIS\_MODULE ,

open : mydev\_open ,

release : mydev\_release ,

read : mydev\_read ,

write : mydev\_write ,

};

static int first\_drv\_write(struct file \* file, const char \_\_user \* buffer, size\_t count, loff\_t \* ppos) {

copy\_from\_user(&val, buffer, count);

}

static long beep\_ioctl(struct file \*filep, unsigned int cmd, unsigned long arg) {

switch(cmd) {

case BEEP\_ON:

fs4412\_beep\_on();

break;

}

}

static int \_\_init init\_mydevDriver ( void ) {

…

result = register\_chrdev (240 , " mydev " , & mydev\_fops )) < 0

gpbcon = (volatile unsigned long \*)ioremap(0x56000010, 16);…

}

static void \_\_exit end\_pp ( void ) {

unregister\_chrdev (240 , " mydev ");

iounmap(gpbcon);

}

int main () {

..

fd = open ("/ dev / mydev " , O\_RDWR );

buffer [0]=0 x00;

write ( fd , buffer ,1 , NULL );

read ( fd , buffer ,1 , NULL );

printf (" Value : 0 x %02 x \ n " , buffer [0]);

close ( fd );

}

* Build the “vram” driver (video\_ram.ko file) by running make with a changed Makefile.
* Load the driver using insmod video\_ram.ko.
* **Write** into /dev/vram, say, using **echo** -n "0123456789" > /dev/vram.
* **Read** the /dev/vram contents using **od** -t x1 -v /dev/vram | less. (The usual cat /dev/vram can also be used, but that would give all the binary content. od -t x1 shows it as hexadecimal. For more details, run man od.)
* Unload the driver using rmmod video\_ram.

Platform

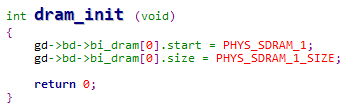
## SDRAM driver(done)

How to access the specific address in the SDRAM:

* Chip select 🡪 Bank select 🡪 Row select(RAS) 🡪 Column select(CAS)
* Based on the given specific address, CPU enables the SDRAM chip select signal first, then sending out the bank select signals, and then enable the row select signal, and then enable the column select signal, then read or write the specific address (for write operation, use WE to enable write).

Given a new SDRAM, how to customize the SDRAM driver code in the bootloader(uboot)?

* In the lowerlevel\_init.s, we need to configure the registers which in the SDRAM bank controller on hardware access sequence and auto refresh frequency.
* dram\_init (): configuration available sdram banks

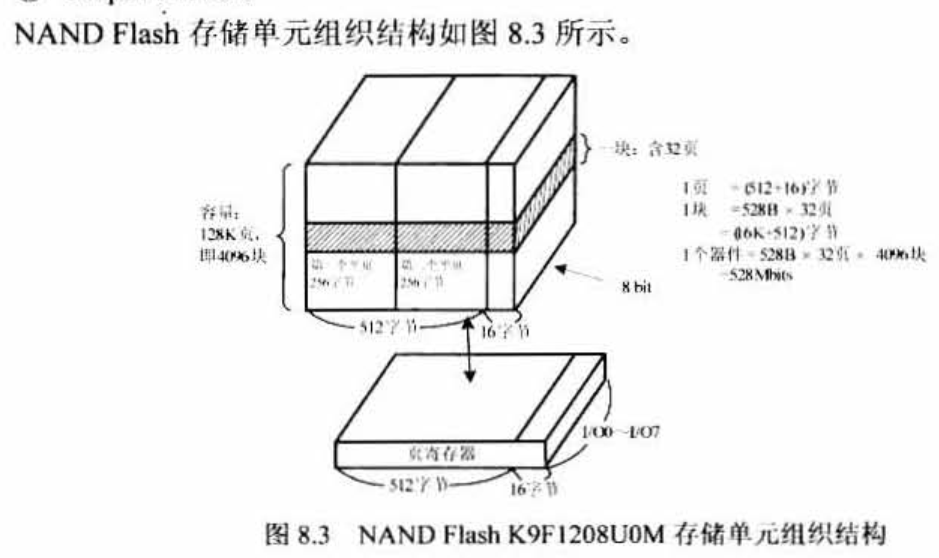


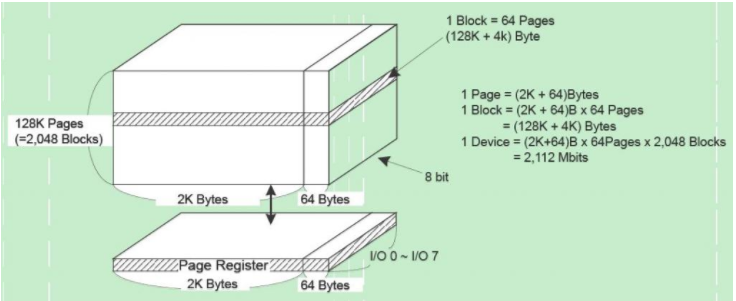
## NAND flash(done)

NAND flash vs NOR flash driver:

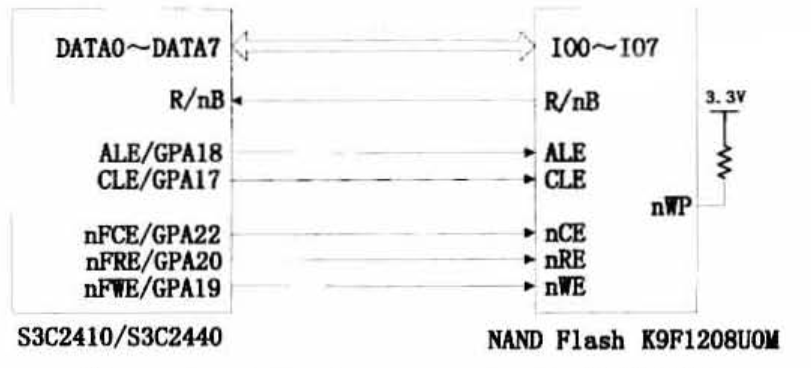
|  |  |  |
| --- | --- | --- |
|  | NOR Flash | NAND Flash |
| Support XIP? | Yes, can run program in NOR flash | NO, cannot run program in NAND flash |
| Performance | Erase/ Write: slow (5s)  Read: fast  Suitable to store program or important data | Erase/Write/read: fast  Suitable to store data |
| Reliability | High | Low (may have bad blocks) |
| Finite number of program | 10,000~100,000 | Ten times of NOR flash (100,000~1000,000) |
| Interface | Same with RAM | I/O interface  Data and address using the same bus |
| Access method | Random access | Sequential access |
| Price | High | Low |
| File system | Usually use jffs2 | Usually use yaffs |

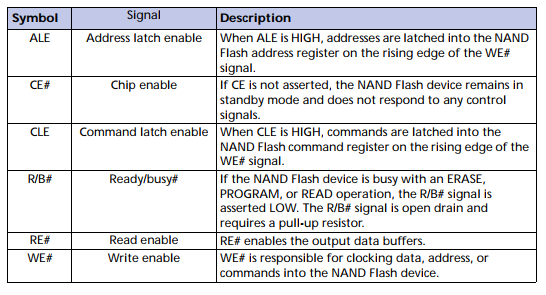
### NAND flash driver summary(done)





* 1 chip consists of blocks-🡪 block consists of pages
* 1 page=512 bytes + 16 bytes





### For the operations on NAND flash: (done)

Transfer command first🡪 then transfer address-🡪 then read/write data

During the process, we need to check the status of NAND flash.

* **We can configure NAND flash controller for NAND flash operation:**

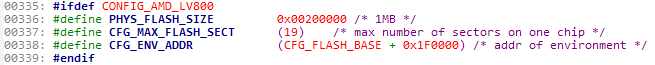
- Configure configuration register (NFCONF) to configure NAND flash.

- Write command into command register (NFCMD: read/write/erase…)

- Write address into address register (NFADDR)

- Read/write data, check the NAND flash status by accessing status register.

* flash\_init()



* nand\_init -> board\_nand\_init -> add\_mtd\_device

Given a new NAND chip, how to modify related code in bootloader(uboot)?

* We need to modify related code in bootloader **to specify the block size/page size/max number of sector/chip size** on one chip, since the write operation is based on page size; and erase operation is based on block size.
* Moreover, we also need to modify related functions like how to select chip, how to read/write/erase NAND flash and check status of NAND flash and so on based on using registers from NAND controller.
* NAND controller provides the command register/address register/data register/status register. For all operations on NAND flash, we use the sequence which is we send the command to the chip first (for example, block erase, page program and so on), and then we send the address to the chip, and then we do the read/write/erase operation. During the process, we need to check the status of the Flash.
  + So, we customize the related functions for read/write/erase and other operations on the specific chip.

How to add a new NAND chip under Linux?

* Firstly, we allocate a nand\_chip structure and a storage for the ioremap chip address.
* Then we fill out some board dependent members in the nand chip description structure like chip size, block size (use nand\_scan\_ident function to read hardware information).

The functions on different operations including read operation/write operation/erase operation and so on.

* If we want to divide the flash into partitions, then define mtd\_partition structure to specify all paritions.
* Define a mtd\_info structure and pointer the priv field to the nand\_chip strcture;
* Call add\_mtd\_partitions()/add\_mtd\_device() to register the NAND flash device/partition mtd subsystem(the device/partition will be added in mtd\_table), so that file system/application program can access the flash/specific partition in the user layer.

Debug issues on porting NAND flash driver?

* We can read/write flash under bootloader, but cannot find flash after we boot up Linux
  + we used the Oscilloscope to measure the signals, we found the driver does not send the correct hardware sequence on accessing the NAND flash chip. Because kernel does not read **correct flash ID** from the chip, after we modify the NAND hardware sequence in the driver, kernel can detect the NAND flash chip.
* After we erase a specific partition, the system cannot boot up
  + During the debugging, I found out the block0 which has the bootloader has been erased, since if I use JTAG to read out the data, it is all 0xFF. I found the reason is that we did not calculate the address properly in the driver.
  + After we modify the driver for the new chip, then the issue is resolved.
* An example:
  + The s3c2442 has signals OM[0:1].
  + If OM[0:1]==00 on system reset the steppingstone is executed and the first 4k of NAND flash are copied to a special portion of SRAM (the stepping stone) which is mapped to memory addresses 0x0000\_0000...0x0000\_0fff. Then executions with %pc = 0x0000\_0000.

NAND flash bad block (OOB detect):

* ECC校验
  + 由于Nand flash的工艺特性，所以nand flash有一个缺点就是位反转。所以加入了ECC校验。具体怎么实现呢？
  + nand flash的存储是以页为单位的，它在每页的后面加入了OOB（16字节），这里面存的就是ECC的值。如何工作？
    1. 写每页的时候生成ECC，将ECC写入OOB
    2. 当读每页的时候先算出ECC，然后读OBB的ECC，两个ECC进行比较。

## NOR Flash driver summary(done)



Initialize the map\_info structure with specifying the chip name, chip size, bankwidth and so on.

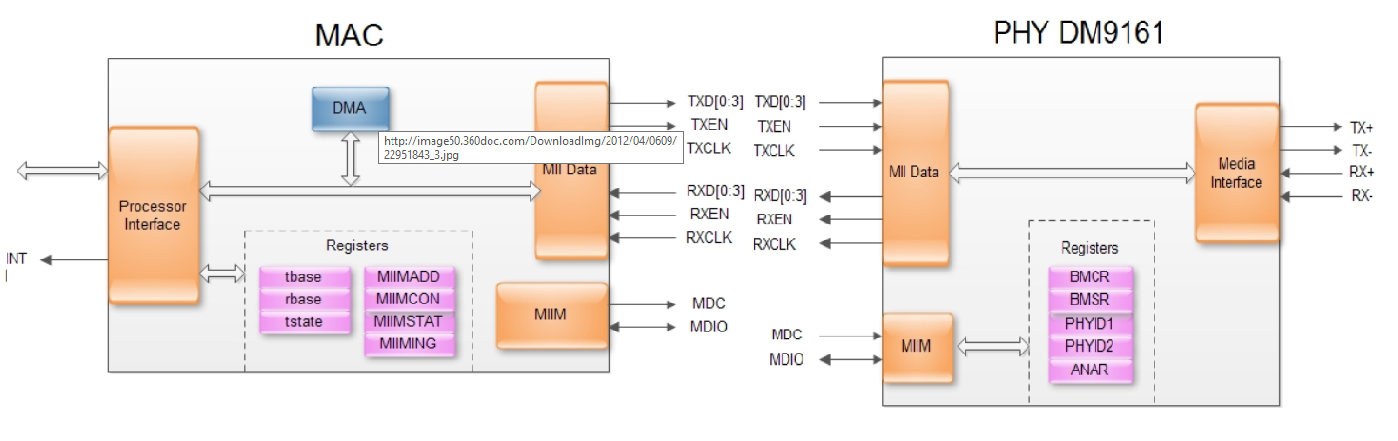
And then call do\_map\_probe() to fetch mtd\_info which includes the functions for different operations like read/write/erase and so on.

If we want to use partitions on the NOR chip, then we can define mtd\_partition structure with specifying the partitions.

Then we call add\_mtd\_partition() or add\_mtd\_device() to register the partition/chip to mtd subsystem.

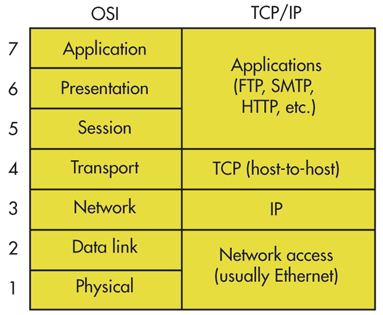
## Ethernet Network driver summary (done)

Hardware architecture:

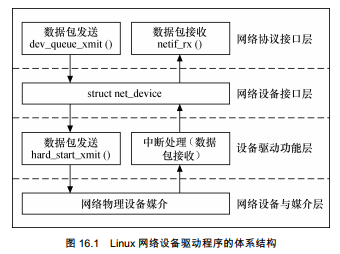
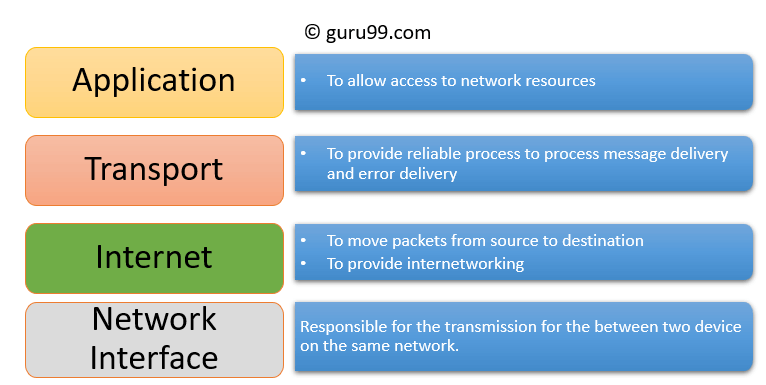


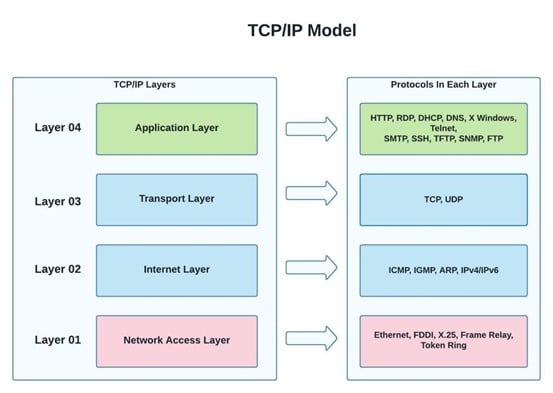
* Network device consists of 2 parts: MAC (Data Link) and PHY (Physical layer).
* The MII interface used to transmit data between MAC and PHY.
* The MDIO interface is the management interface between MAC and PHY (MDC: clock line, MDIO: bi-directional data line)
* CPU needs to use MDIO interface to access the PHY registers.
* CPU use DMA to send/receive data.

TCP/IP layer vs OSI layer



* Data link layer has 3 basic functions
  + **Framing**: 封装成帧就是在一段数据前后分别添加首部和尾部。接收端以便从收到的比特流中识别帧的开始与结束，帧定界是分组交换的必然要求.
  + **Pass-through**: 透明传输避免消息符号与帧定界符号相混淆.
  + **Error detection**: 差错检测防止差错的无效数据帧，浪费网络资源。

How to port driver on a new network device under Linux



**TCP** - stands for Transmission Control Protocol

**IP** - stands for Internet Protocol

* We need to allocate the net\_device structure for the new network device.
  + fill out its members including **device** **name**, **irq**, **base\_addr**,
  + fill out operation functions such as open, hard\_start for sending out data ioctl and rx interrupt routine for receiving data.
* In the open () function, we need to initialize the mac and phy.
  + configure speed, enable/disable auto-negotiation, enable/disable flow control and so on.
* **For sending out the data packet:**
  + We need to fill out hard\_start\_xmit () function.

The IP layer puts the data packets in the **sk\_buff**, we need to move the data from **sk\_buff** to the **DMA** memory (The CPU uses DMA for the data transmission) to start the data transmission.

* **For receiving data packets:**
  + After the DMA receives new packets to memory, it will trigger the interrupt.
  + we implement the interrupt handler routine to receive the data packets; and put the data packets in skb\_buff.
  + And then call netif\_rx () to pass the data packets to the upper layer.
  + Up layer uses dev\_queue\_xmit () to send out the data packets

Other important details:

* netif\_start\_queue (): allow transmit; this function allows upper layers to call the device hard\_start\_xmit () function.
* netif\_stop\_queue (): stop transmitted packets; Stop upper layers calling the device hard\_start\_xmit routine. Used for flow control when transmit resources are unavailable (such as too busy).
* netif\_carrier\_on (): Device has detected that carrier
* netif\_carrier\_off (): Device has detected loss of carrier.

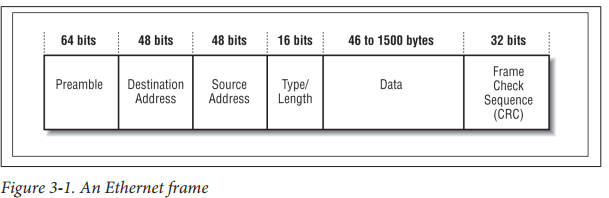
CSMA/CD protocol:

* Multiple computers share access to a single Ethernet channel by using CSMA/CD (carrier sense multiple access with collision detection) protocol.
* Each interface must wait until there is no signal on the shared channel before it can begin transmitting data.
* If multiple interfaces detect the channel is idle and start transmitting their frames, then ethernet signaling devices send out the collision of signals, the signals are used to tell the ethernet interface to stop transmitting.
* Each interface will then choose a random retransmission time to resend the frames (called back off)

Work mode

* Half-duplex mode:
  + only one computer can send data over ethernet cable at any given time. In half-duplex mode,
  + To send data in half-duplex mode, a station first listens to the channel, and if the channel is idle, then the start starts to transmit the data.
* Full-duplex mode:
  + Nowadays, the ports on ethernet switch are using full-duplex mode.

Ethernet frame:



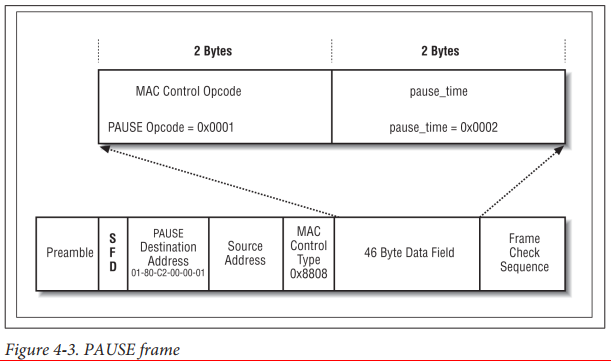
* Preamble: to give the hardware some start-up time to recognize that a frame is being transmitted, notify it to receive data.
* Type/length: the type of high-level network protocol.

**ARP** protocol (Address Resolution Protocol):

* ARP is used for mapping an IP address to a physical address (MAC address) by broadcasting ARP probe packets.

Ethernet Flow Control:

* An interface can send a pause frame to ask the sender to pause the frame transmission.



Auto-negotiation:

* The auto-negotiation system allows the ethernet devices automatically negotiate their configuration to the highest set of common capabilities (speed. Duplex mode etc.).

https://stackoverflow.com/questions/47450231/what-is-the-relationship-of-dma-ring-buffer-and-tx-rx-ring-for-a-network-card

Diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

## Router vs switch (done)

|  |  |
| --- | --- |
| Router | Switch |
| Layer 3 – network | Layer 2 - data link |
| Transport data by using IP | Transport data by MAC address |
| Can find best rout way by using rout protocol / algorithm | Just send the data to the whole network |
|  |  |

## Touch panel driver (done)

Description on the resistive touchscreen

It has two flexible sheets with a resistive material and separated by an air gap.

For example, during the operation of the touchscreen, a voltage gradient is applied to the first sheets.

* When the two sheets are pressed together, the second sheet measures the voltage as distance along the first sheet, so we know the X coordinate. The voltage gradient is applied to the second sheet to confirm the Y coordinate.
* Then we use the **ADC** to convert the voltage on the X coordinate and Y coordinate into digit numbers.

How to integrate touch panel driver in Linux?

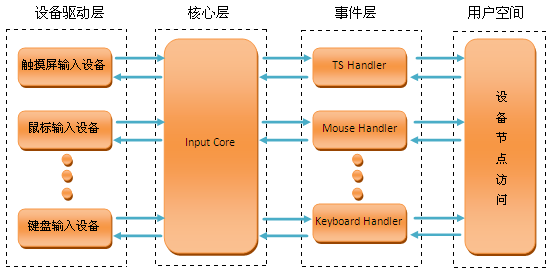
* We integrate the touch panel driver by using **the input system** in Linux:
  + Specify the device ID
  + Do **ioremap** for the registers of the touch panel controller and ADC controller
  + Configure the touch panel controller and ADC controller for the initialization
  + using **request\_irq ()** with input parameters on the **interrupt number**, **interrupt handler** to register IRQ.
  + Allocate input device data structure (call input\_allocate\_device ())
  + Fill out the members of the input device data structure:
    1. Specify the support event types (synchronous event, press key event, absolute coordinates event and so on)
    2. Specify the range on the absolute coordinates.
    3. Register the device to input subsystem.
    4. Call module\_init () to register the module.
* We need to implement the interrupt handler routine to do ADC conversion for the coordinates when we detect the user presses the touch panel.
* Report the event (call input\_report\_key ()/intput\_report\_abs () …), so that user layer can handle the event.

## Description on the input subsystem in Linux? (done)

Input subsystem includes device driver layer, input core layer and even handler layer.

The device driver layer provides the read/write access to related hardware registers; it also handles hardware interrupt and convert it to the input event; core layer will pass this input event the event handler layer.

Use layer can call the related API to get the input event from the hardware device.





Event types

* EV\_SYN:

Used as markers to separate events. Events may be separated in time or in space, such as with the multitouch protocol.

* EV\_KEY:

Used to describe state changes of keyboards, buttons, or other key-like devices.

* EV\_REL:

Used to describe relative axis value changes, e.g. moving the mouse 5 units to the left.

* EV\_ABS:

Used to describe absolute axis value changes, e.g. describing the coordinates of a touch on a touchscreen.

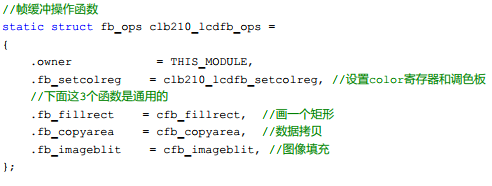
* EV\_MSC: EV\_SW, EV\_LED, EV\_SND, EV\_REP, EV\_FF, EV\_PWR, EV\_FF\_STATUS

## LCD driver (done)

We integrate LCD driver by using **frame buffer** in the Linux.

* Every picture is called frame. It consists of multiple lines. Every line consists of multiple pixels.
* Pixels will be shown to the LCD from left up to right end by line. It will jump to the left of next line after it finished the current line. We use HSYNC to define Very left, and VSYNC to define Very UP.
* allocate a new frame buffer data structure;
* fill out the fields including device id, the color standard such as using RGB, screen size, the resolution;
* fill out operation functions.

For the operation function, refer to below diagram:



* Use ioremap to map the registers on the LCD controller/ LCD GPIO.
* Configure the registers of LCD controller to initialize the LCD.
* Register frame buffer (call the **register\_framebuffer (**struct fb\_info \*fb\_info**)**)

Display the data to the LCD:

* In the user space, we can write data into the frame buffer directly by using mmap () to display the data to the LCD.

Fetch data from the LCD:

* In the user space, we can read data from the frame buffer directly by using mmap ()

Reference information (key concepts):



## Camera - Video stream (V4l2 under Linux) done

How to integrate a camera driver under V4L2 subsystem in Linux:

* Allocate video\_device:
  + Fill out the fields such as image size/resolution/pixel size (HD 1280 x 720 pixels).
  + Fill out the operation functions (I think the manufacturer may provide the functions, please add any detail we should do for camera initialization)

摄像头驱动源代码路径：/drivers/media/video/uvc/uvc\_v4l2.c

make menuconfig

Device Drivers --->

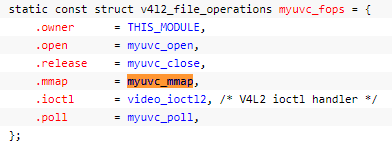
<\*> Multimedia support --->

<\*> Video For Linux

[\*] Video capture adapters --->

[\*] V4L USB devices --->

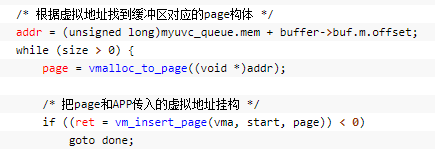
<\*> USB Video Class (**UVC**)





static int myuvc\_mmap(struct file \*file, struct vm\_area\_struct \*vma)

{



}

* Do the ioremap on the related hardware registers, so driver can use it
* Register tie video\_device (call **video\_register\_device** ())

How to do the video stream in the **application** layer?

* Open the video device, (“/dev/video0”, major device id: 81)
* Query the capacity on the video device.
* Configure the video device output format (**PAL:25 frames**, NTSC); configure the image size/resolution/frame rate.
* Request frame buffer which is usually greater than 3.
* Do the mmap to map the frame buffer from kernel space to user space.
* Put the requested framer buffer to the output queue for the video streaming.
* Call ioctl () to start video stream
* User program fetches the video frames from the output queue
* Repeat the video streaming process.
* ⑴打开摄像头设备文件 open// 用非阻塞模式打开摄像头设备

⑵获取本摄像头支持的图像数据编码信息  VIDIOC\_QUERYCAP

⑶获取所支持捕捉格式          VIDIOC\_ENUM\_FMT

⑷设置视频格式 VIDIOC\_S\_FMT

⑸获取视频格式 VIDIOC\_G\_FMT

⑹请求分配缓存 VIDIOC\_REQBUFS

⑹获取内存空间及其假如内存队列 VIDIOC\_VIDIOC\_QUERYBUF   VIDIOC\_QBUF

⑺开始传输视频流数据  VIDIOC\_STREAMON   VIDIOC\_QBUF

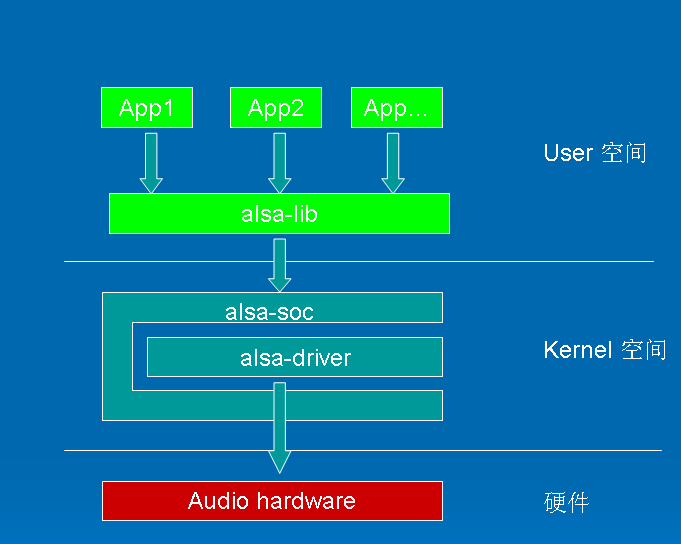
  enum v4l2\_buf\_type type =V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ret = ioctl(fd\_video, VIDIOC\_STREAMON,&type); //图像信息不断的存放在内存队列中

⑻对数据流数据的处理VIDIOC\_DQBUF

(9) Stop the video stream to exit.

## ALSA (done)





Alsa driver

* Create a snd\_card instance by using snd\_card\_create(index, id, THIS\_MODULE, 0, &card) //struct snd\_card \*card
* Assign the resource, such as Interrupt, IO, DMA
* Register the private data to a lowlevel device by using snd\_device\_new();
* Set Driver ID and name
* Create sound card function device, such as PCM, Mixer, MIDI
* Register the sound card by using snd\_card\_register

Alsa application

* 1. We need create a PCM handle, and set the PCM stream is playback or capture, also some parameters, such as buffer size, sample rate, pcm data format.
* We need allocate a hardware config structure on the stack: snd\_pcm\_hw\_params\_alloca()
* 2. Open PCM device for playback.

rc = snd\_pcm\_open(&handle,   "default",  SND\_PCM\_STREAM\_PLAYBACK, 0);

The default read/write is block mode, that means write function will be blocked until write successful.

We also can set it to non-block mode. It will return -EAGAIN, when buffer is full.

* Then we need set the device configurations, such as sample rate, channel number and period size. snd\_pcm\_hw\_params\_any(pcm\_handle, hwparams)
* Read PCM device

snd\_pcm\_readn(pcm\_capture\_handle, capdata, num\_frames);

* Write PCM device

**int** rc = snd\_pcm\_writei(handle, buf, n /4 );

how to mix

* fd = open("/dev/dsp", O\_RDWR);
* fd = open("/dev/mixer", O\_RDONLY);
* Another neat feature in FFADO is that some the DSP mixing features of the hardware have been integrated into the driver, complete with a graphical mixer for controlling the balance of the various inputs and outputs. This is different to the ALSA mixer, because it means audio streams can be controlled on the hardware with zero latency, which is exactly what you need if you're recording a live performance.

## UART: (done)

2 wires: 1 send, 1 receive.

Full duplex

asynchronous

## SPI (done)

3/4 wires: 1 send, 1 receive, 1 CLK, CS

## 485 (done)

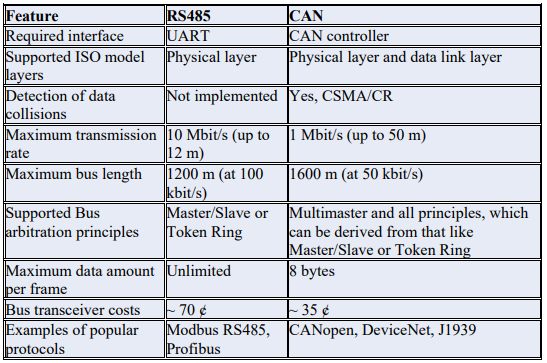
## CAN (done)

A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but is also used in many other contexts.

2 wires, 1 wire, using 1 wire to transport signal and power.

Reliable, EMC/EMI

Differential transmission



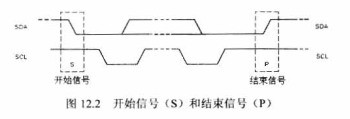
## I2C (done)

Describe I2C device driver

* I2C is a two lines BUS, it has two wires, SCL is the clock line; SDA is the data line.

How does it work

* The devices on the I2C bus are either master or slave, and it is addressable via software.
* When the master wants to talk to a slave, it begins by issuing a start sequence on the I2C bus.
* The start sequence is implemented by setting Clock line as **high** and the Data line changes **from high to low**. Data is transferred in sequence of 8 bits. The stop sequence is implemented by setting clock line as high and the data line changes from low to high.



## SCCB (done)

2 lines: SIO\_C, SIO\_D; for just 1 slave device

3 lines: SIO\_E（cs 低有效）、SIO\_C与SIO\_D for multiple save device

Transmission: 3 phases sending

* 1. OV7649的ID Address
* 2. 写数据的目的目的寄存器地址sub\_address
* 3. 要写入的数据 Write Data
* 4. 如果要连续写数据，因为ov7649会自动把寄存器地址加1，不需要再次输入ID地址（第一步），从而三相变成二相。

Difference between I2C and SCCB

* I2C采用的是7位地址数据，首字节最低位(最后一位)代表读写位，第二字节SUB寄存器高位为1表示自动加subaddress。

比如0v9650，地址是0x60：

挂在SCCB采用的是8位地址数据， 0x60, B0110 000**0 这个**0还是地址位

挂在I2C上，有的直接发地址0x60就可以。但是如果有专门的处理器维护地址，则写入0x30，模块会自动组合成0x60发送：

0x30, B0110 000**0 这个**0是读写位

* SCCB是简化的I2C协议，SIO-l是串行时钟输入线，SIO-O是串行双向数据线，分别相当于I2C协议的SCL和SDA。
* SCCB的总线时序与I2C基本相同，它的响应信号ACK被称为一个传输单元的第9位，分为Don’t care和NA。Don’t care位由从机产生, NA位由主机产生，由于SCCB不支持多字节的读写，NA位必须为高电平。另外，SCCB没有重复起始的概念，因此在SCCB的读周期中，当主机发送完片内寄存器地址后，必须发送总线停止条件。不然在发送读命令时，从机将不能产生Don’t care响应信号。

## DMA

Purpose

DMA stands for direct memory access and refers to the ability of devices or other entities in a computing system to modify main memory contents without going through the CPU.

How to write a DMA driver

* + 1. DMA channels apply
    2. DMA interrupt Apply
    3. Control register setting
       - Destination - where to go
       - Source – from
       - length
    4. Into DMA waiting que
    5. Clear DMA interrupt
    6. Release DMA channel

一、DMA控制器硬件结构

* DMA允许外围设备和主内存之间直接传输 I/O 数据， DMA 依赖于系统。每一种体系结构DMA传输不同，编程接口也不同。
* 数据传输可以以两种方式触发：一种**软件请求数据**，另一种**由硬件异步传输**。

a -- 软件请求数据

     调用的步骤可以概括如下（以read为例）：

* （1）在进程调用 read 时，驱动程序的方法分配一个 DMA 缓冲区，随后指示硬件传送它的数据。进程进入睡眠。
* （2）硬件将数据写入 DMA 缓冲区并在完成时产生一个中断。
* （3）中断处理程序获得输入数据，应答中断，最后唤醒进程，该进程现在可以读取数据了。

b -- 由硬件异步传输

      在 DMA 被异步使用时发生的。以数据采集设备为例：

* （1）硬件发出中断来通知新的数据已经到达。
* （2）中断处理程序分配一个DMA缓冲区。
* （3）外围设备将数据写入缓冲区，然后在完成时发出另一个中断。
* （4）处理程序利用DMA分发新的数据，唤醒任何相关进程。

网卡传输也是如此，网卡有一个**循环缓冲区（通常叫做 DMA 环形缓冲区）**建立在与处理器共享的内存中。每一个输入数据包被放置在环形缓冲区中下一个可用缓冲区，并且发出中断。然后驱动程序将网络数据包传给内核的其它部分处理，并在环形缓冲区中放置一个新的 DMA 缓冲区。

* 驱动程序在初始化时分配DMA缓冲区，并使用它们直到停止运行。

============================================

linux 驱动 dma 为什么 要物理 地址

在Linux驱动中，DMA（直接内存访问）通常使用物理地址而不是虚拟地址。这是因为在DMA操作期间，内存可能会被操作系统重新映射到不同的虚拟地址，这可能会导致DMA写入不正确的位置。通过使用物理地址，驱动程序可以确保DMA访问的内存位置是正确的，并且不会因为虚拟地址重映射而出现问题。

另外，使用物理地址还可以避免缓存等硬件加速技术引起的问题。如果使用虚拟地址，DMA可能会尝试访问被缓存的内存，这可能导致DMA读取过时的数据或写入无效的数据。

因此，为了确保DMA的正确性和可靠性，Linux驱动程序通常使用物理地址而不是虚拟地址。

Linux驱动程序使用DMA（直接内存访问）技术将数据传输到设备或从设备传输数据。在使用DMA时，设备可以直接访问系统内存而无需CPU的干预。因此，为了让设备能够直接访问内存，Linux驱动程序必须告诉设备内存的物理地址。

在Linux内核中，虚拟地址映射到物理地址，这个映射是由内存管理单元（MMU）实现的。然而，在使用DMA时，设备并不使用MMU，它只能使用物理地址。因此，Linux驱动程序需要通过物理地址将虚拟地址转换为设备可以访问的地址，以确保设备可以正确地访问内存中的数据。

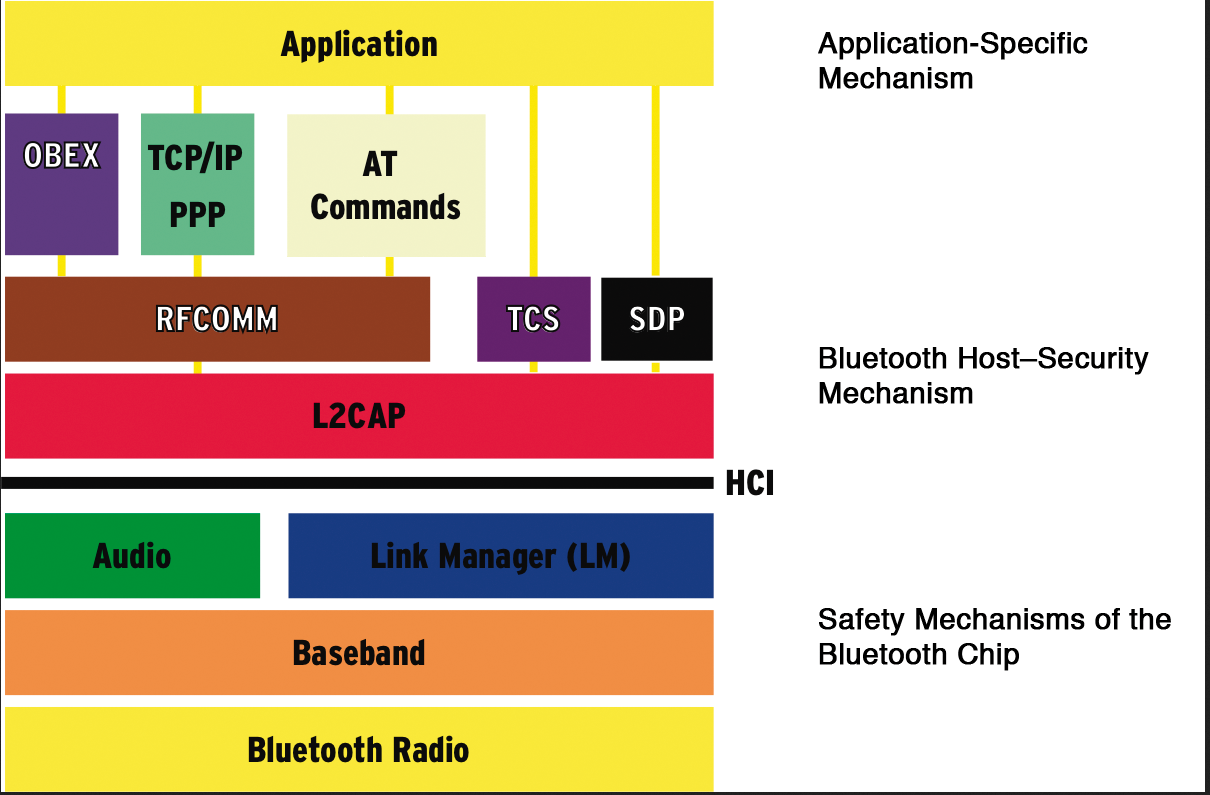
此外，物理地址可以保证DMA的可靠性。如果使用虚拟地址，那么当内存被重新映射或交换时，设备可能无法访问正确的数据。而物理地址是固定的，不受内存管理的影响，因此可以确保DMA传输的可靠性。

因此，在使用DMA时，Linux驱动程序需要使用物理地址来确保设备可以正确地访问内存，并保证DMA传输的可靠性。

## Bluetooth

L2CAP

The Logical Link Control and Adaptation Layer Protocol is layered over the Baseband Protocol and resides in the data link layer.



* SDP: service discovery protocol.
* TCS: telephone communication service.
* RFCOMM: it’s a simple transport protocol, which provide emulation of RS232 serial ports over the L2CAP protocol.
* HCI: it provides a command interface to the baseband controller and link manager, and access to hardware status and control registers.
* **OBEX:** (abbreviation of OBject EXchange, also termed IrOBEX) is a communications **protocol** that facilitates the exchange of binary objects between devices.

Test Bluetooth Commands:

* lsusb
* Hciconfig
* Hcitool dev
* hciconfig hci0 up
* Hcitool scan
* Modify configure files:

/etc/Bluetooth/rfcomm.conf, and change the device to the result which hcitool scan.

* Create Bluetooth device:

rfcomm\_create\_dev

## PCI/PCIe (done)

Hardware

* Passive Optical Network

It’s a telecommunications technology used to provide fiber to the end consumer.

* Product

HBA (GPON storage) and Optical ethernet (GPON network device)

* The difference between PCI and PCIe?

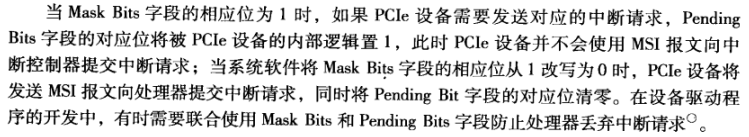
|  |  |
| --- | --- |
| PCI | PCIe |
| Use parallel bus; 0~31, multiplexing the data/address lines | Use serial bus;  have tx and rx, differential circuit  使用数据包(Packet)进行数据传输 |
| The speed is about 133/266MB/s | The speed is higher, between 1.25MB/s~4MB/s |
| 4 interrupt pins | MSI capability,  32bit and 64bit **Message**  32bit and 64bit interrupt Masking |
|  | 传统复位方式和FLR复位 |

PCIe:

的配置空间有0x00~0x3F, 另外还扩展了0x40~0xFF这段配置空间。在这段空间内主要存放一些与MSI/中断机制和电源管理相关的capability结构.

PCIe设备还支持 0x100~0xFFF这段扩展配置空间，最大可以为4KB

MSI:



* The PCI framework?
  + We can have up to 256 devices connected to the same PCI bus.
  + PCI bus is shared bus. We have three types of device under PCI bus: **PCI master device**, **PCI slave device**, **bridge device**. The PCI slave device only can passively accept the read/write request from **HOST bridge** or other PCI master device.

During the data transfer, the initiating device (PCI master device) must get permission to control the PCI bus. This is done during the process of bus arbitration. then it can send read/write request to main memory of processor or PCI slave device.

* + PCI device has its own independent address space, the PCI bus address space.

The processor can not access PCI device directly, it needs to go through HOST bridge to access PCI device.

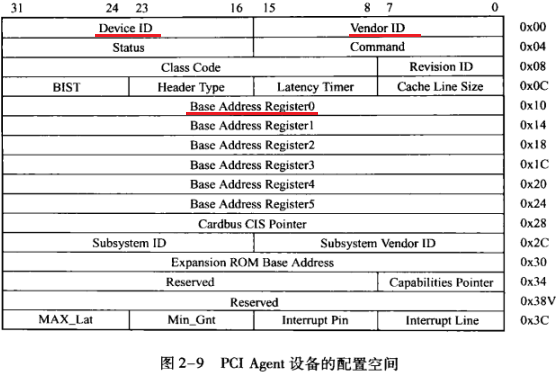
PCI device also can not access the main memory at the processor side. The PCI device needs to go through HOST bridge to access the main memory.

PCI system supports “hot plugin”.

Every PCI device has its own configuration space. It includes the vendor id, device id, and bar register information.

Interrupt mechanism: The PCI bus has 4 interrupt lines, all of them are available to each device.

Every PCI device has it own configuration space listed as below:



Every PCI device has its unique Device ID, unique Vendor ID;

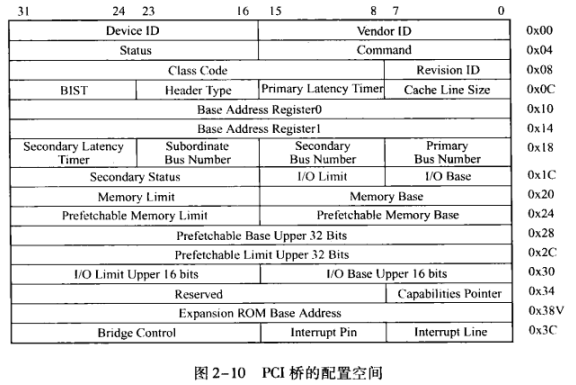
In the “Header type “register:

Bit6-bit0: 0 represents for PCI device; 1 represents for PCI bridge.

Base address register 0-5: Bar register has the information on the base address which this PCI device can use in the PCI address space

There are only 15 interrupt request lines available for hardware.

Every PCI bridge has its own configuration space listed as below:



The configuration space of PCI bridge has information like the memory limit/memory base:

We can have multiple PCI devices connected to the same bridge. The PCI devices may use memory space at the processor side.

The memory base register: Include the information on the base address on the memory space at the processor side which the PCI devices can use.

The memory limit register: Include the information on the total size on the memory space at the processor side which the PCI devices can use.

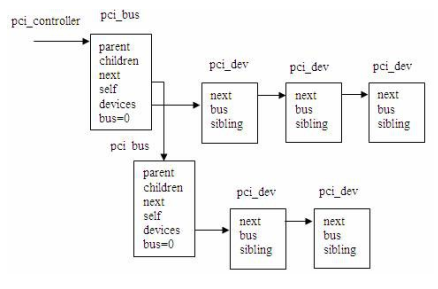
* What device using PCI bus which you worked on, CPU, device, speed?
  + Network device.
* The schematic?
* PCI enumeration?

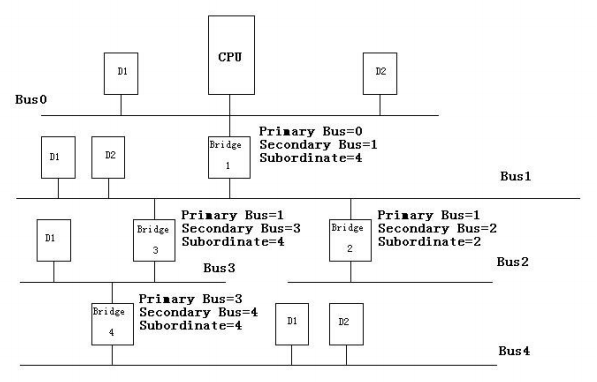
The system does the PCI bus scan from bus 0 to bus 255.

For each bus, the system scans the connected PCI devices by reading the configuration space.

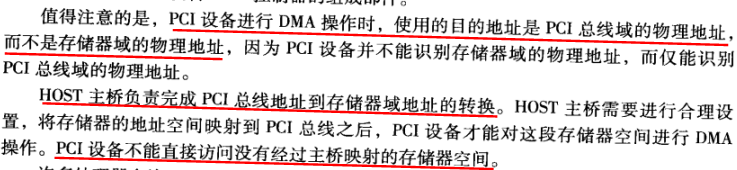
It reads out the device ID register and vendor ID register, if the two registers have valid value (which is not 0xFFFF), then it means the current device is a valid PCI device/PCI bridge. The system will check the Header **Type** register, if this register is **1**, then it means it’s **a PCI bridge**, or it’s a PCI device.

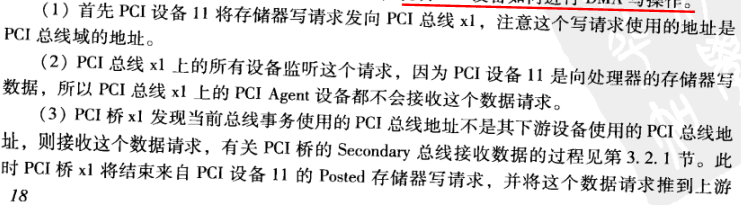
Eventually the system will be able to generate the PCI bus/device tree with numbering all PCI buses.

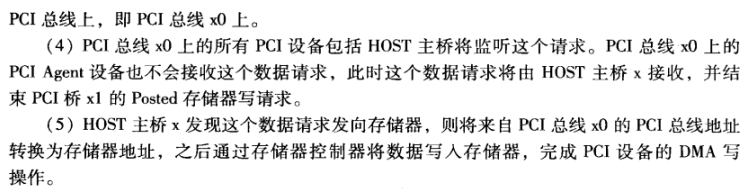




* How does PCI device work?
  + For sending out the data packet:
    1. We need to fill out hard\_start\_xmit () function.
    2. The CPU uses DMA for the data transmission.
    3. The IP layer puts the data packets in the **sk\_buff**, we need to move the data from **sk\_buff** to the **DMA** memory to start the data transmission.
    4. CPU将数据放在内存中，然后DMA模块负责将数据搬移到网卡的txfifo中。
    5. 由于DMA模块只能访问CPU的内存空间，所以DMA模块需要通过host bridge来搬移数据。Host bridge会将CPU域的地址转换成PCI域的地址，将数据传送到PCI侧的网卡的txfifo中。
    6. Txfifo的register的地址是bar register+offset.
  + For receiving data packets:
    1. After the DMA receives new packets to memory, it will trigger the interrupt.
    2. we implement the interrupt handler routine to receive the data packets; and put the data packets in skb\_buff;
    3. And then call netif\_rx () to pass the data packets to the upper layer.
    4. Up layer uses dev\_queue\_xmit () to send out the data packets
  + Receive







* + Sending

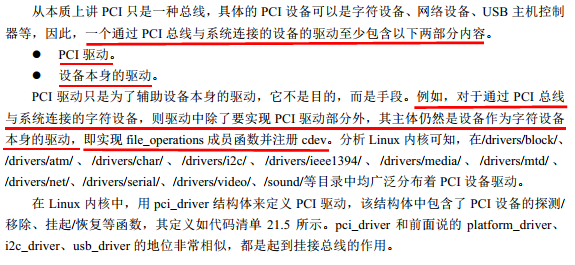
CPU uses DMA to send data packets. Since CPU can not access the PCI address space, so the host bridge will translate the CPU memory space address to PCI space address and then forward the data packets to the corresponding PCI device which is the network card (the txFIFO) for sending out the packets.

CPU将数据放在内存中，然后DMA模块负责将数据搬移到网卡的txfifo中。

由于DMA模块只能访问CPU的内存空间，所以DMA模块需要通过host bridge来搬移数据。Host bridge会将CPU域的地址转换成PCI域的地址，将数据传送到PCI侧的网卡的txfifo中。

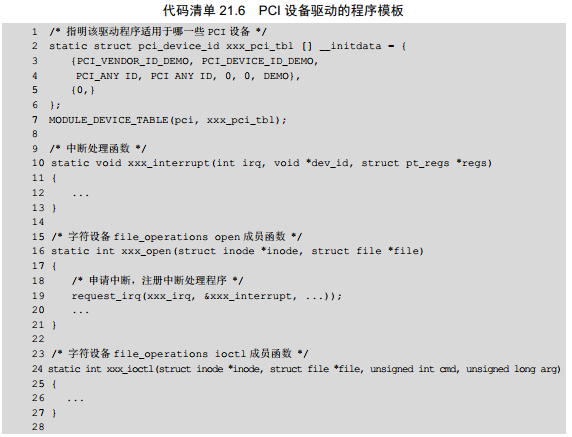
Txfifo的register的地址是**bar** register+offset.

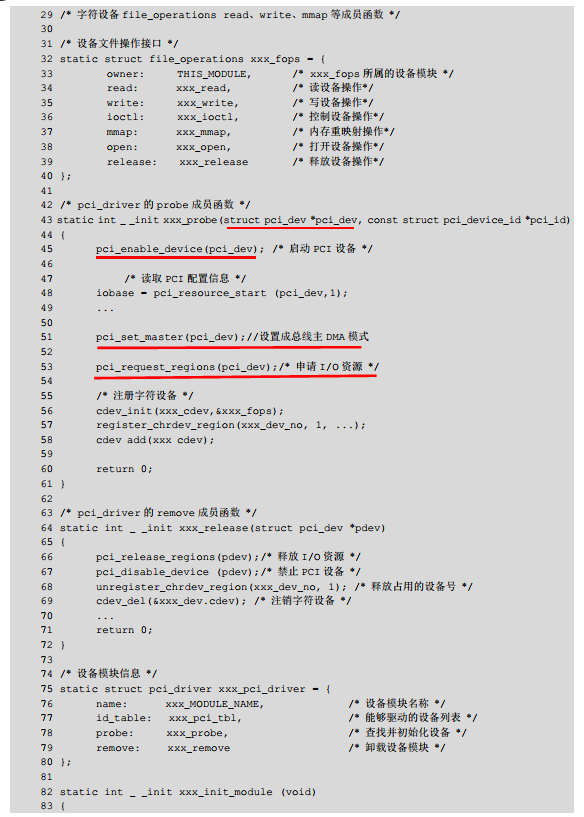
Software in Linux

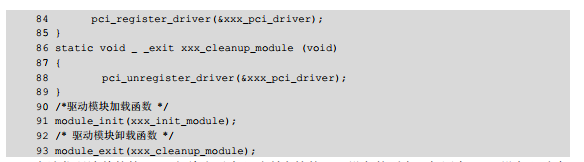
* What drive type is the PCI device?
* 
* How to port the PCI device? PCI subsystem? Which part code do you need to modify?











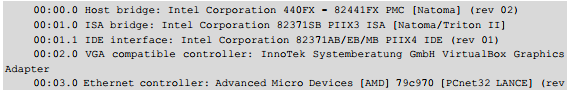
After we get the iobase, we need to use ioremap() to convert it from the physical address to virtual address so linux kernel can access it.

Please note, the iobase is the address in processor’s memory space.

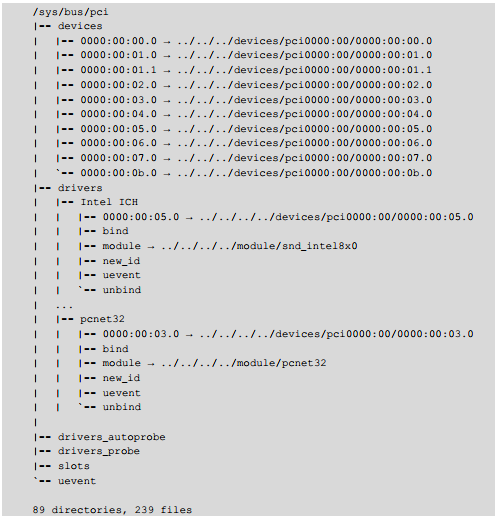
For more details, please refer to “第21章 PCI设备驱动“

* How to test PCI device? Commands?

lspci



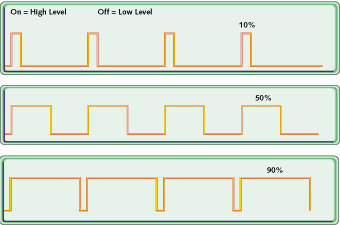
sysfs 文件系统/sys/bus/pci 目录中也给出了系统中总线上挂接的设备及驱动信息，该目录下的  
树型结构如下：



## PWM(Pulse Width Modulation)

PWM is a way of digitally encoding analog signal levels. Using high-resolution counters, the duty cycle of a square wave is modulated to encode a specific analog signal level.

For example, A PWM output at a 10% duty cycle. That is, the signal is on for 10% of the period and off the other 90%. the supply is 10Volt and the duty cycle is 10%, a 1Volt analog signal results.



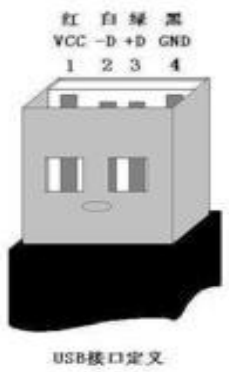
The PWM signal is still digital because, at any given instant of time, the full DC supply is either fully on or fully off.

Given a sufficient bandwidth, any analog value can be encoded with PWM.

## USB (done)

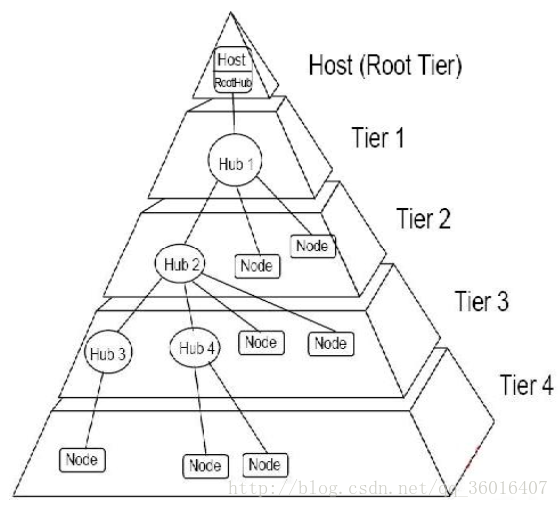
Hardware:

4 wires: vcc(+5v), -D(DATA-), +D(DATA+), GND



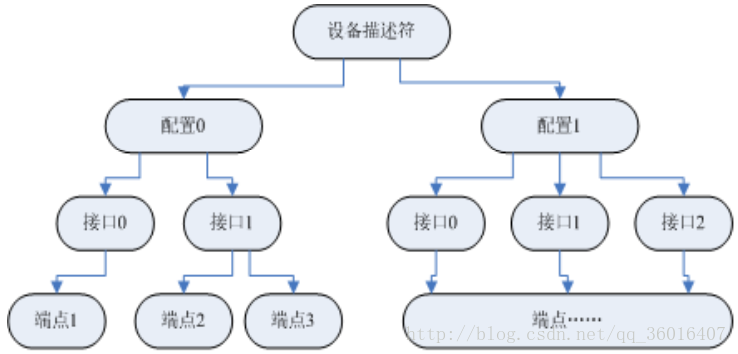
**系统拓扑结构：**

对于每个USB系统来说，都有一个称为主机控制器的设备，该控制器和一个 根 Hub 作为一个整体。这个根Hub下可以接多级的Hub，每个子Hub又可以接子Hub。每个USB设备作为一个节点接在不同级别的Hub上。 每条USB总线上最多可以接127个设备。



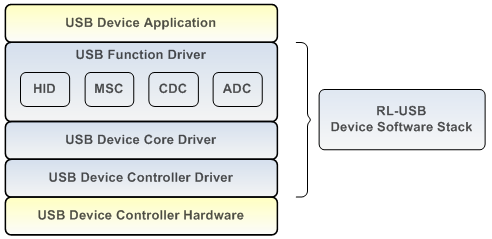
**常见的USB主控制器规格：**   
**OHCI：**主要是非PC系统上的USB芯片   
**UHCI：**大多是 Intel 和 Via 主板上的 USB 控制器芯片。他们都是由USB1.1规格的。   
**EHCI：**由 Intel 等几个厂商研发，兼容 OHCI 和 UHCI ，遵循USB2.0规范。

**USB设备逻辑结构：**   
在USB设备的逻辑组织中，包含 **设备**、**配置**、**接口** 和 **端点** 4个层次。设备通常有一个或多个配置, 配置通常有一个或多个接口, 接口有零或多个端点。



* 每个USB设备都可以包含一个或多个配置，不同的配置使设备表现出不同的功能组合，配置由多个接口组成。在USB协议中，接口代表一个基本的功能，一个功能复杂的USB设备可以具有多个接口, 而接口是端点的汇集。
* 举例说明：   
  一个USB播放器带有音频，视频功能，还有旋钮和按钮。   
  配置1：音频(接口)+旋钮(接口)   
  配置2：视频(接口)+旋钮(接口)   
  配置3：音频(接口)+视频(接口)+按钮(接口)   
  音频接口，视频接口，按钮接口，旋钮接口均需要一个驱动程序。
* **USB设备中的唯一可寻址的部分是设备端点**，端点的作用类似于寄存器。每个端点在设备内部有唯一的端点号，这个端点号是在设备设计时给定的。主机和设备的通信最终都作用于设备上的各个端点。每个端点所支持的操作都是单向的，要么只读，要么只写。
* **USB描述符：**   
  **在每一个USB设备内部，包含了固定格式的数据**，通过这些数据，USB主机就可以获取USB设备的类型、生产厂商等信息。这些固定格式的数据，我们就称之为**USB描述符**。
* 标准的USB设备有 5种 USB描述符： 设备描述符，配置描述符，接口描述符，端点描述符，字符串描述符。

Linux-USB sub system

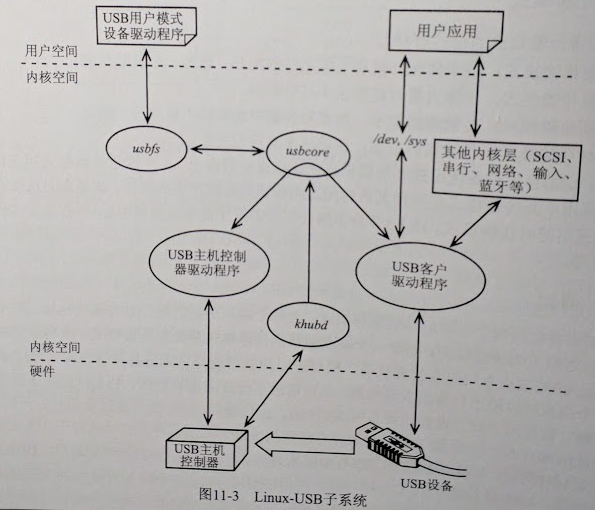


HID: Human Interface Device class

MSC: Mass Storage Device class

CDC: Communication Device class

ADC: Audio Device class



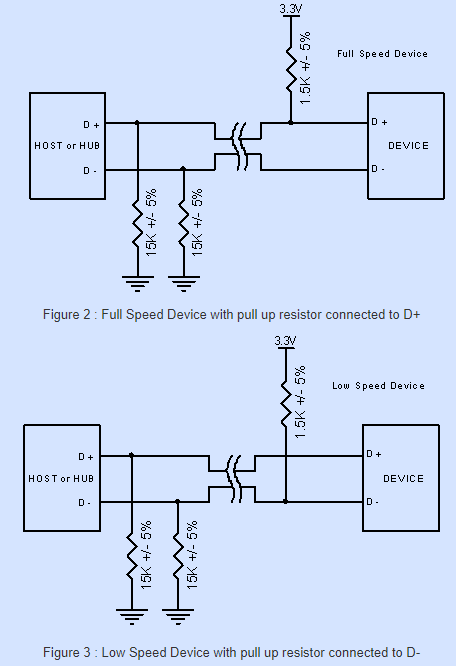
USB supports 4 transfer types:

* Control transfer
* Interrupt transfer
* Isochronous transfer [aɪ'sɑkrənəs] equal in time
* Bulk transfer

6 states

* Attached
* Powered
* Default
* Address
* Configured
* suspend

High-speed or Low-speed



Enumeration: host controller gets device information and set up the device.

Khubd daemon thread will be wake up and run **hub\_events** function, when usb port add new device. This function will do the enumeration.

**Detect** -> **reset** -> **set\_address** -> **device\_add**

* 1. 用户把USB设备插入USB端口或者给系统启动时设备上电。HUB会监测它各个端口数据线上（D+/D-）的电压。并根据D+还是D-被拉高来判断该设备是全速还是低速。

When the usb device plugs in, hub will detect the data line voltage is high. Hub can tell is a high-speed device or low speed device by D+ and D- which one was pull up to HIGH. Then hub will power up the device.

* 2. Host了解连接的设备

Hub will report the endian statues to host. Host will send **Get\_Port\_Status** request to hub if there is a connect/disconnect event happened.

* 3. Hub检测所插入的设备是高速还是低速设备

Hub will reply the device speed type(High/Low) to host after host sent the Get\_Port\_Status.

* 4. Hub复位设备

If host know the new device got connected, it will wait 100ms until the device power up completely. Then Host controller will send a **Set\_Port\_Feature** request to hub. Hub will reset the end-point (the new device) for at least 10ms.

* 5. Host监测所连接的全速设备是否是支持高速模式

Host will test the new device whether is a high-speed device or not.

* 6. Hub建立设备和主机之间的信息通道

Host will send Get\_Port\_Status request to check the reset status. Then the new device will be Default state. Then new device can communicate with Host controller. The new device address is 0, endian port is 0.

* 7. 主机发送Get\_Descriptor请求获取默认管道的最大包长度。

Enumeration

* 8. 主机给设备分配一个地址

Host allocate an address for the new device by sending **set\_address**. Then new device enters address state.

* 9. 主机获取设备的信息

Host requires device descriptor by sending **Get\_Descriptor** to new device.

* 10. 主机给设备挂载驱动

Host will add the new device to USB bus device list by using **device\_add**.

USB is a master-slave protocol where a host controller communicates with client devices. The SoC in the figure has built-in USB controller silicon that supports four buses and three modes of operation:

* Bus 1 runs in host mode and is wired to an A-type receptacle via a USB transceiver. You can connect a USB pen drive or a keyboard to this port.
* Bus 2 also functions in host mode, but the associated transceiver is connected to an internal USB device rather than to a receptacle. Examples of internal USB devices are biometric scanners, cryptographic engines, printers, *Disk-On-Chips* (DOCs), touch controllers, and telemetry cards.
* Bus 3 runs in device mode and is wired to a B-type receptacle through a transceiver. The B-type receptacle connects to a host computer via a B-to-A cable. In this mode, the embedded device functions as, for example, a USB pen drive, and exports a storage partition to the outside world. Embedded devices such as MP3 players and cell phones are more likely than PC systems to be at the device side of USB, so many embedded SoCs support a USB device controller in addition to a host controller.
* Bus 4 is driven by an *On-The-Go* (OTG) controller. You can use this port, for example, to either connect a pen drive to your system or to turn your system into a pen drive and connect it to a host. Unlike buses 1 to 3, bus 4 uses an intelligent transceiver that exchanges control information with the processor over I2C. The transceiver is wired to a Mini-AB OTG receptacle. If two embedded devices support OTG, they can directly communicate without the intervention of a host computer.

USB's similarities with I2C:

* USB and I2C are master-slave protocols.
* Device addresses reside in a private 7-bit space.
* Device-resident memory is not mapped to the CPU's memory or I/O space, so it does not consume CPU resources.

USB's similarities with PCI:

* Devices can be hotplugged.
* Device driver architecture resembles PCI drivers. Both classes of drivers own probe()/disconnect()[[1]](http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch11lev1sec1.html#ch11fn01) methods and ID tables identifying the devices they support.
* Supports high speeds. Slower than PCI, however. See [Table 10.1](http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch10lev1sec1.html#ch10tab01) in [Chapter 10](http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch10.html#ch10), "Peripheral Component Interconnect," for the speeds supported by different members of the PCI family.
* USB host controllers, like PCI controllers, usually have built-in DMA engines that can master the bus.
* Supports multifunction devices. USB supports interface descriptors per function. Each PCI device function has its own device ID and configuration space.

## how does Jtag work (done)

The official JTAG standard requires 4 standard pins: Test Data In, Test Data Out, Test Clock Test Mode, and defines an optional 5th.

By using **Boundary-Scan** technology, we can capture the output signal from **Boundary-Scan Chain**.

TAP (**Test Access Port**) controls those Boundary-Scan Chain. By adding a shift register unit to the close by input/output pin.

## Objdump (done)

Objdump -D vmlinux > tt symbol table

## Secure boot (done)

Secure boot provides a foundation for the security architecture of the device. Technically, secure boot is defined as a boot sequence in which each software image that is loaded and executed on a device is authorized using software previously authorized by this system. This sequence is designed to prevent unauthorized or modified code from being run by ensuring that all code is checked before it is executed.

# interrupt

## (done)Why should we avoid the context switching during interrupt?

Some interrupt cannot be switched, such as clock, DMA

It may cause the interrupt cannot be waked up.

For the ISP nest, the interrupt has high task priority, it only can be waked up by higher priority task. So, if it is falling into sleep, it cannot be waked up by process, it must be waked up by higher priority interrupt, which is risky.

~~The sleeping function calls schedule(), the context(CPU registers, process status and stack) will be saved. But during the interrupt sleep, the context may change. So, the saved contexts are obsolete. It will cause OS issue.~~

## (done)Request\_irq

request\_irq (unsigned int **irq**, irq\_handler\_t **handler**, unsigned long **flags**, const char **\*name**, void **\*dev**)

* **irq**: hardware interrupt number
* **handler:** interruptfunction
* **第三个参数flags:**为**irq trigger type**标志位。可以取IRQF\_DISABLED、IRQF\_SHARED和IRQF\_SAMPLE\_RANDOM之一。在本实例程序中取 **IRQF\_SHARED，该标志表示多个中断处理程序共享irq中断线**。一般某个中断线上的中断服务程序在执行时会屏蔽请求该线的其他中断，**如果取** **IRQF\_DISABLED标志，则在执行该中断服务程序时会屏蔽所有其他的中断**。**取IRQF\_SAMPLE\_RANDOM则表示设备可以被看作是事件随见的发生源。**
* **name**: interrupt name. it can be set to device driver name. we can see it by “cat /proc/interrupts”
* **dev**:set to device structure or NULL
* **return**: **0 (success). -INVAL (invalid or NULL). -EBUSY (interrupt has been occupied)**

## (done)Explain software interrupt?

The SWI (software interrupt instruction) causes a software interrupt exception. We can use SWI to do the system call.

Each SWI instruction has an associated SWI number, which is used for a specific function call.

## (done)Top and Bottom Halves

The reason why we need top and bottom halves is because we may need to do a lot of work in response to an interrupt, but at the same time we also need to finish the interrupt handlers as soon as possible so that the other interrupts will not be blocked too long.

Top half is the routine we register with request\_irq; top half is used to respond the interrupt.

Bottom half is the routine which is scheduled by top half and it will be executed later at a safe time.

Mechanism for bottom-half:

* Takslet: atomic, very fast
* Workqueue: more delay, allow to sleep.

## (done)To deploy a module inside kernel, what are the possible methods?

First method: Use modprobe or insmod to insert the new module into the kernel dynamically.

Second method: Or link the new module statically into the kernel; we can use “make menuconfig” to select devices to compile the new module into kernel and then compile the kernel again.

## In how many ways we can assign a major minor number to any device?

We can choose device numbers, or Linux system allocate device numbers.

* register\_chrdev\_region (dev\_t, unsigned int count, char \*name)
* alloc\_chrdev\_region (dev\_t \*dev, unsigned int firstminor, unsigned int count, char \*name)
* unregister\_chrdev\_region (dev\_t first, unsigned int count)

## (done) When is that we want to use "user virtual address" instead of "kernel virtual address"? List some situations when we cannot go with kernel virtual address.

When we run a program in user space, we need to use user virtual address.

And if we’re running code in kernel mode, we use kernel virtual address. But if the kernel code must interact with some userspace component, then we need to translate the “kernel virtual address” to “user virtual address” before passing them and vice-versa.

## (done) how does kernel space communicate with user space?

Systemcall

* Copy\_to\_user, copy\_from\_user: Will cause block, cannot used in swi and interrupt.
* Mmap
* Open, close, read, write

Procfs

Netlink socket



* User:
  + Socket ()
  + Bind
  + Sendmsg ()
  + Recvmsg ()
* kernel
  + Socket (AF\_NETLINK, SOCK\_RAW, netlink\_type)

# synchronization

## What are the best synchronization techniques used in linux kernel?

|  |  |  |
| --- | --- | --- |
|  | Characteristic |  |
| Mutex | **Only for thread**  can cause sleep.  The call to mutex/semaphore my block the current running thread. So, don’t use it in ISR. | 1. Only one task can hold the mutex at a time.  2. Only the owner can unlock the mutex.  3. multiple unlocks are not permitted.  4. recursive locking is not permitted |
| Semaphore | Can for thread and process  Can cause sleep.  The call to mutex/semaphore my block the current running thread. So, don’t use it in ISR. | More suitable for some synchronization problems like producer-consumer.  1. task may not exit with mutex held.  2. memory areas where held locks reside must not be free.  3. mutexes may not be used in hardware or software interrupt contexts such as tasklets and timers. |
| Spinlock  (writelock/read lock) | Busy waiting;  Hold the locker for short time,  no sleep;  can be used in interrupt context. | 1. for preemption kernel  2. for nonpreemption kernel  3. for multiple kernel |
| Atomic | For simple counter variable/for bitwise |  |

They are all kernel resources that provide synchronization services.

**Mutex:**

Mutexes are typically used to serialize access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section.

Lock more than once will call dead lock. Because on other thread can unlock the mutex.

**Semaphore:**

A semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore) and can signal that they have finished using the resource (incrementing the semaphore).

虽然 Mutex和Semaphore 在一定程度上可以互相替代，比如你可以把 值最大为1 的Semaphore当Mutex用，也可以用Mutex＋计数器当Semaphore。

但是对于设计理念上还是有不同的：

* Mutex管理的是资源的使用权，mutex used to protect shared resources.
* 而Semaphore管理的是资源的数量，有那么一点微妙的小区别。

打个比方，在早餐餐厅，大家要喝咖啡。

* 如果用Mutex的方式，同时只有一个人可以使用咖啡机，他获得了咖啡机的使用权后，开始做咖啡，其他人只能在旁边等着，直到他做好咖啡后，另外一个人才能获得咖啡机的使用权。
* 如果用Semaphore的模式，服务员会把咖啡做好放到柜台上，谁想喝咖啡就拿走一杯，服务员会不断做咖啡，如果咖啡杯被拿光了，想喝咖啡的人就排队等着。
* Mutex管理的是咖啡机的使用权，而Semaphore管理的是做好的咖啡数量。

## IPCs in the Linux user space?

|  |  |  |
| --- | --- | --- |
| IPC for PROCESS |  | Communication for THREADS |
| Pipe | Between parent and children |  |
|  |  | Locks: mutex, 条件变量，读写锁  1. 互斥锁提供了以排他方式防止数据结构被并发修改的方法。  2. 读写锁允许多个线程同时读共享数据，而对写操作是互斥的。  3. 条件变量可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。 |
| FIFO |  |  |
| Semaphore | F**or process** s**ynchronization**  For threads synchronization | **Semaphore**  包括无名线程信号量和命名线程信号量 |
| Message Queue | can have many messages |  |
| Shared memory | 共享内存有两种实现方式：  1、内存映射  2、共享内存机制 |  |
| **Signal** |  | **Signal** |
| **socket** |  |  |

## (done)How are function pointers shared across different processes? Using which IPCs?

Two processes cannot share function pointers.   
If you want to use functions in two processes, make library for that functions and use that library in your processes.

## (done)In Linux, how to communicate between kernel space and user space?

In Linux, communication between kernel space and user space can be achieved through a variety of mechanisms, including system calls, ioctl commands, and file operations.

A **system call** is a request made by a user-space program to the kernel for a service or a resource. The program makes the request by calling a specific function, such as **read**() or **write**(), **copy\_to**(), **copy\_from**() which transfers control to the kernel. The kernel then performs the requested service or provides the requested resource and transfers control back to the user-space program.

The **ioctl** (I/O control) command is a device-specific command that can be used to configure or control a device driver. It works by passing a command and an argument to the device driver through the ioctl() system call, which is then handled by the device driver.

File operations such as read and write can be used to read and write data to a file descriptor, this file descriptor is associated with a file, pipe or other objects and can be used to access or share data.

Another way to achieve this is by using the /**proc** file system which is a virtual file system in Linux kernel which is used to provide information about the system and to provide a means of communication between kernel and user space.

## (done)Linux kernel中的同步机制

Diagram

Description automatically generated

## (done)Difference between semaphore and mutex

**Mutex:**  
Is a key to a toilet. One person can have the key - occupy the toilet - at the time. When finished, the person gives (frees) the key to the next person in the queue.  
  
Officially: "Mutexes are typically used to serialize access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section."  
Ref: Symbian Developer Library  
  
(A mutex is really a semaphore with value 1.)  
  
**Semaphore:**Is the number of free identical toilet keys. Example, say we have four toilets with identical locks and keys. The semaphore count - the count of keys - is set to 4 at beginning (all four toilets are free), then the count value is decremented as people are coming in. If all toilets are full, ie. there are no free keys left, the semaphore count is 0. Now, when eq. one person leaves the toilet, semaphore is increased to 1 (one free key) and given to the next person in the queue.  
  
Officially: "A semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore) and can signal that they have finished using the resource (incrementing the semaphore)."  
Ref: Symbian Developer Library

## (done)Difference between binary semaphore and mutex

**Mutex** is for exclusive access to a resource.

A **Binary semaphore** should be used for Synchronization (i.e. "Hey Someone! This occurred!"). The Binary "giver" simply notifies whoever the "taker" that what they were waiting for happened.

Mutex一定要由获得锁的线程来释放。Mutex can be released **only** **by thread that had acquired it**.

Pthreads has 3 different kinds of mutexes: **Fast mutex**, **recursive mutex**, and **error checking mutex**. You used a fast mutex which, for performance reasons, will not check for this error. If you use the error checking mutex on Linux, you will find you get the results you expect.

## (skipped)Mute

//1、使用特定的宏

pthread\_mutex\_t myMutex = PTHREAD\_MUTEX\_INITIALIZER;

//2、调用初始化的函数

pthread\_mutex\_t myMutex;

pthread\_mutex\_init(&myMutex , NULL);

int pthread\_mutex\_lock(pthread\_mutex\_t\* mutex); //实现加锁

int pthread\_mutex\_trylock(pthread\_mutex\_t\* mutex); //实现加锁

int pthread\_mutex\_unlock(pthread\_mutex\_t\* mutex); //实现解锁

## (skipped)Semaphore

## (done) Socket

Diagram

Description automatically generated

## (done)Kernel mutual exclusion:

|  |  |
| --- | --- |
| **Semaphore** | Can cause sleep. It is implemented based on P and V function. Read/write semaphore is also supported.  down()  down\_interruptible() |
| **Completion** | **Why do we need completion?**  If significant contention for the semaphore exists, performance suffers.  Completion can wake up one/more tasks waiting for a resource. |
| **Spinlock** | **1) Why do we need spinlock?**  Because sometimes we can NOT sleep  **2) How is spinlock implemented?**  It is usually implemented as a single bit in an integer value. If the lock is available, the “locked” bit is set, and the code continues into the critical section. If, instead,  The lock has been taken by somebody else, the code goes into a tight loop where it repeatedly checks the lock until it becomes available. The loop is the “spin” part of a spinlock.  **3) How to use spinlock properly?**  a) While holding a spinlock, be atomic. It cannot sleep.  Example functions that could cause sleep:  copy\_to\_user() / copy\_from\_user()  b) Disable interrupt if an interrupt uses the spinlock shared with some driver code.  c) spinlocks must be held for the minimum time possible.  d) If one function acquires a lock and then calls another function that also attempts to acquire the lock, your code deadlocks.  e) Lock ordering rules:  when multiple locks must be required, they should always be acquired in the same order. |
| **Lock-free algorithm** | Circular buffer |
| **Atomic variable** | **Why do we use atomic variable?**  A full locking regime seems like overhead for a simple integer value. |
| **Seqlocks** | **When should we use seqlocks?**  Seqlocks work in situations where the resource to be protected is small, simple, and frequently accessed,  And where write access is rare but must be fast. |
| **Read-Copy-Update** | It is for situations where reads are common, and writes are rare. The resources being protected should be accessed via pointers, and all references to those resources must be held only by atomic code.  When the data structure needs to be changed, the writing thread makes a copy, changes the copy, then aims the relevant pointer at the new version-thus, the name of the algorithm. When the kernel is sure that no references to the old version remain, it can be freed. |

# kernel

## (06/28 good)As kernel can access user space memory, why should copy\_from\_user/copy\_to\_user is needed?

We can use the function to check whether the user space pointer is valid.

The user pages being addressed might not be currently present in the memory (s.g. must be retrieved from swap space)

**For more details: please refer to ldd3 chapter 3 p64**

## (06/29 good) System call

What’s the system call?

System call is the programming interface to Request a service from the kernel.

Kernel provides a set of functions known as system calls, that allow processes to interact with the underlying hardware and access different operating system resources.

Example system call functions：

Fork(), open(), read(), write(), close(), socket()

How does the system call gets implemented in Linux for ARM?

In ARM world, you do a software interrupt (mechanism to signal the kernel) to request a system call.

For ARM EABI, use SWI 0 to make the system call; use r7 to pass the system call number.

Use r0-r6 to pass the arguments.

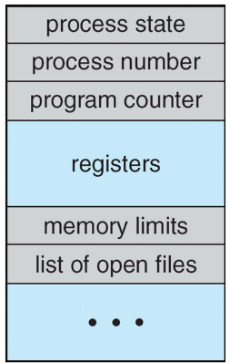
It is implemented by using SWI(software interrupt). The SWI instruction triggers a software interrupt, which transfers control from the user mode to the kernel mode. This allows user applications to request services from the operating system.

1. User Application Setup: **The user application prepares the system call parameters and places them in specific registers** as per the ARM calling convention. **The system call number is usually placed in the R7 register.**
2. Triggering the Software Interrupt**: The user application executes the SWI instruction, which generates a software interrupt** and transfers control to a predefined exception vector address in the kernel.
3. Kernel Mode Execution: The processor switches from user mode to privileged kernel mode, and the kernel takes control. The exception vector points to a specific handler within the kernel.
4. Interrupt Handling: **The SWI handler within the kernel examines the system call number provided by the user application. It uses this number to determine which system call service to execute.**
5. Execution of the System Call: **The SWI handler dispatches the system call to the appropriate function within the kernel.** The kernel executes the requested operation, such as file I/O, process management, or memory management.
6. Result Return: After completing the requested operation, the kernel places the return value (if any) in a designated register, **typically R0.** Additional data, such as error codes, may also be returned in other registers.
7. User Application Resumption: The kernel mode execution ends, and the processor switches back to user mode. The user application continues execution from the instruction following the SWI instruction.

It's worth noting that the specific implementation details of system calls can vary depending on the ARM architecture version, the Linux distribution, and the specific kernel version. **However, the overall concept of using software interrupts to transition from user mode to kernel mode remains consistent across ARM-based Linux systems.**

## (06/29 good) What’s the PCB and what’s context switching

PCB: process control block



**In operating systems, a Process Control Block (PCBis a data structure that contains information about a specific process or task.** Each process in an operating system is associated with its own PCB, which is created and managed by the operating system.

1. **Process state:** This indicates the current state of the process, such as running, waiting, ready, or terminated.
2. **Process identifier** (PID): A unique identification number assigned to each process by the operating system.
3. **Program counter (PC):** The address of the next instruction to be executed by the process.
4. **CPU registers:** The values of the CPU registers at the time of process interruption or context switch. This includes general-purpose registers, stack pointers, and program status registers.
5. **Memory management information**: Information about the memory allocated to the process, such as the base address and limit of its memory segment.
6. **Scheduling information**: This includes the priority of the process, its scheduling history, and any other information needed by the scheduler to determine the order in which processes are executed.
7. **I/O status information**: Details about any I/O devices or files currently being used by the process, including **open files,** pending I/O requests, and I/O device status.
8. **Accounting information**: Statistics related to the process, such as the amount of CPU time used, execution time, and time limits.

The operating system uses the PCB to manage the execution of processes efficiently. **When a context switch occurs, the contents of the CPU registers are saved in the PCB of the currently running process, and the PCB of the next process to be executed is loaded into the registers**. **This allows the operating system to resume the execution of processes from the point where they were interrupted.**

Overall, the PCB is a crucial data structure used by the operating system to track and manage processes effectively, enabling multitasking and resource allocation in a coordinated manner.

## (06/29 good)Difference between Process and Thread (good answer 06/29)

|  |  |  |
| --- | --- | --- |
| Difference | Process | Thread |
| **definition** | It is an independent program under execution.  (Another way to say is- an independent instance of a running program) | It is a basic unit of execution with in a process. |
| **Resource allocation** | Background information:  Int global\_variable c = 5; //stored data section  Int calc(void)  {  Int a = 0; //stored in stack  Int b = 0; //stored in stack  Int sum = 0; //stored in stack  Int \*p\_arr = (int \*)malloc(sizeof(int) \*10); //stored in heap  Memset(p\_arr,0,10);  Sum = a+b+c;  Return sum; //C function is based on using stack  }  The C language will be translated into assembly code, the calculation using PC to fetch the instruction, data is stored in registers for calculation.  The above function is part of the text section.  **Processes have their own individual resources:**  Text section  Data section  Heap section  Stack section  Register  Open files handles  Open network connections | Share below resources:  Text section  Heap section  Data section(We can use global variables for threads to do the communication)  Open files  Network connection  Do NOT share below resources:  Stack section  Register  Stack |
| **Overhead** | * + 1. Creating a new process uses more system resources, memory and more time-consuming compared to create a thread.     2. More overhead for context switching for processes as process do not share any resources compared to threads, we need to store more things before switching to another process. | 1) Less overhead to create a thread compared to create a process as it requires less new system resources and memory.  2) Less overhead for context switching for threads compared to processes as threads share many resources. |
| **Concurrency and Parallelism/communication** | IPC communication:  -Pipe  -signal  -shared memory  - Message queue  -Semaphore  -Socket  Process can execute in parallel on multiple core, but involve more overhead for inter-process communication. | Thread within a process can execute concurrently on multiple CPU cores(improve performance)  Threads share memory, so it does not need IPC communication. |

## (done)Explain the Linux task scheduler? Is Linux RTOS?

Since Linux 2.6.23, the **default scheduler** is CFS (“Completely Fair Scheduler"), it is the default Linux time-sharing scheduling. The static priority is 0.

We can also **configure process** to be scheduled under one of the real-time policies (FIFO scheduling, Round-Robin scheduling) based on static priority ranging from 0 to 99;

**Each thread** has related scheduling policy and a static scheduling priority.

The scheduler makes its decisions **based on** what kind of scheduling policy and static priority of all threads on the system.

Linux kernel is a fully preemptive kernel.

The Linux kernel task scheduler is responsible for managing the execution of processes and threads on a Linux system. It determines which tasks (processes or threads) should run and for how long, taking into account factors such as CPU utilization, task priorities, and fairness.

The Linux kernel uses a scheduler called the Completely Fair Scheduler (CFS) as the default scheduler starting from version 2.6.23. The CFS is a time-sharing scheduler that aims to provide fairness and good interactive response times for tasks running on the system.

The CFS scheduler treats all tasks as if they were running continuously, even though they are time-sliced. It maintains a red-black tree data structure to keep track of the tasks and their priorities. The tasks with higher priorities are given more CPU time than those with lower priorities.

The CFS scheduler uses a concept called "virtual runtime" (vruntime) to determine the next task to run. The vruntime represents the amount of time a task has consumed on the CPU relative to its priority. Tasks with smaller vruntime values are considered to have consumed less CPU time and are given higher priority to run.

When a task is scheduled to run, it is allocated a time slice called a "timeslice" or "sched\_latency\_ns," which is typically around a few milliseconds. If a task exhausts its entire timeslice without blocking or voluntarily yielding the CPU, it is preempted and moved to the end of the runqueue to allow other tasks to run.

The CFS scheduler also takes into account various factors such as load balancing across CPUs, CPU affinity (binding tasks to specific CPUs), and dynamic adjustments based on the system load and task characteristics.

Apart from the CFS scheduler, Linux also provides other scheduling policies like the real-time scheduler (SCHED\_FIFO and SCHED\_RR) for time-critical tasks and the deadline scheduler (SCHED\_DEADLINE) for tasks with hard deadlines.

## (done) Difference between Linux and other RTOS (e.g. VxWorks)?

Since Linux 2.6.23, Linux supports real-time policies based on static priority ranging from 0 to 99;

Linux also provides Time-sharing scheduling;

RTOS schedules tasks based on task priority (e.g. VxWorks).

## (6/29 good) Priority inversion?

##### 简单解释

**什么是优先级反转？**

所谓的优先级翻转问题：即当一个高优先级线程通过信号量机制访问共享资源时，该型号量以被一个低优先级线程占有，而这个低优先级的任务在访问共享资源时可能又被一个中等优先级任务抢占。从上面的描述，高优先级线程被许多较低优先级的任务阻塞，导致高优先级的实时性得不到保证。

解决方案：  
（1）**设置优先级上限**，给临界区一个高优先级，进入临界区的进程都将获得这个高优先级，如果其他试图进入临界区的进程的优先级都低于这个高优先级，那么优先级反转就不会发生。

（2）**优先级继承**，当一个高优先级进程等待一个低优先级进程持有的资源时，低优先级进程将暂时获得高优先级进程的优先级别，在释放共享资源后，低优先级进程回到原来的优先级别。嵌入式系统VxWorks就是采用这种策略。

Priority inversion is where a high priority task is blocked by a low priority task which holds the shared resource (e.g. semaphore) it is trying to use; but then the low priority task is preempted by a medium priority task which can execute without using the shared resource.

How to resolve priority inversion:

* Use priority inheritance: Raise the low priority task to the high priority when it executes the critical section.
* Priority ceiling protocol: This protocol assigns a fixed priority to a shared resource, which is higher than any task accessing that resource. When a task needs to access the resource, it temporarily raises its priority to the priority ceiling, preventing priority inversion.

##### 详细解释

下面这个时序图就是一个经典的优先级反转

Timeline

Description automatically generated with medium confidence

线程A在一个比较低的优先级上工作, 假设是10吧。然后在时间点T1的时候，线程A锁定了一把互斥锁，并开始操作互斥数据。

这时有个高优线级线程C（比如优先级20）在时间点T2被唤醒，它也也需要操作互斥数据。当它加锁互斥锁时，因为互斥锁在T1被线程A锁掉了，所以线程C放弃CPU进入阻塞状态，而线程A得以占据CPU，继续执行。

事情到这一步还是正确的，虽然优先级10的A线程看上去抢了优先级20的C线程的时间，但因为程序逻辑，C确实需要退出CPU等完成互斥数据操作后，才能获得CPU。

但是，假设我们有个线程B在优先级15上，在T3时间点上醒了过来，因为他比当前执行的线程A优先级高，所以它会立即抢占CPU。而线程A被迫进入READY状态等待。

一直到时间点T4，线程B放弃CPU，这时优先级10的线程A是唯一READY线程，它再次占据CPU继续执行，最后在T5解锁了互斥锁。

在T5，线程A解锁的瞬间，线程C立即获取互斥锁，并在优先级20上等待CPU。因为它比线程A的优先级高，系统立刻调度线程C执行，而线程A再次进入READY状态。

上面这个时序里，线程B从T3到T4占据CPU运行的行为，就是事实上的优先级反转。一个优先级15的线程B，通过压制优线级10的线程A，而事实上导致高优先级线程C无法正确得到CPU。这段时间是不可控的，因为线程B可以长时间占据CPU（即使轮转时间片到时，线程A和B都处于可执行态，但是因为B的优先级高，它依然可以占据CPU），其结果就是高优先级线程C可能长时间无法得到 CPU。

上面所说的美国宇航局的火星车，就是因为有高优先级的线程被压制，从而在指定时间内无法获得CPU，导致 “看门狗”认为系统出了无法恢复的故障，直接重启了系统。重启后系统再次进入相同状态，导致不断重启，无法正常工作。

##### 解决方案

1. 优先级继承。当一个高优先级的进程等待一个低优先级的进程时，低优先级进程将临时获取高优先级进程的优先级别，在释放共享资源后，低优先级进程将回到原来的优先级别。

2. 设置优先级上限。给临界区一个高优先级，进入临界区的进程都将获取这个高优先级，如果其他试图进入临界区的进程的优先级都低于这个高优先级，那么优先级反转就不会发生。

## fopen vs open

|  |  |  |  |
| --- | --- | --- | --- |
|  | With buffer | System call |  |
| Open | No | Yes |  |
| Fopen | Yes | No |  |

## (06/29 good)kernel thread vs user thread

Why do we need kernel thread?

* Kernel needs multiple threads to implement different tasks:

#Kernel threads responsible for memory management

# Kernel threads within the device drivers handle low-level device operations, such as sending and receiving data, handling interrupts, and managing device-specific tasks.

#Interrupt handling: Kernel threads play a vital role in handling hardware interrupts. When an interrupt occurs (such as a key press or network packet arrival), a kernel thread is responsible for servicing that interrupt. It processes the interrupt, determines the appropriate action, and invokes the necessary kernel functions to handle the event.(refer to ChatGPT if you want to know more details)

* Kernel thread can be used as daemon, used to manage the hardware/software resources, responding the user requests.
* Kernel thread can NOT be suspended.
* Only running on kernel layer, can NOT access to user space.
* Can created by Kthread\_create() and kthread\_run()

## (06/29 good) Critical section

**What’s critical section?**

* The critical section cannot be executed by more than one process. It typically is a part of a program which accesses a shared resource.

**What’s race condition?**

* Race condition occurs when two or more processes can access the shared data and they try to manipulate it at the same time. the result depends on the order of manipulation. Data inconsistency may arise.

**How to protect critical section?**

* Use mutual exclusion mechanism (e.g., mutex) to protect critical section.

## (done)Deadlock

What’s Deadlock

* There’s a set of blocked processes. Each process is holding a resource and is waiting to require another resource which is held by another process in the set.

4 conditions which cause deadlock:

* **Mutual exclusion** for the shared resource.
* **Hold and wait**: a process is holding at least one resource and waiting to request another resource which is held by another process.
* **No preemption** on the shared resource.
* Circular wait

How to detect deadlock

* Use magic key to detect deadlock in Linux.

How to avoid deadlock:

* Mutual exclusion: not required for sharable resources; only required for non-sharable resource.
* Hold and wait: guarantee whenever a process requests a resource, it does NOT hold ANY other resources;
* No preemption:
* Make resources preemptable.
* If a process that is holding some resources and is trying to request another resource which cannot be allocated immediately, then this process releases all the resources it currently holds.

## (done)Circular Wait

Impose a total ordering for all resources and require each process requests resources in an increasing order.

## (done)What’s Starvation?

When a low priority process does not get access to the resources it needs because there’s a high priority process accessing the resources. The low priority process does not get executed.

## (done)MMAP vs ioremap

* mmap是将设备内存线性地址映射到用户地址空间
* ioremap是将物理地址转换为虚拟地址
  + linux的线程只能访问虚拟地址，不管是不是内核，ioremap应用，比如有个寄存器地址是
  + 你要用ioremap映射后，才能访问地址。
  + 这两个地址是不同的，mmu会帮你搞定，对你是透明的

## (done)MMU?

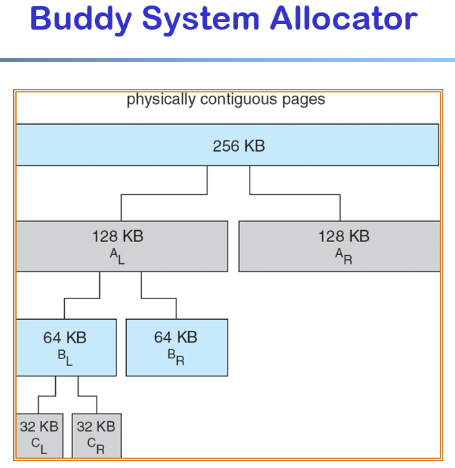
Basic concepts:

* Logical address (virtual address): generated by the CPU
* Physical address: address seen by the memory unit
* Logical and physical addresses are the same in the compile-time and load-time address-binding schemes.
* Logical(virtual) and physical addresses differ in execution-time address-binding scheme.
* MMU: hardware device that maps virtual address to physical address.

Memory fragmentation:

* External fragmentation: Total memory space exists to satisfy a request, but it is not contiguous.
* Internal fragmentation: Memory is usually allocated in “blocks”; allocated memory may be a bit larger than requested memory.

Paging:

* Set up a page table to translate logical to physical addresses (internal fragmentation is still possible)
* 

## (done)Zombie process and orphan process:

unix提供了一种机制可以保证只要父进程想知道子进程结束时的状态信息，就可以得到。

这种机制就是: 在每个进程退出的时候, 内核释放该进程所有的资源, 包括打开的文件, 占用的内存等。 但是仍然为其保留一定的信息(包括进程号the process ID,退出状态the termination status of the process,运行时间the amount of CPU time taken by the process等)。直到父进程通过wait / waitpid来取时才释放。 但这样就导致了问题，**如果进程不调用wait / waitpid的话， 那么保留的那段信息就不会释放，其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵死进程，将因为没有可用的进程号而导致系统不能产生新的进程. 此即为僵尸进程的危害，应当避免。**

Zombie (子进程自己先退出，但是父进程没有调用wait/waitpid来获取子进程的状态信息)

* A process which has **finished the execution** but **still has entry in the process table** **to report to its parent process** is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.
* Solution:

Child process send a SIGCHILD signal when it’s exiting, then parent process deal with the SIGCHILD by using Wait/waitpid.

 Orphan Process: (父进程先退出，但是子进程还在运行)

* A process whose parent process no more exists i.e., either finished or terminated without waiting for its child process to terminate is called an orphan process. (**wait**()/**waitpid**())

## What are the process items that are inherited from the process’s parent?

the child gets a copy of the parent's data space, heap, and stack, all file descriptors that are open in the parent are duplicated in the child.

the parent and the child **do not** share these portions of memory.

## (done) VFORK () vs fork ()

The basic difference between vfork and fork is that when a new process is created with vfork (), the parent process is suspended, and the child process might borrow the parent's address space.

The intent of vfork was to eliminate the overhead of copying the whole process image if you only want to do an exec\* in the child. Because exec\* replaces the whole image of the child process, there is no point in copying the image of the parent.

## (done)[What is the difference between a process and a thread?](https://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread)

An executing instance of a program is called a process. A thread is a subset of the process.

The typical difference is that threads of the same process executes within the context of a process and shares the same resources allotted to the process by the kernel, while process run in separate memory spaces.

Threads share all segments except stack. Threads have independent call Stacks, program counter(PC), Registers, State.

## (done)Do you know the segments in which a program gets divided?

Stack, Heap, BSS, Data (global or static variables), text.

## ****(done) which segments do threads share?****

threads share all segments except the stack.

## (done)Reentrant function

Do not access mutable global or function-static variables.

Do not self-modify code.

Do not invoke another function that is itself non-reentrant.

## （done）Exec and system

exec: The exec function replaces the currently running process image when successful, no child is created (unless you do that yourself previously with fork ()).

System: The system () function does fork a child process and returns when the command supplied is finished executing or an error occurs.

clone: is the system call used by fork ().

## (done)Linux的system() 和popen()差异

### summarize

在Linux中我们可以通过system()来执行一个shell命令，popen()也是执行shell命令并且通过管道和shell命令进行通信。

system()、popen()给我们处理了fork、exec、waitpid等一系列的处理流程，让我们只需要关注最后的返回结果(函数的返回值)即可。

### 执行流程

从上面的源码可以看到system和popen都是执行了类似的运行流程，大致是fork->execl->return。但是我们看到system在执行期间调用进程会一直等待shell命令执行完成(waitpid等待子进程结束)才返回，但是popen无须等待shell命令执行完成就返回了。我们可以理解system为串行执行，在执行期间调用进程放弃了”控制权”，popen为并行执行。

popen中的子进程没人给它” 收尸” 了啊？是的，如果你没有在调用**popen**后调用**pclose**那么这个子进程就可能变成” **僵尸**” 。

上面我们没有给出pclose的源码，其实我们根据system的源码差不多可以猜测出pclose的源码就是system中第4部分的内容。

### 信号处理

我们看到system中对SIGCHLD、SIGINT、SIGQUIT都做了处理，但是在popen中没有对信号做任何的处理。

SIGCHLD是子进程退出的时候发给父进程的一个信号，system()中为什么要屏蔽SIGCHLD信号可以参考：system函数的总结、waitpid(or wait)和SIGCHILD的关系，总结一句就是为了system()调用能够及时的退出并且能够正确的获取子进程的退出状态(成功回收子进程)。

popen没有屏蔽SIGCHLD，主要的原因就是popen是”并行”的。如果我们在调用popen的时候屏蔽了SIGCHLD，那么如果在调用popen和pclose之间调用进程又创建了其它的子进程并且调用进程注册了SIGCHLD信号处理句柄来处理子进程的回收工作(waitpid)那么这个回收工作会一直阻塞到pclose调用。这也意味着如果调用进程在pclose之前执行了一个wait()操作的话就可能获取到popen创建的子进程的状态，这样在调用pclose的时候就会回收(waitpid)子进程失败，返回-1，同时设置errno为ECHLD，标示pclose无法获取子进程状态。

system()中屏蔽SIGINT、SIGQUIT的原因可以继续参考上面提到的system函数的总结，popen()函数中没有屏蔽SIGINT、SIGQUIT的原因也还是因为popen是”并行的”，不能影响其它”并行”进程。

### 4. 功能

从上面的章节我们基本已经把这两个函数剖析的差不多了，这两个的功能上面的差异也比较明显了，system就是执行shell命令最后返回是否执行成功，popen执行命令并且通过管道和shell命令进行通信。

### NOTE

在特权(setuid、setgid)进程中千万注意不要使用system和popen。

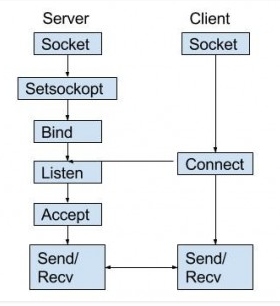
(done)big-endian and little-endian

Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first.

(done) 5 stages pipeline

Fetch Decode execute memory write

(done) Socket programming



int sockfd = socket(domain, type, protocol)

**sockfd:** socket descriptor, an integer (like a file-handle)  
**domain:** integer, communication domain e.g.,

AF\_INET (IPv4 protocol)

AF\_INET6 (IPv6 protocol)  
**type:** communication type

SOCK\_STREAM: TCP (reliable, connection oriented)

SOCK\_DGRAM: UDP (unreliable, connectionless)  
**protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet. (man protocols for more details)

# command(done)

Kill -9 vs kill -6 vs killall/pkill

Kill command is used to kill a process by sending a signal to kernel. We can use “**kill -l**” to display parameters.

Killall/pkill: can specify process name, and it’ll kill the whole process group.

“Kill -9” is used to send SIGKILL to terminate the specific process immediately.

“kill -6” SIGABRT

ntpdate

network time protocol date

ntpdate pool.ntp.org

# C++ (done)

Major C++ features

**Class**: Class is a blueprint of data and functions or methods. Class does not take any space. It is a way of encapsulating data, defining abstract data types along with initialization conditions and operations allowed on that data; a way of hiding the implementation (hiding the guts & exposing the skin); a way of sharing behavior and characteristics

**Object**: Objects are basic run-time entities in an object-oriented system, objects are instances of a class these are defined user defined data types.

**Overloading**

* Operator overloading: C++ provides ability to overload most operators so that they perform special operations relative to classes. For example, a class String can overload the + operator to concatenate two strings. The conversion operator has no return type.
* Function overloading: Function overloading is the process of using the same name for two or more functions. The function must use either different types of parameters or a different number of parameters.

**Encapsulation and Data abstraction**: Wrapping up(combining) of data and functions into a single unit is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapping in the class can access it. This insulation of the data from direct access by the program is called data hiding or information hiding.

**Data abstraction** – providing only needed information to the outside world and hiding implementation details. For example, consider a class Complex with public functions as getReal () and getImag (). We may implement the class as an array of size 2 or as two variables. The advantage of abstractions is, we can change implementation at any point, users of Complex class won’t be affected as our method interface remains same. Had our implementation be public, we would not have been able to change it.

**Inheritance**: **The purpose of inheritance is Code Reuse.** Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. Inheritance provides reusability. This means that we can add additional features to an existing class without modifying it.

**Polymorphism**: Polymorphism means ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.

**Dynamic Binding**: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

**Message Passing**: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

**Static Member**: Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following,

* Interesting facts about Static Members Functions in C++
  + static member functions do not have this pointer.
  + A static member function cannot be virtual
  + Member function declarations with the same name and the name parameter-type-list cannot be overloaded if any of them is a static member function declaration.
  + static member function cannot be declared const, volatile, or const volatile.

What are the differences between C and C++?

1) C++ is a kind of superset of C, most of C programs except few exceptions (See this and this) work in C++ as well.

2) C is a procedural programming language, but C++ supports both procedural and Object-Oriented programming.

3) Since C++ supports object-oriented programming, it supports features like function overloading, templates, inheritance, virtual functions, friend functions. These features are absent in C.

4) C++ supports exception handling at language level, in C exception handling is done in traditional if-else style.

5) C++ supports references, C doesn’t.

6) In C, scanf() and printf() are mainly used input/output. C++ mainly uses streams to perform input and output operations. cin is standard input stream and cout is standard output stream.

When should you use new?

You should use new when you wish an object to remain in existence until you delete it. If you do not use new then the object will be destroyed when it goes out of scope. Some examples of this are:

void foo()

{

Point p = Point(0,0);

} // p is now destroyed.

for (...)

{

Point p = Point(0,0);

} // p is destroyed after each loop

Some people will say that the use of new decides whether your object is on the heap or the stack, but that is only true of variables declared within functions.

In the example below the location of 'p' will be where its containing object, Foo, is allocated. I prefer to call this 'in-place' allocation.

class Foo

{

Point p;

}; // p will be automatically destroyed when foo is.

Allocating (and freeing) objects with the use of new is far more expensive than if they are allocated in-place so its use should be restricted to where necessary.

A second example of when to allocate via new is for arrays. You cannot\* change the size of an in-place or stack array at run-time so where you need an array of undetermined size it must be allocated via new.

E.g.

void foo(int size)

{

Point\* pointArray = new Point[size];

...

delete [] pointArray;

}

(\*pre-emptive nitpicking - yes, there are extensions that allow variable sized stack allocations).

What are the differences between a C struct and a C++ struct?

A C struct is just a way of combining data together; it only has characteristics (the data) and does not include behavior (functions may use the structure but are not tied up to it)

Typedefed names are not automatically generated for C structure tags; e.g.,:

// a C struct

struct my\_struct {

int someInt;

char\* someString;

};

// you declare a variable of type my\_struct in C

struct my\_struct someStructure;

// in C you have to typedef the name to easily

// declare the variable

typedef my\_struct MyStruct;

MyStruct someOtherStuct;

// a C++ struct

struct MyCppStruct {

int someInt;

char\* someString;

};

// you declare a variable of type MyCppStruct in C++

MyCppStruct someCppStruct;

// as you can see the name is automatically typedefed.

But what’s more important is that a C struct does not provide enablement for OOP concepts like encapsulation or polymorphism. Also “C structs can’t have static members or member functions”, [bmerry]. A C++ struct is actually a class, the difference being that the default member and base class access specifiers are different: class defaults to private whereas struct defaults to public.

What does the keyword const mean and what are its advantages over #define?

In short and by far not complete, const means “read-only”! A named constant (declared with const) it’s like a normal variable, except that its value cannot be changed. Any data type, user-defined or built-in, may be defined as a const, e.g.,:

// myInt is a constant (read-only) integer

const int myInt = 26;

// same as the above (just to illustrate const is

// right and also left associative)

int const myInt = 26;

// a pointer to a constant instance of custom

// type MyClass

const MyClass\* myObject = new MyObject();

// a constant pointer to an instance of custom

// type MyClass

MyClass\* const myObject = new MyObject();

// myInt is a constant pointer to a constant integer

const int someInt = 26;

const int\* const myInt = &someInt;

#define is error prone as it is not enforced by the compiler like const is. It merely declares a substitution that the preprocessor will perform without any checks; that is const ensures the correct type is used, whereas #define does not. “Defines” are harder to debug as they are not placed in the symbol table.

A constant has a scope in C++, just like a regular variable, as opposed to “defined” names that are globally available and may clash.

A constant must also be defined at the point of declaration (must have a value) whereas “defines” can be “empty.”

Code that uses const is inherently protected by the compiler against inadvertent changes: e.g., to a class’ internal state (const member variables cannot be altered, const member functions do not alter the class state); to parameters being used in methods (const arguments do not have their values changed within methods) [sql\_lall]. A named constant is also subject for compiler optimizations.

In conclusion, you will have fewer bugs and headaches by preferring const to #define.

What are the differences between references and pointers?

Both references and pointers can be used to change local variables of one function inside another function. Both of them can also be used to save copying of big objects when passed as arguments to functions or returned from functions, to get efficiency gain.  
Despite above similarities, there are following differences between references and pointers.

References are less powerful than pointers

* 1) Once a reference is created, it cannot be later made to reference another object; it cannot be reseated. This is often done with pointers.
* 2) References cannot be NULL. Pointers are often made NULL to indicate that they are not pointing to any valid thing.
* 3) A reference must be initialized when declared. There is no such restriction with pointers

Due to the above limitations, references in C++ cannot be used for implementing data structures like Linked List, Tree, etc. In Java, references don’t have above restrictions, and can be used to implement all data structures. References being more powerful in Java, is the main reason Java doesn’t need pointers.

References are safer and easier to use:

* 1) Safer: Since references must be initialized, wild references like wild pointers are unlikely to exist. It is still possible to have references that don’t refer to a valid location (See questions 5 and 6 in the below exercise)
* 2) Easier to use: References don’t need dereferencing operator to access the value. They can be used like normal variables. ‘&’ operator is needed only at the time of declaration. Also, members of an object reference can be accessed with dot operator (‘.’), unlike pointers where arrow operator (->) is needed to access members.

What are virtual functions – Write an example?

Virtual functions are used with inheritance, they are called according to the type of object pointed or referred, not according to the type of pointer or reference. In other words, virtual functions are resolved late, at runtime. Virtual keyword is used to make a function virtual.

Following things are necessary to write a C++ program with runtime polymorphism (use of virtual functions)

1) A base class and a derived class.

2) A function with same name in base class and derived class.

3) A pointer or reference of base class type pointing or referring to an object of derived class.

For example, in the following program bp is a pointer of type Base, but a call to bp->show() calls show() function of Derived class, because bp points to an object of Derived class.

#include<iostream>

using namespace std;

class Base {

public:

virtual void show() { cout<<" In Base \n"; }

};

class Derived: public Base {

public:

void show() { cout<<"In Derived \n"; }

};

int main(void) {

Base \*bp = new Derived;

bp->show(); // RUN-TIME POLYMORPHISM

return 0;

}

Output:

In Derived

What is this pointer?

The ‘this’ pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions.

‘this’ pointer is a constant pointer that holds the memory address of the current object.

‘this’ pointer is not available in static member functions as static member functions can be called without any object (with class name).

What are VTABLE and VPTR?

vtable is a table of function pointers. It is maintained per class.

vptr is a pointer to vtable. It is maintained per object (See this for an example).

Compiler adds additional code at two places to maintain and use vtable and vptr.

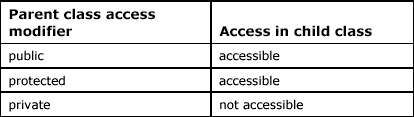
* 1) Code in every constructor. This code sets vptr of the object being created. This code sets vptr to point to vtable of the class.
* 2) Code with polymorphic function call (e.g. bp->show () in above code). Wherever a polymorphic call is made, compiler inserts code to first look for vptr using base class pointer or reference (In the above example, since pointed or referred object is of derived type, vptr of derived class is accessed). Once vptr is fetched, vtable of derived class can be accessed. Using vtable, address of derived derived class function show () is accessed and called.

What are C++ access specifiers?

Access specifiers are used to define how the members (functions and variables) can be accessed outside the class.

Private: Members declared as private are accessible only within the same class and they cannot be accessed outside the class they are declared. Child classes are also not allowed to access private members of parent.

Public: Members declared as public are accessible from anywhere.

Protected: Only the class and its child classes can access protected members.[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/12/table5.6.gif)

explain public and private inheritance?

Public inheritance is the “default” inheritance mechanism in C++ and it is realized by specifying the public keyword before the base class

class B : public A

{

};

Private inheritance is realized by specifying the private keyword before the base class or omitting it completely, as private is the default specifier in C++

class B : private A

{

};

or

class B : A

{

};

The public keyword in the inheritance syntax means that the publicly/protected/privately accessible members inherited from the base class stay public/protected/private in the derived class; in other words, the members maintain their access specifiers. The private keyword in the inheritance syntax means that all the base class members, regardless of their access specifiers, become private in the derived class; in other words, private inheritance degrades the access of the base class’ members – you won’t be able to access public members of the base class through the derived one (in other languages, e.g., Java, the compiler won’t let you do such a thing).

From the relationship between the base and derived class point of view,

class B : public A {}; B "is a" A but class B : private A {};

means B “is implemented in terms of” A.

Public inheritance creates subtypes of the base type. If we have class B : public A {}; then any B object is substituteable by its base calls object (through means of pointers and references) so you can safely write

A\* aPointer = new B();

Private inheritance, on the other hand, class B : private A {};, does not create subtypes making the base type inaccessible and is a form of object composition. The following illustrates that:

class A

{

public:

A();

~A();

void doSomething();

};

void A :: doSomething()

{

}

class B : private A

{

public:

B();

~B();

};

B\* beePointer = new B();

// ERROR! compiler complains that the

// method is not accessible

beePointer->doSomething();

// ERROR! compiler complains that the

// conversion from B\* to A\* exists but

// is not accessible

A\* aPointer = new B();

// ! for the following two the standard

// stipulates the behavior as undefined;

// the compiler should generate an error at least

// for the first one saying that B is not a

// polymorphic type

A\* aPointer2 = dynamic\_cast<A\*>(beePointer);

A\* aPointer3 = reinterpret\_cast<A\*>(beePointer

Malloc () vs new / Delete vs Free

the differences between malloc () and operator new.

* new is an operator, while malloc () is a function.
* new returns exact data type, while malloc () returns void \*.
* new calls constructors (class instances are initialized and deinitialized automatically), while malloc () does not (classes won’t get initialized or deinitialized automatically
* Syntax:
  + int \*n = new int(10); // initialization with new()

str = (char \*) malloc (15); //malloc ()

free () vs delete ()

* free () is used on resources allocated by malloc (), or calloc () in C
* Delete is used on resources allocated by new in C++

Inline Functions

C++ provides an inline function to reduce the function call overhead.

Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

The syntax for defining the function inline is:

inline return-type function-name(parameters)

{

// function code

}

Remember, inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining.

Friend class and function in C++

A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example, a Linked List class may be allowed to access private members of Node.

Friend Function Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:

a) A method of another class

b) A global function

Important points about friend functions and classes:

1) Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.

2) Friendship is not mutual. If a class A is friend of B, then B doesn’t become friend of A automatically.

3) Friendship is not inherited (See this for more details)

4) The concept of friends is not there in Java.

Function overloading VS Operator Overloading

Function overloading is a feature in C++ where two or more functions can have the same name but different type of parameters and different number of parameters.

Note: Overloading of functions with different return types are not allowed.

Operating overloading allows us to make operators to work for user defined classes. For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.

Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Copy Constructor

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype: ClassName (const ClassName &old\_obj);

Point (int x1, int y1) { x = x1; y = y1; }

// Copy constructor

Point (const Point &p2) {x = p2.x; y = p2.y; }

When is copy constructor called?

In C++, a Copy Constructor may be called in following cases:

When an object of the class is returned by value.

When an object of the class is passed (to a function) by value as an argument.

When an object is constructed based on another object of the same class.

When compiler generates a temporary object.

Can we make copy constructor private?

Yes, a copy constructor can be made private

When “must” you use a constructor initialization list?

Constant and reference data members of a class may only be initialized, never assigned, so you must use a constructor initialization list to properly construct (initialize) those members.

In inheritance, when the base class does not have a default constructor, or you want to change a default argument in a default constructor, you must explicitly call the base class’ constructor in the initialization list.

For reasons of correctness – any calls you make to member functions of sub-objects (used in composition) go to initialized objects.

For reasons of efficiency. Looking at the previous question example we could rewrite the SimpleBase constructor as follows:

SimpleBase :: SimpleBase(string &name)

{

m\_name = name;

}

The above will generate a call to the default string constructor to construct the class member m\_name and then the assignment operator of the string class to assign the name argument to the m\_name member. So you will end up with two calls before the data member m\_name is fully constructed and initialized.

SimpleBase :: SimpleBase(string &name) : m\_name(name)

{

}

The above will only generate a single call, which is to the copy constructor of the string class, thus being more efficient.

What are virtual functions and pure virtual functions?

Virtual functions allow the programmer to redefine in each derived class functions from the base class with altered behavior so that you can call the right function for the right object (allow to perform the right operation for an object through only a pointer/reference to that object’s base class)

A pure virtual function (or abstract function) in C++ is a [virtual function](https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/)for which we don’t have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration.

class CShape

{

public:

virtual void Show() **= 0**;

};

What’s the difference between pointer and reference

Reference is to one object cannot be changed to another object

Reference should be initialized always.

Any other Differences??

* POINTER

1) It’s not necessary to initialize the pointer at the time of declaration. Like

int a = 10;

int \*P = &a; //It is not necessary

Another way is :

int a = 10;

int \*P;

P = &a;

2) You can create the array of Pointer.

3) You can assign NULL to the pointer like

int \*P = NULL; //Valid

4) You can use pointer to pointer.

* REFERENCE

1) Its necessary to initialize the Reference at the time of declaration. Like

int &a = 10;

int &a; //Error here but not in case of Pointer.

2) You cannot create the Array of reference.

3) You cannot assign NULL to the reference like

int &a = NULL; //Error

4) You cannot use reference to reference.

Is reference a pointer?

\* Reference is an implicit pointer

\* Must tell it which variable it will become an alias for

\* References do not require dereferencing

\* No acquisition of memory address required

\* References are often implemented by the compiler writers as pointers.

\* References are “safer”

Is there any adv of references over pointers?

\* References do not require dereferencing

\* No acquisition of memory address required

\* References are “safer”

## weak、弱符号

若两个或两个以上全局符号（函数或变量名）名字一样，而其中之一声明为weak属性，则这些全局符号不会引发重定义错误。链接器会忽略弱符号，去使用普通的全局符号来解析所有对这些符号的引用，但当普通的全局符号不可用时，链接器会使用弱符号。

当有函数或变量名可能被用户覆盖时，该函数或变量名可以声明为一个弱符号。

## new vs malloc

“在C++中，内存区分为5个区，分别是堆、栈、自由存储区、全局/静态存储区、常量存储区”。

在C中，则没有自由存储区的概念

1. memory location

1). 自由存储是C++中通过new与delete动态分配和释放对象的抽象概念，而堆（heap）是C语言和操作系统的术语，是操作系统维护的一块动态分配内存。

2). new所申请的内存区域在C++中称为自由存储区。藉由堆实现的自由存储，可以说new所申请的内存区域在堆上。

将上面所述的10点差别整理成表格：

| **特征** | **new/delete** | **malloc/free** |
| --- | --- | --- |
| 分配内存的位置 | 自由存储区 | 堆 |
| 内存分配成功的返回值 | 完整类型指针 | void\* |
| 内存分配失败的返回值 | 默认抛出异常 | 返回NULL |
| 分配内存的大小 | 由编译器根据类型计算得出 | 必须显式指定字节数 |
| 处理数组 | 有处理数组的new版本new[] | 需要用户计算数组的大小后进行内存分配 |
| 已分配内存的扩充 | 无法直观地处理 | 使用realloc简单完成 |
| 是否相互调用 | 可以，看具体的operator new/delete实现 | 不可调用new |
| 分配内存时内存不足 | 客户能够指定处理函数或重新制定分配器 | 无法通过用户代码进行处理 |
| 函数重载(overload) | 允许 | 不允许 |
| 构造函数与析构函数 | 调用 | 不调用 |

## 重写(overwrite)、重载(overload)和覆盖(override)三者之间的区别

1. 覆盖：子类继承了父类的同名**无参**函数。当子类从父类继承了一个无参函数，而又定义了一个同样的无参函数，则子类定义的方法覆盖父类的方法，称为覆盖。

2. 重载：子类继承了父类的同名**有参**函数。当子类继承了父类的一个同名方法，且方法参数不同，称为重载。通过方法的重载，子类可以重新实现父类的某些方法，使其具有自己的特征。

3. 重写：当前类的同名方法。通过方法的重写，一个类可以有多个具有相同名字的方法，由传递给它们不同的个数和类型的参数来决定使用哪种方法。因此，重写的名称是**当前类中的同名函数**，不是父类中的函数名。

4. 隐藏：子类中定义了与父类中的同名函数，但是参数不同。那么如果从子类中和从子类的实例中调用父类中的该函数（使用父类中该函数的参数类型）会失败。——编译器会从子类开始，向上查找函数的定义。一旦找到了同样名字的函数，就会使用它。结果是父类中的该函数看起来被影藏了。

## Virtual and pure virtual

Virtual function:

Can be rewrite Child class

Pure virtual function:

Virtual void fun() = 0

child class