厦门大学《嵌入式系统》课程期末试卷

软件学院　　软件工程系　　2017 级　　软件工程专业

主考教师：曾文华 试卷类型：（A 卷） 考试时间：2020.1.7

一、填空题（30 个空，每 1 空 1 分，共 40 分；在答题纸填写答案时请写上每个空格的对应编号）

1、最新的 ARM 处理器产品是 ARM Cortex-A 系列、ARM Cortex-R 系列、ARM Cortex-M 系列，其中 Cortex-A 系列又称为 （1）高性能 处理器，Cortex-R 系列又称为 （2）实时 处理器 ，ARM Cortex-M 系列又称为 （3）低成本、低功耗 处理器。

2、实验箱的主 CPU i.MX6 是基于 （4）ARM Cortex-A9 架构的嵌入式处理器；实验箱的从 CPU STM32 其内核是 （5）ARM Cortex-M3 。

3、最受欢迎的 10 个 Linux 发行版：（6）Ubuntu 、（7）Fedora 、OpenSUSE、Debian、Mandriva、Mint、PCLinuxOS、Slackware、Gentoo、CentOS。

4、ARM 处理器的异常模式是指除 （8）用户 模式和 （9）系统 模式外的其他五种模式。

5、ARM 指令有两种状态，分别是 ARM 状态和 Thumb 状态。ARM 状态和 Thumb 状态切换可以通过 （10）BX 指令来实现。

6、RT-Linux 是具有 （11）硬实时 特性的多任务操作系统。

7、Linux 是 （12）单内核 的，（12）单内核 最大的优点是效率高，因为所有的内容都集中在一起，（12）单内核 也有可扩展性以及可维护性差的缺点，（13）模块机制 的引入就是为了弥补这一缺点。

8、μCLinux 是专门针对没有 （14）MMU（存储管理单元） 的处理器设计的。

9、Linux 内核支持动态可加载模块，模块通常是 （15）设备驱动程序 。可以通过 insmod 命令加载模块，通过 rmmod 命令卸载模块，通过 lsmod 命令列出已经安装的模块。

10、嵌入式应用软件开发通常采用交叉开发模式，也称为 （16）宿主机/目标机 模式。

11、IMX6 实验箱（嵌入式 Linux 系统）启动后（即打开实验箱的电源开关，或者按下实验箱的 Reset 键），

先执行（17）Boot Loader，进行硬件和内存的初始化工作，然后加载（18）Linux 内核和（19）根文件系统，完成 Linux 系统的启动。

12、Linux 的设备驱动程序开发调试有两种方法，第一种是直接编译到（20）内核；第二种是编译为（21）模块的形式，单独加载运行调试。

13、块设备驱动程序没有 read 和 write 操作函数，对块设备的读写是通过（22）请求函数完成的。

14、YAFFS（Yet Another Flash File System）是专为嵌入式系统使用（23）NAND 型闪存而设计的一种日志型文件系统。

15、使用 mmap 系统调用，可以将（24）内核空间的地址映射到（25）用户空间。

16、CAN 总线信号使用差分电压传送，两条信号线被称为 CAN_H 和 CAN_L。CAN_H 和 CAN_L 均是 2.5V 左右时，表示为逻辑（26）1，称为“隐性”。CAN_H=3.5V、CAN_L=1.5V 时，表示逻辑（27）0，称为“显性”。

17、Android 的软件架构采用了分层结构，由上至下分别为：Application 应用层、Application Framework 应用框架层、Android Runtime & Libraries 运行时库和本地库层、（28）Linux Kernel 内核层。

18、Android 应用程序开发是基于 Android 架构提供的 API 和类库编写程序，这些应用程序是完全的（29）Java 代码程序，它们构建在 Android 系统提供的 API 之上。

19、开发 Android 应用程序可以基于 Google 提供的（30）Android SDK 开发工具包，也可以 直接在 Android 源码中进行编写。

二、名词解释（请写出下列英文缩写的中文全称，10 小题，每 1 小题 1 分，共 10 分；在答题纸填写答案时请写上每小题的对应编号）

1、AVD：Android Virtual Device，Android 虚拟设备

2、BOOTP：Bootstrap Protocol，引导程序协议

3、IP 核：Intellectual Property，知识产权核，知识产权模块

4、I2C（IIC，I^2C）：Inter Integrated-Circuit，内部集成电路总线

5、JTAG：Joint Test Action Group，联合测试工作组，JTAG 协议

6、MDK：Microcontroller Development Kit

7、NDK：Native Development Kit

8、Rootfs：Root File System，根文件系统

9、SPI：Serial Peripheral Interface，串行外设接口

10、SoC：System on Chip，片上系统

三、简答题（8 小题，共 25 分；在答题纸填写答案时请写上每小题的对应编号）

1、什么是嵌入式系统的交叉开发（交叉编译）？（2 分）

答：即宿主机/目标机模式。宿主机为 PC 机；目标机可以是实际的运行环境，也可以用仿真系统替代实际的运行环境。

2、Ubuntu 的“NFS 服务”的功能是什么？“Samba 服务”的功能是什么？（2 分）

答：Samba 服务：在 Windows 下，可以访问 Ubuntu 的文件夹（如：/home/now）。

NFS 服务：将 Ubuntu 的文件夹（如：/imx6），设为 NFS 共享，在实验箱上执行 mount 命令，即可将 Ubuntu 的文件夹（/imx6），共享到实验箱上的文件夹上（如：/mnt）。

3、ARM 指令格式如下：

<opcode> {<cond>} {S} <Rd>, <Rn> {, <shift_op2>}

< >内的项是必须的，{ }内的项是可选的

请写出 ARM 指令格式中各个字段的含义。（3 分）

答：

opcode：指令助记符（操作码），如 LDR，STR 等

cond：执行条件（条件码），如 EQ，NE 等

S：可选后缀，加 S 时影响 CPSR 中的条件码标志位，不加 S 时则不影响

Rd：目标寄存器

Rn：第 1 个源操作数的寄存器

op2：第 2 个源操作数

shift：位移操作

4、ARM 处理器的运行模式有哪 7 种？（3 分）

答：

1) 用户模式（USR）
2) 快速中断模式（FIQ）
3) 外部中断模式（IRQ）
4) 管理模式（SVC）
5) 数据访问终止模式（ABT）
6) 系统模式（SYS）
7) 未定义指令终止模式（UND，未定义模式）

5、什么是 Boot Loader？其作用是什么？常见的 Boot Loader 有那几个？（3 分）

答：

（1）Bootloader：引导加载程序。

（2）嵌入式系统（实验箱）启动后（打开电源，或者按 Reset 键），先执行 Bootloader，进行硬件和内存的初始化工作，然后加载 Linux 内核和根文件系统，完成 Linux 系统的启动。

（3）常见的 Boot Loader 有：U-Boot、vivi、Blob。

6、IMX6 实验箱的固态存储器（Flash 存储器）的典型空间分配结构是什么？（2 分）

答：

1) Boot Loader
2) 内核的启动参数（Boot parameters）
3) 内核映像
4) 根文件系统映像

7、什么是设备文件？（2 分）

答：Linux 抽象了对硬件的处理，所有的硬件设备都可以作为普通文件一样对待，可以使用标准的系统调用接口来完成对设备的打开（open）、关闭（close）、读写（read、write）和 I/O 控制操作（ioctl），驱动程序的主要任务是实现这些系统调用函数。

8、请解释以下命令中每个字段（共 5 个字段）的具体含义：（2 分）

mknod /dev/lp0 c 6 0

答：

mknod：创建设备文件的命令

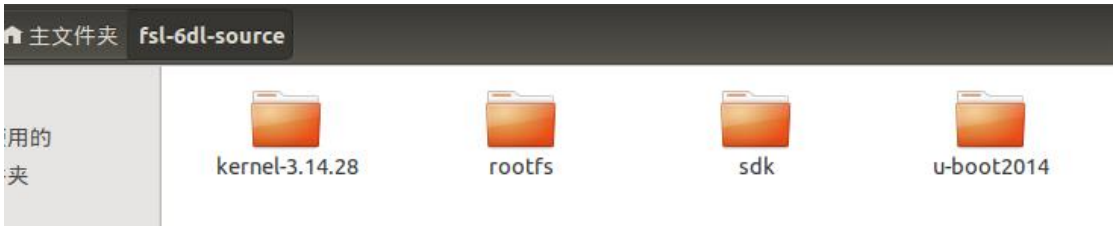
/dev/lp0：设备名

c：表示字符设备

6：主设备号

0：次设备号

9、IMX6 实验箱 Linux 环境（Ubuntu 环境）的 fsl-6dl-source.tar.gz 压缩文件，解压后得到如下的 4 个文件夹，请问这 4 个文件夹分别是存放什么内容？（2 分）



答：

① kernel-3.14.28：内核源代码目录

② rootfs：文件系统目录

③ sdk：交叉编译器目录

④ u-boot2014：uboot 源代码目录

10、简述 Android NDK 开发过程，包括 Android NDK 开发环境的搭建、HelloJni 程序的编译和运行过程。（3 分）

答：

第一步搭建 NDK 开发环境：

（1）在 Vmware 虚拟机中，打开 Android NDK 系统用的 Ubuntu

（2）下载 Android NDK，得到源码包：android-ndk-r9-linux-x86.tar.bz2，并将其拷贝到 Ubuntu 中

（3）解压 NDK 源码，在虚拟机 Ubuntu 环境下执行：# tar xjvf android-ndk-r9-linux-x86.tar.bz2 -C /Android/

（4）配置环境变量

第二步：NDK 开发与编译。执行$NDK，编译 C/C++代码，生成 libhello-jni.so

第三步：将 libhello-jni.so 文件拷贝到电脑硬盘的 HelloJni 的..\app\libs\armeabi\目录中

第四步：打开 Android Studio，打开 HelloJni 工程，编译 HelloJni 工程

第五步：在 Android Studio 中，选择在实验箱上运行 HelloJni 工程

11、简述实验箱从 CPU（STM32）程序的开发过程，包括 MDK 的安装、J-Link 的安装，以及 LED 实验程序的编译和运行。（3 分）

答：

第一步：安装 MDK5

（1）安装 MDK5 主体。运行“MDK520.EXE”文件

（2）安装对应 MCU 的 Pack 支持包。运行“Keil.STM32F1xx_DFP.2.1.0.pack”文件

（3）注册破解 MDK520。运行“注册.exe”文件（需要关闭电脑的杀病毒软件）

第二步：安装调试工具 J-Link

（1）将 J-Link 的扁线插入实验箱的 STM32 JTAG 插槽，J-Link 的 USB 口连到电脑的 USB 口

（2）运行“Setup_JLink_V512h.exe”

第三步：LED 实验程序的编译和运行

（1）运行“Keil uVision5”（MDK5）。

打开 LED 工程（D:\UP-Tech\STM32\Exp\LED\Project\MDK\LED.uvprojx）

（2）编译 LED 工程

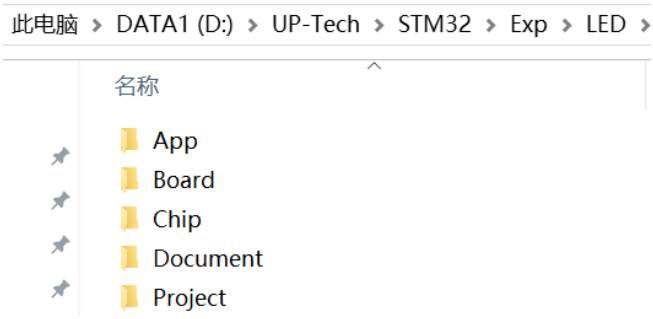
（3）下载目标代码（可执行程序的十六进制文件）到实验箱的 Flash 中（STM32 核心板上）

方法一：通过 MDK5 下载目标代码

方法二：通过 J-Link 下载目标代码

（4）按实验箱的 STM32 核心板旁边的 Reset 键，则开始执行程序

12、实验箱从 CPU（STM32）LED 程序的 MDK 工程文件夹如下：



请问该文件夹 5 个子文件夹存放什么内容？（3 分）

答：（1）App 存放用户程序

（2）Board 存放开发板（实验箱）的驱动程序

（3）Chip 存放 STM32 芯片驱动程序

（4）Document 存放工程说明文档

（5）Project 存放工程文件

四、综合题（7 小题，共 25 分；在答题纸填写答案时请写上每小题的对应编号）

1、设当前目录下有 pthread.c 文件和 Makefile 文件，Makefile 文件的内容如下：

```
CC = arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/1.7/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi
EXTRA_LIBS += -lpthread
EXP_INSTALL = install -m 755
INSTALL_DIR = ../bin
EXEC = ./pthread
OBJS = pthread.o
```

```
all: $(EXEC)
$(EXEC): $(OBJS)
	$(CC) -o $@ $(OBJS)  $(EXTRA_LIBS)
install:
	$(EXP_INSTALL) $(EXEC) $(INSTALL_DIR)
clean:
	-rm -f $(EXEC) *.elf *.gdb *.o
```

已知“/opt/poky/1.7/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi”文件中有以下的内容：

```
export CFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types"
```

请问在当前目录下执行 make 命令，其结果是什么（屏幕上显示什么内容）？执行 make install 命令，其结果是什么（屏幕上显示什么内容）？执行 make clean 命令，其结果是什么（屏幕上显示什么内容）？

该 Makefile 文件将完成交叉编译工作（即编译在实验箱上运行的可执行文件）。如果要完成本地编译工作（即编译在虚拟机上执行的可执行文件），请问怎么修改 Makefile 文件（只需写出修改的地方）？（3 分）

注：同学们在答题时，可以用 CC 代替 arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/1.7/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi，用 CFLAGS 代替 -O2 -pipe -g -feliminate-unused-debug-types

答：

（1）执行 make 命令，显示：

arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/1.7/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi -O2 -pipe -g -feliminate-unused-debug-types -c -o pthread.o pthread.c

arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/1.7/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi -o pthread pthread.o -lpthread

或者：

CC CFLAGS -c -o pthread.o pthread.c

CC -o pthread pthread.o -lpthread

（2）执行 make install 命令，显示：

install -m 755 ./pthread ../bin

（3）执行 make clean 命令，显示：

rm -f ./pthread *.elf *.gdb *.o

（4）将

CC = arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/1.7/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi

修改为：

CC = gcc

2、以下程序为汇编语言调用 C 语言的例子：

```
    （1）          @声明要调用的 C 函数
    MOV r0, 1
    MOV r1,2       @通过 r0、r1 传递参数（参数传递规则）
    （2）          @调用 C 函数 add；返回结果由 r0 带回（子程序返回结果规则）

    int   add (int x, int y)
    {
        return(x+y);
    }
```

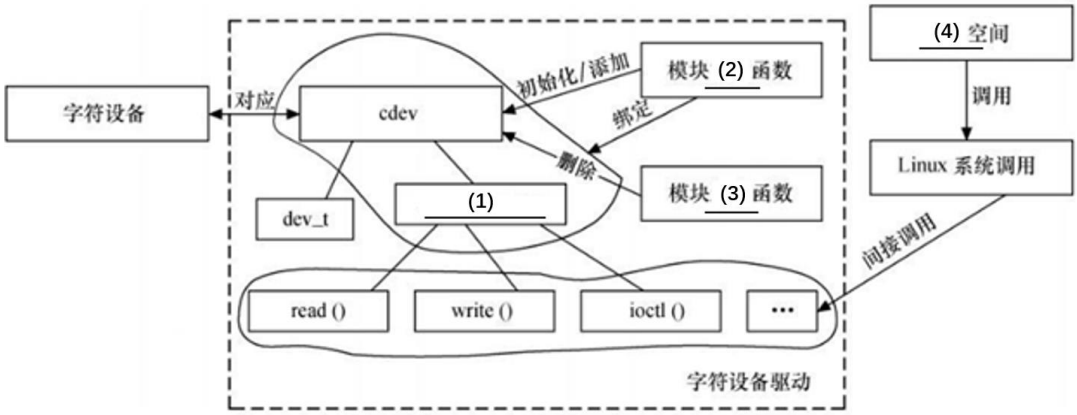
请填写程序中空白（划线）的那二行。（2 分）

答：

（1）IMPORT add

（2）BL add

3、字符设备驱动框架如下：



请填写该框架中的 4 个空格（(1)至(4)）部分的内容。（4 分）

答：

（1）file_operations

（2）加载

（3）卸载

（4）用户

4、以下是 RS-485 驱动程序的头文件和全局变量，请问该程序的第 16）、17）、18）行分别是做什么事情？（3 分）

```
1)    #include <linux/kernel.h>
2)    #include <linux/module.h>
3)    #include <linux/init.h>
4)    #include <linux/errno.h>
5)    #include <linux/fs.h>
6)    #include <linux/cdev.h>
7)    #include <linux/types.h>
8)    #include <linux/device.h>
9)    #include <asm/system.h>
10)   #include <asm/uaccess.h>
11)   #include <linux/platform_device.h>
12)   #include <asm/irq.h>
13)   #include <linux/of.h>
```

```
14)     #include <linux/of_device.h>
15)     #include <linux/of_gpio.h>
16)     #define DRVNAME "UART485"
17)     #define UART485_MAJOR 30
18)     #define UART485_MINOR 0
19)     #define UART485_TX 1
20)     #define UART485_RX 0
```

答：

第 16）行：定义设备名

第 17）行：定义主设备号

第 18）行：定义次设备号

5、以下是 RS-485 驱动程序的初始化和退出函数，请问该程序的第 6）、33）行分别是做什么事情？（2 分）

```
1)      static int Uart485Init(void)
2)      {
3)          dev_t devt;
4)          int retval;
5)          devt = MKDEV(UART485_MAJOR,UART485_MINOR);
6)          retval = register_chrdev_region(devt,1,DRVNAME);
7)          if(retval>0)
8)              return retval;
9)          cdev_init(&uart485cdev,&uart485_fops);
10)         retval = cdev_add(&uart485cdev,devt,1);
11)         if(retval)
12)             goto error;
13)         uart485_class = class_create(THIS_MODULE,"UART485");
14)         if (IS_ERR(uart485_class))
15)         {
16)             printk(KERN_ERR "Error creating raw class.\n");
17)             cdev_del(&uart485cdev);
18)             goto error;
19)          }
20)         device_create(uart485_class,  NULL,  MKDEV(UART485_MAJOR,UART485_MINOR),
```

```
        NULL,DRVNAME);
21)         gpio_request(gpio_ctrl,"uart485Ctrl");
22)         gpio_direction_output(gpio_ctrl,0);
23)         gpio_free(gpio_ctrl);
24)         return 0;
25)         error:
              unregister_chrdev_region(devt, 1);
26)         return retval;
27)     }
28)     static void Uart485Exit(void)
29)     {
30)          device_destroy(uart485_class,MKDEV(UART485_MAJOR,UART485_MINOR));
31)          class_destroy(uart485_class);
32)          cdev_del(&uart485cdev);
33)          unregister_chrdev_region(MKDEV(UART485_MAJOR,UART485_MINOR), 1);
34)     }
```

答：

第 6）行：注册字符设备

第 33）行：注销字符设备

6、以下是 RS-485 驱动程序的模块初始化和模块退出函数，请填写程序中的 2 个空格部分的内容：（2 分）

```
static int __init gpio_uart485_init(void)
{
    printk("\n\n\nkzkuan___%s\n\n\n",__func__);
    return platform_driver_register(&gpio_uart485_device_driver);
}

static void __exit gpio_uart485_exit(void)
{
    printk("\n\n\nkzkuan___%s\n\n\n",__func__);
    platform_driver_unregister(&gpio_uart485_device_driver);
}
```

_____（1）_____(gpio_uart485_init);

_____（2）_____(gpio_uart485_exit);

答：
（1）module_init
（2）module_exit

7、如果我们不采用挂载的方式，而是采用下载的方式运行程序，即将 Ubuntu 中的可执行文件下载到实验箱中，再运行程序，请写出操作步骤（包括下载程序、运行程序）。设可执行文件（hello）存放在 Ubuntu 的/imx6/whzeng/hello/目录下，tftpd32.exe 文件（TFTP 服务）在 Windows 的 D:\UP-Tech\Linux 目录下，需要将可执行文件（hello）下载到实验箱的/home/root 目录中，Windows 系统的 IP 地址为 59.77.5.121。（4 分）

答：
第一步：将 Ubuntu 下的/imx6/whzeng/hello/hello 文件，复制到 Windows 的 D:\UP-Tech\Linux 目录下（使用 Samba 服务）

第二步：在 Windows 下运行“tftpd32.exe”（TFTP 服务），将 tftpd32 的 Current Directory 设为 D:\UP-Tech\Linux，Server interface 设为 59.77.5.121。

第三步：在实验箱的“超级终端（Xshell 2.0）”下，执行：

```
cd /home/root
tftp -gr hello 59.77.5.121
```

第四步：在实验箱的“超级终端（Xshell 2.0）”下，执行：

```
chmod 777 hello
./hello
```

8、以下为 RS-485 双机通讯程序的一部分，请问该程序中的第 7）、8）、9）、14）行分别是做什么事情？（4 分）

```
1)  void* receive(void * data)
2)  {
3)      int c;
4)      printf("RS-485 Receive Begin!\n");
5)      for(;;)
6)      {
7)          ioctl(fd485, UART485_RX);
8)          read(fdCOMS1,&c,1);
9)          write(1,&c,1);
10)         if(c == 0x0d)
```

```
11)                 printf("\n");
12)             if(c == ENDMINITERM)
13)                 break;
14)             ioctl(fd485, UART485_TX);
15)     }
16)     printf("RS-485 Receive End!\n");
17)     return NULL;
18) }
```

答：

第 7）行：设置 RS-485 为接收模式

第 8）行：从 RS-485 中读 1 个字符

第 9）行：将读到的字符在标准输出设备（显示器）上输出（显示）

第 14）行：设置 RS-485 为发送模式

9、以下为按键（小键盘）的主程序，请说明程序中的第 22）、23）、24）行的具体功能是什么？ （3 分）

```
1)  #include <stdio.h>
2)  #include <sys/types.h>
3)  #include <sys/stat.h>
4)  #include <fcntl.h>
5)  #include <linux/input.h>
6)  #define NOKEY 0
7)  int main(int argc,char *argv[])
8)  {
9)      int keys_fd;
10)     char ret[2];
11)     struct input_event t;
12)     keys_fd = open(argv[1], O_RDONLY);
13)     if(keys_fd<=0)
14)     {
15)         printf("open %s device error!\n",argv[1]);
16)         return 0;
17)     }
18)     while(1)
19)     {
20)         if(read(keys_fd,  &t,  sizeof(t)) == sizeof(t))
21)         {
```

```
22)                 if(t.type == EV_KEY)
23)                     if(t.value == 0 || t.value == 1)
24)                         printf("key %d %s\n",t.code,(t.value)?"Pressed":"Released");
25)             }
26)         }
27)         close(keys_fd);
28)         return 0;
29)     }
```

答：

第 22）行：判断输入设备是不是键盘？

第 23）行：判断有没有键按下（0 表示按下），或者有没有键释放（1 表示释放）？

第 24）行：显示按下或释放的键的代码

10、以下为小键盘控制电子钟的主程序中的关键代码，请说明程序中的第 5）、12）、13）行的具体功能是什么？ （3 分）

```
1)    int main(int argc, char *argv[])
2)    {
3)        keys_fd = open(KEYDevice, O_RDONLY);
4)        mem_fd = open("/dev/mem", O_RDWR);
5)        cpld = (unsigned char*)mmap(NULL,(size_t)0x10,PROT_READ | PROT_WRITE |
      PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
6)        pthread_create(&th_time, NULL, time_counter, 0);
7)        pthread_create(&th_key, NULL, key_input, 0);
8)        while(1)
9)        {
10)           for(i=0; i<8; i++)
11)           {
12)               *(cpld+(0xe6<<1)) = addr[i];
13)               *(cpld+(0xe4<<1)) = tube[number];
14)               usleep(1000);
15)           }
16)       }
17)       pthread_join(th_time, &retval);
18)       pthread_join(th_key, &retval);
19)       munmap(cpld,0x10);
```

```
20)         close(mem_fd);
21)         close(keys_fd);
22)         return 0;
23)   }
```

答：

第 5）行：内存映射操作，将数码管的内容映射到用户空间

第 12）行：设置数码管的位地址（即哪一个数码管）

第 13）行：设置数码管的段值（即显示什么内容，七段码或八段码）