厦门大学《嵌入式系统》课程期末试卷

信息学院　　软件工程系　　2021 级　　软件工程专业

主考教师：曾文华　试卷类型：（A 卷）答案　考试时间：2024.1.10

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

1. IMX6 嵌入式教学科研平台（实验箱）有二个 CPU，一个是主 CPU，采用飞思卡尔公司生产的基于 ARM　Cortex-A9（1）　的最新单核的 IMX6DL 嵌入式微处理器（Freescale IMX6DL）；另一个是从 CPU，采用 ARM　Cortex-M3（2）　内核架构的意法半导体公司生产的 STM32F103 芯片。

2. 嵌入式系统主要的存储设备为　Flash Memory（闪存）（3）　和　SDRAM（4）　。

3. ARM 处理器有两种状态，分别是　ARM（5）　状态和　Thumb（6）　状态。

4. Ubuntu 是 Linux 系统最受欢迎的的　发行版（7）　。

5. μCLinux 中的 μ 表示　Micro（微）（8）　，C 表示　Control（控制）（9）　，μCLinux 是专门针对没有　MMU（存储管理单元）（10）　的处理器设计的。

6. RT-Linux 是具有　硬实时（11）　特性的多任务操作系统；RT-Linux 通过在 Linux 内核与硬件中断之间增加一个精巧的可抢先的　实时内核（12）　，把标准的 Linux 内核作为　实时内核（12）　的一个进程与用户进程一起调度。

7. 嵌入式 Linux 系统启动后，先执行　Bootloader（13）　，进行硬件和内存的初始化工作，然后加载 Linux 内核（14）　和　根文件系统映像（15）　，完成 Linux 系统的启动。

8. Linux 的设备驱动程序开发调试有两种方法，一种是直接编译到　内核　（16）　，另一种是编译为　模块（17）　的形式；第一种方法效率较　低（18）　，第二种方式效率较　高（19）　。

9. 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）（20）　，驱动程序的主要任务是实现这些系统调用函数。

10. Qt 是一个由 Qt Company 开发的跨平台 C++　图形用户界面　（21）　应用程序开发框架。

11. Boot Loader 的操作模式有两种，分别是 启动加载模式（22） 和 下载模式（23） 。

12. 在 Ubuntu 上执行 make 命令（交叉编译生成可执行文件）前，需要先执行“source /opt/poky/1.7/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi”命令，该命令的作用是 指定交叉编译器的路径（24） 。

13. 使用 mmap 系统调用（mmap()函数），可以将 内核 （25） 空间的地址映射到 用户 （26） 空间。

14. 块设备没有 read 和 write 操作函数，对块设备的读写是通过 请求 （27） 函数完成的。

15. 对网络设备的访问必须使用 套接字（Socket）（28） ，而非读写设备文件。

16. Android 系统的底层是由 Linux（29） 操作系统作为内核。

17. Atlas 200 DK 是华为公司生产的面向 AI 应用的开发者套件，其核心是 Ascend 310 AI （30） 处理器。

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. Android NDK：Android Native Development Kit
2. CAN：全称为 Controller Area Network，即控制器局域网
3. CPSR：Current Program Status Register，当前程序状态寄存器
4. GPIO：General Purpose Input/Output，通用输入输出
5. I2C：（Inter Integrated-Circuit，IIC，I2C，内部集成电路）总线
6. JTAG：Joint Test Action Group，联合测试工作组
7. JFFS3：Journalling Flash File System Version3，闪存日志型文件系统第 3 版
8. NFC：近场通信（Near Field Communication）
9. Ramfs：基于 RAM 的文件系统。
10. SPI：是串行外设接口（Serial Peripheral Interface）

三、简答题（7 小题，共 20 分；在答题纸填写答案时请写上每小题的对应编号）

1. （2 分）常见的嵌入式操作系统有哪些？

答：

嵌入式 Linux

VxWorks

μC/OS-II

Windows CE

Sysbian

Android
iOS
其它：QNX，Palm OS，LynxOS，NucleusPLUS，ThreadX，eCos

2. （3 分）ARM Cortex-A、ARM Cortex-R、ARM Cortex-M 系列处理器分为针对什么应用场合？

答：

（1）ARM Cortex-A 系列又称“高性能处理器”（Highest Performance），它是面向移动计算如智能手机、平板电脑和服务器市场定制的高端处理器内核，支持了包括 Linux、Android、Windows 和 iOS 等系统必须的内存管理单元（MMU），而且也是与我们平时接触最为密切的存在。

（2）ARM Cortex-R 系列实时处理器（Real-Time Processing）为要求可靠性、高可用性、容错功能、可维护性和实时响应的嵌入式系统提供高性能计算解决方案。Cortex-R 系列处理器通过已经在数以亿计的产品中得到验证的成熟技术提供极快的上市速度，并利用广泛的 ARM 生态系统、全球和本地语言以及全天候的支持服务，保证快速、低风险的产品开发。

（3）Cortex-M 系列针对成本和功耗敏感（Lowest Power, Lower Cost）的 MCU 和终端应用（如智能测量、人机接口设备、汽车和工业控制系统、大型家用电器、消费性产品和医疗器械）的混合信号设备进行过优化。

3. （2 分）什么是交叉开发（交叉编译）？

答：

宿主机/目标机模式：

宿主机：PC 机（x86 环境）

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（ARM 环境）

4. （3 分）什么是 Boot Loader？其作用是什么？常见的 Boot Loader 有那几个？

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

5. （3 分）请写出 ARM 指令的格式。

答：

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

< >内的项是必须的，{ }内的项是可选的

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

6. （4 分）宿主机与目标机通常有 4 种连接方式，请结合 IMX6 实验箱分别说明每一种连接方式的具体内容和应用场景。

答：

宿主机与目标板的连接方式：

（1）串口：实验箱的 COM1 TO USB 就是串口，在宿主机（电脑）上运行“Xshell 2.0”时，就是利用该串口，实现宿主机（电脑）与目标机（实验箱）的连接。

（2）以太网接口（RJ45）：挂载方式在“Xshell 2.0”上执行“ mount -t nfs 59.77.5.101:/imx6 /mnt”时，就是利用网口，将 Ubuntu 上的文件挂载到实验箱上。

（3）USB 接口：如实验箱的 USB OTG 口就是 USB 接口，实现将 Windows 系统文件夹下的实验箱内核（操作系统）烧写到实验箱。

（4）JTAG 接口：在做 ST32 实验时，利用 JTAG 接口，将可执行文件烧写到从 CPU 的 Flash 中。

7. （3 分）简述在 IMX6 实验箱上开发 Android NDK 程序的步骤。

答：

第一步：在 Ubuntu 环境下编写 hello-jni.c 和 Android.mk 程序，并编译生成 libhello-jni.so 库文件

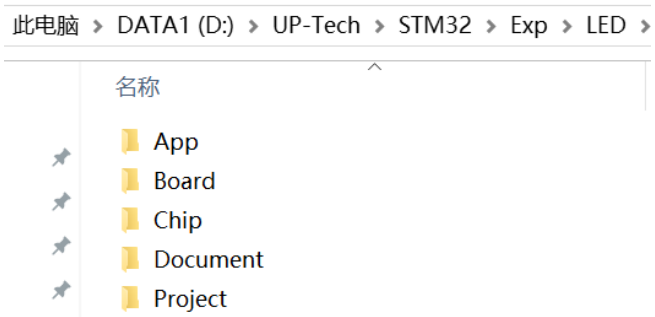
第二步：在 Android Studio 环境下编写 HelloJni 工程

第三步：将 libhello-jni.so 库文件拷贝到 Android Studio 的 HelloJni 工程中

第四步：在 Android Studio 环境下编译 HelloJni 工程，并在实验箱上运行 HelloJni 工程

四、综合题（10 小题，共 40 分；在答题纸填写答案时请写上每小题的对应编号）

1. （5 分）STM32 LED 灯实验程序的工程项目文件夹如下，请问该工程项目文件夹的 5 个子文件夹中分别存放什么内容？



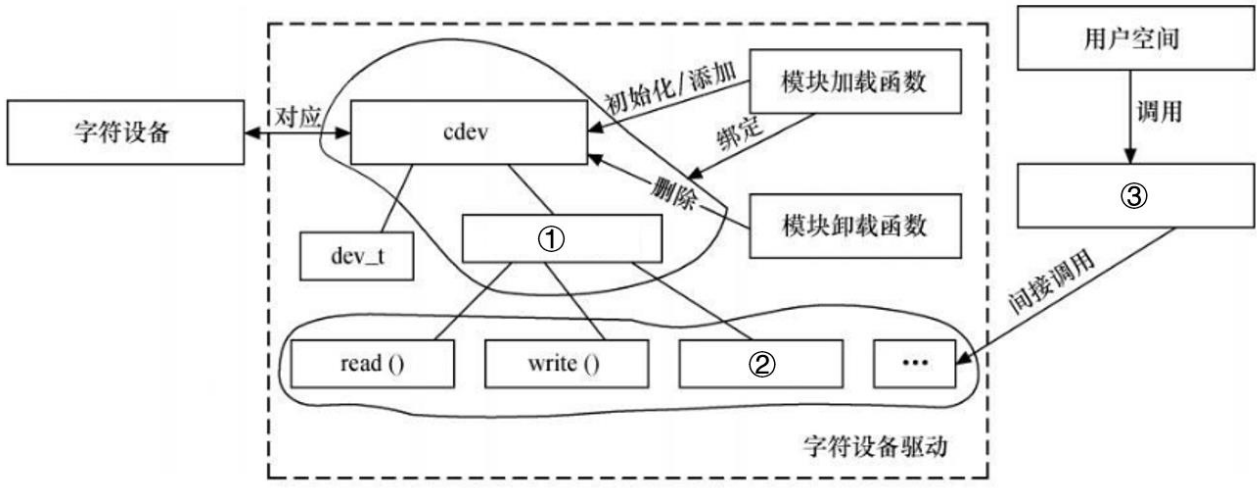
答：

App：用户程序

Board：开发板驱动程序

Chip：STM32 处理器芯片驱动程序
Document：工程说明文档
Project：工程文件

2. （3 分）下图为字符设备驱动框架，请填写图中 3 个空白方框的内容。



答：
（1）file_operations
（2）ioctl()
（3）Linux 系统调用

3. （3 分）我们在做实验时，通常采用挂载的方式，在实验箱的“超级终端（Xshell 2.0）”下，执行存放在 Ubuntu 中的可执行文件。此时运行实验箱的“超级终端（Xshell 2.0）”后，我们首先需要设置实验箱的 IP 地址，执行挂载命令，然后再运行可执行文件。设实验箱的 IP 地址为 59.77.5.120，Ubuntu 的 IP 地址为 59.77.5.122，需要将 Ubuntu 的“/imx6”目录挂载到实验箱的“/mnt”目录下，可执行文件（hello）存放在 Ubuntu 的/imx6/whzeng/hello 目录下。请写出设置实验箱的 IP 地址的命令，实现挂载功能的命令，以及运行 hello 可执行文件的命令。

答：
ifconfig eth0 59.77.5.120
mount -t nfs 59.77.5.122:/imx6 /mnt
cd /mnt/whzeng/hello
./hello

4. （5 分）已知当前目录下有 pthread.c 和 Makefile 两个文件，其中 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/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-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
```

请问在当前目录下分别执行 make、make install、make clean 命令，分别会显示什么结果？如果要将编译后的可执行文件能够在 Ubuntu 环境（x86 环境）下运行，请问如何修改 Makefile 文件？

假设：pthread.c 文件是正确的（不会出现编译错误）。在写显示结果时请用 CC1 代替 arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -O2 -pipe -g -feliminate-unused-debug-types；用 CC2 代替 arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi。

答：

（1）执行 make 命令：

```
CC1     -c -o pthread.o pthread.c
CC2 -o pthread pthread.o     -lpthread
```

（2）执行 make install 命令：

```
install -m 755    ./pthread ../bin
```

（3）执行 make clean 命令：

```
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/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi

修改为 CC = gcc

5. （5 分）以下 2 个程序为 C 语言调用汇编语言的程序，请问程序 1 和程序 2 分别属于什么调用形式（什么汇编）？并补充 2 个程序中 3 个划线处的内容。

程序 1：

```
_____ add                          @声明 add 子程序将被外部函数调用
add:
    ADD r0,r0,r1
    MOV pc,lr

_____ int add(int x, int y);       //声明 add 为外部函数
void main()
{
    int a=1, b=2, c;
    c = add(a, b);
}
```

程序 2：

```
void enable_IRQ(void)
{
    int tmp;
    _____                          //声明内联汇编代码
    {
        MRS tmp, CPSR
        BIC tmp, tmp, #0x80
        MSR CPSR_c, tmp
    }
}
```

答：

（1）嵌入式汇编

（2）内联汇编

EXPORT
Extern
__asm

6. （2 分）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。

```
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. （3 分）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）、14）行分别是做什么事情？

```
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 14）行：设置 RS-485 为发送模式

8. （5 分）以下程序为改进后的读取小键盘按键值的主程序，请问该程序中的第 5）、13）、15）、16）、18）行分别是完成什么任务？

```
1)  int main(int argc,char *argv[])
2)  {
3)          int keys_fd;
4)          struct input_event t;
5)          keys_fd = open(KEYDevice, O_RDONLY);
6)          if(keys_fd <= 0)
7)          {
8)                  printf("open key device error!\n");
9)                  return 0;
10)         }
11)         while(1)
12)         {
13)                 if(read(keys_fd,&t,sizeof(t)) == sizeof(t))
14)                 {
15)                         if(t.type == EV_KEY)
16)                                 if(t.value == 0)
17)                                 {
18)                                         printf("%c\n",key_value(t.code));
19)                                 }
20)                 }
21)         }
22)         close(keys_fd);
23)         return 0;
24) }
```

答：

5）打开输入设备文件

13）读取输入设备事件，并判断是否读取成功

15）判断是不是小键盘
16）判断小键盘的按键是不是按下
18）将小键盘的键值转换为 0-9 和*、#显示出来

9. （4 分）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)、19）行的具体功能是什么？

```
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;
}
```

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）
第 19）行：解除内存映射

10. （5 分）以下为 CAN 总线双机通信中接收程序的主函数，请什么程序中第 10)、12)、15)、18)、21）行的含义。

```
1)      int main(int argc, char *argv[])
2)      {
3)        int s, nbytes, nbytes_send;
4)        struct sockaddr_can addr;
5)        struct ifreq ifr;
6)        struct can_frame frame_rev;
7)        struct can_frame frame_send;
8)        struct can_filter rfilter[1];
9)        int len = sizeof(addr);
10)       s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
11)       strcpy(ifr.ifr_name, "can0" );
12)       ioctl(s, SIOCGIFINDEX, &ifr);
13)       addr.can_family = AF_CAN;
14)       addr.can_ifindex = ifr.ifr_ifindex;
15)       bind(s, (struct sockaddr *)&addr, sizeof(addr));
16)       rfilter[0].can_id = 0x00;
17)       rfilter[0].can_mask = CAN_SFF_MASK;
18)       setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter));
19)     while(1)
20)       {
21)             nbytes = read(s, &frame_rev, sizeof(frame_rev));
22)             if(nbytes > 0)
23)             {
24)                   printf("ID=0x%X                                                    DLC=%d
        data[0]=%X\n",frame_rev.can_id,frame_rev.can_dlc,frame_rev.data[0]);
25)             }
26)       }
27)       close(s);                                                    /
28)       return 0;
29)     }
```

答：第 10）创建套接字

第 12）行指定 can0 设备

第 15）行将套接字与 can0 绑定

第 18）行设置过滤规则，只接收表示符等于 0x00 的报文

第 21）行接收报文