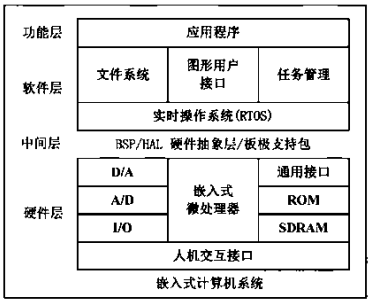
题型：填空 3*12=36’,简答 6 题 34’，程序设计 3 题 40’

1.1 嵌入式系统的基本组成

➢**硬件组成**：处理器/微处理器、存储器、I/O 接口及输入/输出设备；
软件组成：嵌入式操作系统、应用程序

嵌入式系统的软/硬件框架



嵌入式系统的软件组成

1.嵌入式操作系统
2.嵌入式应用软件
3.硬件抽象层HAL
4.板级支持包BSP
5.设备驱动程序
6.操作系统的应用程序接口函数API

1.2 嵌入式系统中，微处理器控制 I/O 端口或部件的数据传送方式（结合串口工作方式，2 种）

➢**中断方式和程序查询方式**
程序查询方式是由微处理器周期性地执行一段查询程序来读取 I/O 端口或部件中状态寄存器的内容，并判断其状态，从而使微处理器与 I/O 端口或部件在进行数据、命令传送时保持同步。程序查询方式下效率非常低，因为微处理器要花费大量时间测试 I/O 端口或部件的状态。并且 I/O 端口或部件的数据也不能得到实时地处理。
中断方式是 I/O 端口或部件在完成了一个 I/O 操作后，产生一个信号给微处理器，这个信号叫做“中断请求”，微处理器响应这个请求信号，停止其当前的程序操作，而转向对该 I/O 端口或部件进行新的读/写操作，特点是实时性能好。

1.3 为何采用交叉开发？（交叉开发是什么）

➢**交叉开发**是指在通用电脑上把程序编写、编译、调试好，再下载到嵌入式产品中去运行。
为何采用：嵌入式系统是计算机专用的系统。由于嵌入式系统硬件上的特殊性，一般不能安装发行版的 Linux 系统。例如 flash 储存空间很小，没有足够的空间安装，或者处理器很特殊，没有发行版的 linux 可用。所以需要专门为特定的目标版指定 linux 操作系统，这必然需要相应的开发环境，于是人们想到了交叉开发模式。
简略版：由于嵌入式系统资源匮乏，一般不能像 PC 一样安装本地编译器和调试器，不能在本地编写、编译和调试自身运行的程序，而需借助其它系统如 PC 来完成这些工作。

1.4 嵌入式设计的几个阶段及主要目标（主要是硬件设计）

➢**系统方案分析与设计阶段**：根据系统所要完成的功能，选择合适的处理器和外围部件，完成系统的功能框图设计

PCB 设计仿真阶段：在 EDA 仿真设计平台下，进行系统原理图及 PCB，并对 PCB 板上的信号完整性、EMI 等进行仿真，根据仿真结果来对 PCB 进行合理的布局布线调整，完成 PCB 的设计。

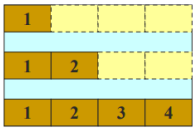
PCB 的加工、测试：对加工完成的 PCB 进行器件焊接、调试和测试，完成整个系统硬件的设计。

2.1 ARM 系列微处理器支持的数据类型

➢字节（8 位） 半字（16 位） 字（32 位）

ARM处理器支持下列数据类型：

- 字节 8位
- 半字 16位（必须分配为占用两个字节）
- 字 32位（必须分配为占用4各字节）



§ 所有数据操作，例如 ADD，都以字为单位；

§ 装载和保存指令可以对字节、半字和字进行操作，当装载字节或半字时自动实现零扩展或符号扩展；

§ ARM 指令的长度刚好是 1 个字（分配为占用 4 个字节），Thumb 指令的长度刚好是半字（占用 2 个字节）

2.2 ARM 处理器有几种状态，如何切换？（2 种）

➢ARM7TDMI 处理器内核使用 V4T 版本的 ARM 结构，该结构包含 32 位 ARM 指令集和 16 位 Thumb 指令集。因此 ARM7TDMI 处理器有**两种操作状态：**

§ ARM 状态：32 位，这种状态下执行的是字方式的 ARM 指令；

§ Thumb 状态：16 位，这种状态下执行半字方式的 ARM 指令。

注意：两个状态之间的切换并不影响处理器模式或寄存器内容。

切换：使用 BX 指令将内核的操作状态在 ARM 状态和 Thumb 状态之间进行切换。

```
;从Arm状态切换到Thumb状态
    LDR     R0,=Lable+1
    BX      R0
;从Thumb状态切换到ARM状态
    LDR     R0,=Lable
    BX      R0
```

跳转地址标号

地址最低位为1，表示切换到Thumb状态

地址最低位为0，表示切换到ARM状态

2.3 ARM 处理器有哪些工作模式？（7 种）用户模式与特权模式？

➢7 种工作模式，如下所示。

处理器模式	说明	备注
用户 (usr)	正常程序工作模式	不能直接切换到其它模式
系统 (sys)	用于支持操作系统的特权任务等	与用户模式类似，但具有可以直接切换到其它模式等特权
快中断 (fiq)	支持高速数据传输及通道处理	FIQ异常响应时进入此模式
中断 (irq)	用于通用中断处理	IRQ异常响应时进入此模式
管理 (svc)	操作系统保护代码	系统复位和软件中断响应时进入此模式
中止 (abt)	用于支持虚拟内存和/或存储器保护	在ARM7TDMI没有大用处
未定义 (und)	支持硬件协处理器的软件仿真	未定义指令异常响应时进入此模式

特权模式：除**用户模式**外，其他模式均为**特权模式**。ARM 内部寄存器和一些片内外设在硬件设计上只允许特权模式下访问。特权模式可自由切换处理器模式，用户模式不能直接切换到别的模式。

2.4 LR 寄存器的作用（R14，哪个寄存器+作用）

➢R14 为**链接寄存器**（LR），在结构上有**两个特殊功能**：

§在每种模式下，模式自身的 R14 版本用于保存**子程序返回地址**；

§当发生异常时，将 R14 对应的异常模式版本设置为**异常返回地址**（有些异常有一个小的固定偏移量）。

2.5 CPSR 与 SPSR？

➢寄存器 CPSR 为**程序状态寄存器**，CPSR 反映了当前处理器的状态，其包含：4 个条件代码标志：负(N)、零(Z)、进位(C)和溢出(V)；2 个中断禁止位，分别控制一种类型的中断；5 个对当前处理器模式进行编码的位；1 个用于指示当前执行指令(ARM 还是 Thumb)的位。

在异常模式中，另外一个寄存器“**程序状态保存寄存器**（SPSR）”可以被访问。每种异常都有自己的 SPSR，在进入异常时它保存 CPSR 的当前值，异常退出时可通过它恢复 CPSR。

CPSR 和 SPSR 通过特殊指令（MRS、MSR）进行访问。

2.6 大端模式和小端模式（存储都按字节 8 位存储）

➢**大端模式**：最高有效字节位于最低地址：字数据的高字节存储在低地址中，低字节存放在高地址中；

小端模式：最低有效字节位于最低地址：低地址中存放字数据的低字节，高地址存放字数据的高字节；

0x12345678字数据的大小端存储方式

2.7 异常，进入和退出异常（了解）

➢**异常**是由内部或者外部原因引起的，当异常发生时 CPU 将暂停执行当前指令自动到指定的向量地址读取指令并且执行。

进入异常：在异常发生后，ARM7TDMI 内核会作以下工作：

1、在适当的 LR 中保存下一条指令的地址;

2、将 CPSR 复制到适当的 SPSR 中;

3、将 CPSR 模式位强制设置为与异常类型相对应的值;

4、强制 PC 从相关的异常向量处取指。

退出异常：当异常结束时，异常处理程序必须：

1、将 LR(R14)中的值减去偏移量后存入 PC，偏移量根据异常的类型而有所不同;

2、将 SPSR 的值复制回 CPSR;

3、清零在入口置位的中断禁止标志。

3.1 流水线，作用（流水线是什么，作用：提高…）

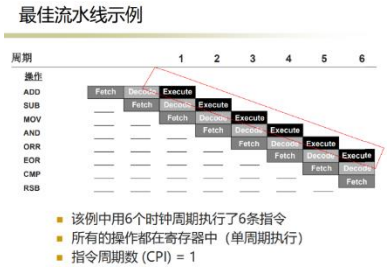
➢**流水线(Pipeline)技术**：几个指令可以并行执行。

指令流水线：是为提高处理器执行指令的效率，把一条指令的操作分成多个细小的步骤，每个步骤由专门的电路完成的方式。

流水线的执行顺序：取指令->译码->执行

作用：提高了 CPU 的运行效率；内部信息流要求通畅流动

3.2 利用流水线计算指令的执行时间（填空题）



执行时间 = 单条指令所需时间 +（n-1）×（流水线周期） （n 为指令的个数）

流水线周期指指令分段执行中时间最长的一段

3.3 并行与串行，同步与异步的基本概念

➢**并行通信**：将数据的各位用多条数据线同时进行传送，外加地址线和通信控制线。优点是传输速率高，缺点是长距离传输成本高，可靠性差，只适用于近距离

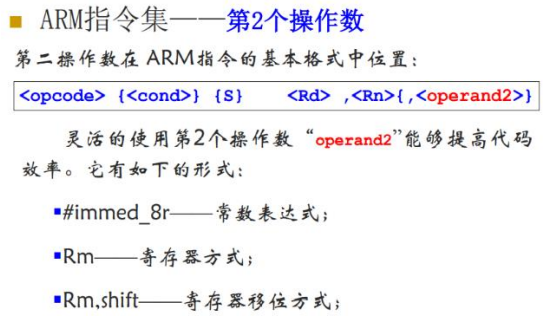
传输。
串行通信：将数据逐位按顺序在一条传输线上传送。优点是传输线少，长距离传送时成本低，缺点是传输速率低。
根据数据传输方式的不同，可将串行通信分为同步通信和异步通信。
同步传输时发送方发送时钟，接收方的时钟和发送方是同一个时钟；
异步传输时发送方不需发送时钟，并不要求接收方的时钟与发送方完全一样。

3.4 IIC,SPI 总线的特点（物理组成上的特点，了解）

IIC：一种用于内部 IC 控制的简单的双向两线串行总线，最高速率 100Kbps，25 英尺，最多可支持 40 个设备。
SPI：串行外设接口(Serial Peripheral Interface)，是一种高速的，全双工，同步的通信总线。
异同：（SDA 数据线,SCL 时钟线）
IIC 总线不是全双工（半双工），2 根线 SCL、SDA。SPI 总线实现全双工，4 根线。
IIC 总线是多主机总线，通过 SDA 上的地址信息来锁定从设备。SPI 总线只有一个主设备，主设备通过 CS 片选来确定从设备
IIC 总线传输速度在 100kbps-4Mbps。SPI 总线传输速度更快，可以达到 30Mbps 以上。
IIC 总线和 SPI 总线时钟都是由主设备产生，并且只在数据传输时发出时钟
IIC 总线读写时序比较固定统一，SPI 不同从设备读写时序差别比较大

4.1 第二操作数，遵循的原则，分析原因？

➢在算术指令中，可以将第二操作数在运算之前移位（如：LDREQ，R0,[R1,R2,LSR #16]!）。移位通过组合电路完成，而不需时钟脉冲的作用，不影响指令执行时间。
每条指令由：操作码域、条件码域、条件码设置域、目标操作数、第一操作数和**第二操作数**组成。



1.#immed_8r（常数表达式）**遵循的原则**：该常数必须能对应 8 位位图，即一个 8 位的常数通过循环右移偶数位得到，而不是一个任意数。（12 位）

原因：指令编码时，第二操作数要用 12 位来表示 32 位数，因此采用 8 位存放数据，4 位存放位移量的方式进行编码。
2. 寄存器方式下，即为寄存器的数值。
3. 寄存器移位方式，将寄存器移位的结果作为操作数，Rm 的值保持不变。

4.2 常用指令的理解

存储器访问指令：LDR、STR（特别是基址寻址）
AND 逻辑运算指令

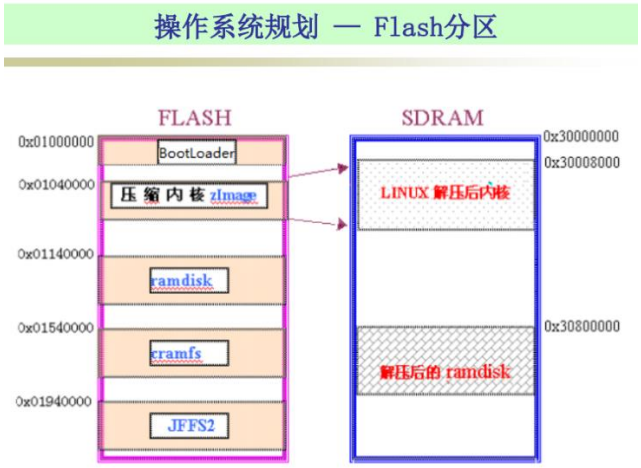
4.3 简单的 ARM 汇编程序的编写

ARM 汇编程序基本格式（框架）
数据传送、处理指令：MOV、ADD、SUB；分支转移指令：B、BL；条件码的使用：NE、HI、LS；带 S 指令；注释、标号的使用；杂项指示符：AREA、ENTRY、END

➢简单的 ARM 程序

```
;文件名：TEST1.S
;功能：实现两个寄存器相加
;说明：使用ARMulate软件仿真调试
        AREA    Example1,CODE,READONLY   ;声明代码段Example1
        ENTRY                            ;标识程序入口
        CODE32                           ;声明32位ARM指令
START   MOV     R0,#0                    ;设置参数
        MOV     R1,#10
LOOP    BL      ADD_SUB                  ;调用子程序ADD_SUB
        B       LOOP                     ;跳转到LOOP
ADD_SUB
        ADDS    R0,R0,R1                 ;R0 = R0 + R1
        MOV     PC,LR                    ;子程序返回
        END                              ;文件结束
```

5.1 Linux 操作系统规划，Flash 分区（NAND、NOR……）BootloaderCaploader？



Uboot 分区、kernel 分区、rootfs 分区，后者需要前者提供功能支持，前者先启动，后者再启动。
Bootloader 在运行过程中虽具有初始化系统和执行用户输入的命令等作用，位

于 Flash 最前端。但它最根本的功能就是为了启动 Linux 内核
RootFileSystem 是 Linux 系统的核心组成部分，它可以作为 Linux 系统中文件和数据的存储区域，通常它还包括系统配置文件和运行应用软件所需要的库。

5.2 嵌入式系统在启动时，引导代码、操作系统的运行和应用程序的加载的 2 种架构（与 Flash 有关，怎么用 Flash，结合 5.1 的分区）

➢嵌入式系统在启动时，引导代码、操作系统的运行和应用程序的加载主要有两种架构，**一种是直接从 Nor Flash 启动的架构，另一种是直接从 Nand Flash 启动的架构**。

•从 Nor Flash 启动

Nor Flash 具有芯片内执行（XIP eXecute In Place）的特点，在嵌入式系统中常做为存放启动代码的首选。从 Nor Flash 启动的架构又可细分为只使用 Nor Flash 的启动架构和 Nor Flash 与 Nand Flash 配合使用的启动架构。

单独使用 Nor Flash：在该架构中，引导代码、操作系统和应用代码共存于同一块 Nor Flash 中。系统上电后，引导代码首先在 Nor Flash 中执行，然后把操作系统和应用代码加载到速度更高的 SDRAM 中运行。另一种可行的架构是，在 Nor Flash 中执行引导代码和操作系统，而只将应用代码加载到 SDRAM 中执行。

•从 Nand Flash 启动

S3C2410 支持从 Nand Flash 启动的模式，它的工作原理是将 NandFlash 中存储的前 4KB 代码装入一个称为 Steppingstone（BootSRAM）的地址中，然后开始执行该段引导代码，从而完成对操作系统和应用程序的加载。

•Nor Flash 和 Nand Flash 配合使用

Nor Flash（2M 或 4M）中存放启动代码和操作系统，而 Nand Flash 中存放应用代码，根据存放的应用代码量的大小可以对 Nand Flash 容量做出相应的改变。系统上电后，引导代码直接在 Nor Flash 中执行，把 Nand Flash 中的操作系统和应用代码加载到速度更高的 SDRAM 中执行。也可以在 Nor Flash 中执行引导代码和操作系统，而只将 Nand Flash 中的应用代码加载到 SDRAM 中执行。该架构是当前嵌入式系统中运用广泛的启动架构之一。

6.1 Linux 操作系统中，设备分为哪几类？（3 类 字符设备 块设备 ？）

➢3 类：字符设备、块设备、网络设备

- 字符设备
  - 以字节为单位逐个进行I/O操作
  - 字符设备中的缓存是可有可无
  - 不支持随机访问
  - 如串口设备
- 块设备
  - 块设备的存取是通过buffer、cache来进行
  - 可以进行随机访问
  - 例如IDE硬盘设备
  - 可以支持可安装文件系统
- 网络设备
  - 通过BSD套接口访问

6.2 驱动程序的功能，与应用程序的关系

➢**与应用程序的关系**：从应用程序看，驱动程序应为应用程序提供访问硬件设备的编程接口，主要提供以下功能：

应用程序通过驱动程序安全有效地访问硬件；

驱动程序隐藏底层细节，从而提高应用软件的可移植性和可复用性；

驱动程序文件节点可方便地提供访问权限控制。

从驱动开发人员看，**驱动程序是直接操控硬件的软件**，主要完成以下功能：

初始化和释放设备；

直接读写硬件寄存器来控制硬件；

实现内核与硬件之间的数据交换；

操作设备缓冲区；

操作输入、输出设备，如键盘、打印机等；

实现应用程序与设备之间的数据交换；

检测和处理设备出现的错误。

6.3 Linux 内核主要组成

➢Linux 内核主要由五大块组成：**进程调度、内存管理、虚拟文件系统、网络接口、进程间通信。**

1、内存管理：主要完成的是如何合理有效地管理整个系统的物理内存，同时快速响应内核各个子系统对内存分配的请求。Linux 内存管理支持虚拟内存，而多余出的这部分内存就是通过磁盘申请得到的，平时系统只把当前运行的程序块保留在内存中，其他程序块则保留在磁盘中。在内存紧缺时，内存管理负责在磁盘和内存间交换程序块。

2、进程管理：主要控制系统进程对 CPU 的访问。当需要某个进程运行时，由进程调度器根据基于优先级的调度算法启动新的进程。：Linux 支持多任务运行，那么在一个单 CPU 上支持多任务就是由进程调度管理来实现的。

3、进程间通信：主要用于控制不同进程之间在用户空间的同步、数据共享和交换。由于不用的用户进程拥有不同的进程空间，因此进程间的通信要借助于内核的中转来实现。

一般情况下，当一个进程等待硬件操作完成时，会被挂起。当硬件操作完成，进程被恢复执行，而协调这个过程的就是进程间的通信机制。

4、虚拟文件系统：用一个通用的文件模型表示了各种不同的文件系统，这个文件模型屏蔽了很多具体文件系统的差异，使 Linux 内核支持很多不同的文件系统。这个文件系统可以分为逻辑文件系统和设备驱动程序：逻辑文件系统指 Linux 所支持的文件系统，设备驱动程序指为每一种硬件控制器所编写的设备驱动程序模块。

5、网络接口

网络接口提供了对各种网络标准的实现和各种网络硬件的支持。网络接口一般分为网络协议和网络驱动程序。网络协议部分负责实现每一种可能的网络传输协议。网络设备驱动程序则主要负责与硬件设备进行通信，每一种可能的网络硬件设备都有相应的设备驱动程序。

6.4 Linux 作为嵌入式操作系统的优势

➢Linux 做嵌入式的**优势**：
首先，Linux 是**开放源代码的**，不存在黑箱技术，遍布全球的众多 Linux 爱好者又是 Linux 开发者的强大技术支持；
其次，Linux 的**内核小、效率高，内核的更新速度很快**,Linux 是**可以定制**的，其系统内核最小只有约 134KB。
第三，Linux 是免费的 OS，**在价格上极具竞争力**。 Linux 还有着嵌入式操作系统所需要的很多特色，突出的就是 Linux 适应于多种 CPU 和多种硬件平台，是一个**跨平台**的系统。

6.5 交叉开发、交叉编译？交叉编译的两个组成部分（学名：PC 机、嵌入式操作系统；书名：？）

➢在一种计算机环境中运行的编译程序，能编译出在另外一种环境下运行的代码，我们就称这种编译器支持交叉编译。这个编译过程就叫交叉编译。简单地说，就是**在一个平台上生成另一个平台上的可执行代码**,同一个体系结构可以运行不同的操作系统，同样，同一个操作系统也可以在不同的体系结构上运行。这里需要注意的是所谓平台，实际上包含两个概念：体系结构（Architecture）、操作系统（OperatingSystem）。同一个体系结构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。
交叉开发：
首先在通用计算机上编写程序,然后通过交叉编译生成目标平台上可以运行的二进制代码格式，最后再下载到目标平台上的特定位置上运行。
交叉编译：
在宿主机平台上使用某种特定的交叉编译器,为某种与宿主机不同平台的目标系统编译程序，得到的程序在目标系统上运行而非在宿主机本地运行。
组成部分：
宿主机（Host）是一台通用计算机（如 PC 机或者工作站），它通过串口或者以太网接口与目标机通信。
目标机（Target）一般在嵌入式应用软件开发和调试期间使用，用来区别与嵌入式系统通信的宿主机。

7.1 进程与线程，区别？

➢进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。
线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。
区别：线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程；同一个进程中的多个线程之间可以并发执行。
进程与线程的关系：通常在一个进程中可以包含若干个线程，它们可以利用进程所拥有的资源。在引入线程的操作系统中，通常都是把进程作为分配资源的基本单位，而把线程作为独立运行和独立调度的基本单位。由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高

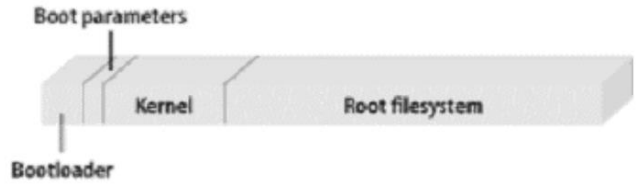
系统内多个程序间并发执行的程度。

7.2 多线程与忙等待

➢线程是程序中一个单一的顺序控制流程。在单个程序中同时运行多个线程完成不同的工作，称为多线程。
忙等待：处理线程同步时，需要等待某种资源，线程可以在一个循环中不断检查，直到条件满足退出循环。可以采用阻塞等待的方式进行处理，这样进程在等待被唤醒的过程中不会占用 CPU 内存。

7.3 Bootloader 及其工作的两个阶段

➢简单地说，**Boot Loader 就是在操作系统内核运行之前运行的一段小程序**。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。
不同的目标板需要不同的 BootLoader 支持。内存中的位置：



•两个阶段：
依赖于 CPU 体系结构的代码，比如设备初始化代码等，通常都放在阶段 1 中，而且通常都用汇编语言来实现，以达到短小精悍的目的。而阶段 2 则通常用 C 语言来实现，这样可以实现一些复杂的功能，而且代码会具有更好的可读性和可移植性。
Boot Loader 的阶段 1 通常包括以下步骤：
①硬件设备初始化。为加载 Boot Loader 的阶段 2 准备 RAM 空间。
②拷贝 Boot Loader 的阶段 2 到 RAM 空间中。
③设置好堆栈。跳转到阶段 2 的 C 入口点。
Boot Loader 的阶段 2 通常包括以下步骤：
①初始化本阶段要使用到的硬件设备，串口。
②检测系统内存映射(memory map)。
③将 kernel 映像和根文件系统映像从 Flash 读到 RAM 空间中。
④为内核设置启动参数。
⑤调用内核。

7.4 Linux 操作系统，设备分为哪几类？（与 6.1 同）

实验（重点）

汇编程序：4-5 行代码 框架 （不要乱写 会扣分）
驱动程序实验：调懂一个程序 组件构成 与驱动程序作用接结合
线程实验：线程创建、同步的概念与方法（互斥量、信号量）线程 A 卷进程 B 卷