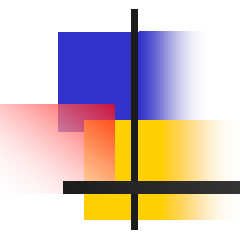
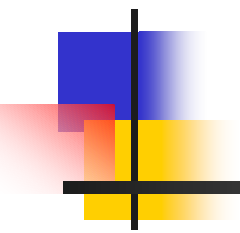


嵌入式操作系统知识

主要内容

- 1 嵌入式系统基本知识
- 2 嵌入式系统开发过程
- 3 LINUX操作系统介绍
- 4 GNU工具的使用
- 5 LIUNX下应用程序的开发



1 嵌入式系统基本知识

嵌入式系统概述

- 嵌入式系统定义
  - 嵌入式系统定义：以应用为中心，以计算机技术为基础，软硬件可裁减，从而能够适应实际应用中对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统
- 嵌入式系统组成
  - 嵌入式处理器 将通用计算机中许多板卡完成的任务集成在芯片内部，有利于小型化、高效率、高可靠性等。ARM、MIPS、PowerPC、MC68000
  - 外围设备 存储器、接口
  - 嵌入式操作系统 稳定、安全的软件模块集合，用以实现管理存储器分配、中断处理、任务间通信和定时响应，以及提供多任务处理等
  - 应用软件 基于实际专业领域，基于相应嵌入式硬件平台，并能实现预期任务的计算机软件。（有些需要嵌入式操作系统的支持 ）

嵌入式系统概述

- 嵌入式系统特点
  - 软硬件一体化，集计算机技术、微电子技术和行业技术为一体化
  - 需要操作系统支持，代码小，执行速度快
  - 专用紧凑，用途固定，成本敏感
  - 可靠性要求高
  - 多样性，应用广泛，种类繁多
- 嵌入式系统应用

  军事、家用、工业用、商用、办公、医用

  热门：个人数字助理PDA、机顶盒STB、IP电话

嵌入式系统概述

- 实时系统：一个能够在指定或确定的时间内完成系统功能以及对外部或内部事件在同步或异步内做出响应的系统
- 实时系统指标：
  - 响应时间
  - 生存时间
  - 吞吐量
- 实时系统分类
  - 根据响应时间分为3种类型 ： 强实时系统、弱实时系统和一般系统
  - 根据确定性可分为 ： 硬实时和软实时

嵌入式处理器

- 嵌入式处理器分类：MCU、EMPU、DSP、SoC
- 嵌入式微处理器特点：低功耗、可扩展结构、存储区保护、调试功能和实时多任务支持
- 微控制器
- 片上系统
- DSP
- 典型的嵌入式处理器

嵌入式操作系统

- 操作系统的概念　一组计算机程序的集合，用来有效地控制和管理计算机的硬件和软件资源，即合理地对资源进行分配、调度、控制、协调并开发活动，并为用户提供方便的应用接口。它必须体现其所在的系统的特征，能够通过装卸某些模块来达到系统所要求的功能。主要 功能有：
  - 处理器管理
  - 存储器管理
  - 设备管理
  - 文件管理
  - 用户接口

嵌入式操作系统

- 操作系统的特点
  - 可装卸性：开放性、可伸缩性的体系结构
  - 较强的实时性：EOS实时性强，可用于各种设备控制当中
  - 统一的接口：提供各种设备驱动接口
  - 操作方便、简单、友好GUI
  - 功能强大的网络功能：支持TCP/IP、USB
  - 强稳定性，弱交互性：嵌入式系统一旦开始运行就不需要用户过多的干预，这就要负责系统管理的EOS具有较强的稳定性
  - 固化代码：操作系统与应用软件被固化在嵌入式系统计算机ROM中
  - 更好的硬件适应性：能支持多种处理器构架，具有良好的移植性

嵌入式操作系统

- 操作系统分类

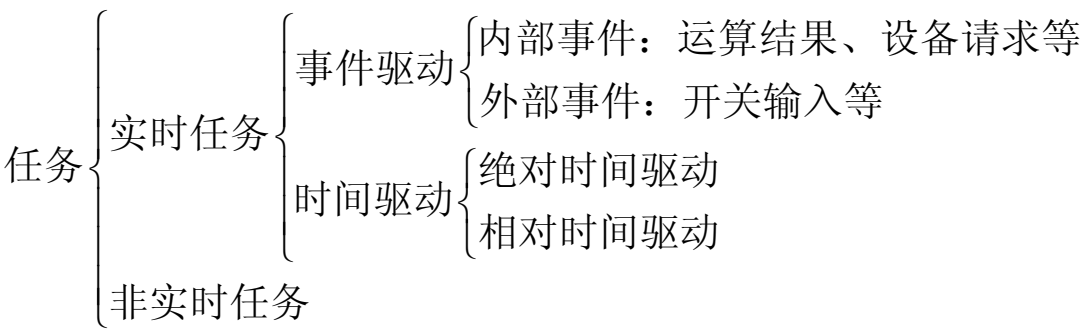
  从应用角度分为：

  - 面向低端信息家电 智能电话
  - 面向高端信息家电 数字电视
  - 面向个人信息终端 PDA
  - 面向通信设备
  - 面向汽车电子
  - 面向工业控制

  从实时性的角度分为 ：强实时、弱实时和没有实时

嵌入式操作系统

- 几个主要的概念
  - 任务
  - 进程
  - 线程
  - 多任务
  - 多用户



嵌入式操作系统

- 几个主要的概念
  - 中断
  - 中断优先级
  - 同步
  - 异步：随机发生的事件
  - 资源：程序运行时可使用的软、硬件环境统称为资源
  - 容错
  - 安全性

嵌入式操作系统

- 实时操作系统
  - 实时操作系统RTOS是具有实时性且能支持实时控制系统工作的操作系统。
  - RTOS与通用计算机OS区别：
    - 实时性。
    - 代码尺寸小
    - 应用程序开发较难

    例如：VxWorks、Uc/OS、Nucleus
  - 实时操作系统的组成
    - 实时内核
    - 网络组件
    - 文件系统
    - 图形用户界面

嵌入式操作系统

- 实时操作系统
  - 常见的嵌入式操作系统
    - VxWorks：美国WindRiver公司设计开发。在美国F－16、FA－18战斗机等。支持：ARM、x86、MC68000。特点是：高性能实时微内核；与POSIX兼容；自由配置；友好开发调试环境。
    - μC/OS-Ⅱ：Jean J.Labrosse开发。与Linux相似，只有8.3KB。只包含进程调度、时钟管理、内存管理和进程间的通信与同步等基本功能，而没有包括I/O管理、文件系统、网络等额外模块。抢占式、多任务系统，最多可以运行64个进程
    - WindowsCE：Microsoft公司，多线程、完整优先权、多任务的操作系统。200KB
    - 嵌入式Linux：RT-Linux、μClinux

嵌入式操作系统

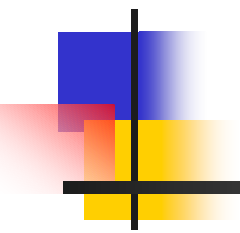
- 嵌入式技术发展现状及趋势

  宏观方面发展趋势：

  - 经济性 买得起
  - 小型化 PDA，携带方便
  - 可靠性 汽车
  - 高速度 飞机
  - 智能性 知识推理、模糊查询、识别和感知运动

习题

- 什么是嵌入式系统？由几部分组成？有何特点？
- 什么是实时系统？有何特征？如何分类？
- RTOS由哪几部分组成？由哪些特点？与一般操作系统有何不同？



2 嵌入式系统开发过程

嵌入式系统开发特点

- 需要交叉开发环境：

  交叉开发环境是指具有实现编译、链接和调试应用程序代码的环境。与运行应用程序的环境不同，它分散在有通信连接的宿主机与目标机环境之中。

  - 宿主机（Host）是一台通用计算机，一般是PC机
  - 目标机（Target）嵌入式应用软件的实际运行环境，也可以是能替代实际环境的仿真系统
  - 交叉软件开发工具包括交叉编译器、交叉调试器和模拟软件等。

嵌入式系统开发特点

- 引入任务设计方法
- 需要固化程序
- 软件开发难度大

绝大多数嵌入式应用有实时性要求。实时性的体现一部分来源于实时操作系统的实时性，另一部分来源于应用软件本身的设计和代码的质量。

同时，嵌入式应用软件对稳定性、可靠性、抗干扰性等性能的要求更为严格和苛刻。

嵌入式系统开发流程

- 需求分析阶段

  包括功能需求、操作界面需求和应用环境需求等。包括以下3个方面

  - 对问题的识别和分析 对提出的问题进行抽象识别以产生以下需求：功能需求、性能需求、环境需求、可靠性需求、安全需求、用户界面需求、资源使用需求、软件开发成本与开发进度需求；
  - 制订规格说明文档 包括需求规格说明书和初级的用户手册等；
  - 需求评审 包括正确性、无歧异性、安全性、一致性、可验证性、可理解性、可修改性、可扩展性和可追踪性等

嵌入式系统开发流程

- 设计阶段

  系统设计、任务设计和任务的详细设计

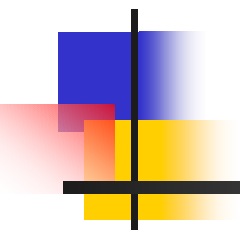
- 代码生成阶段

  要完成的工作包括：代码编程、交叉编译和链接、交叉调试和测试等

- 固化阶段
- 嵌入式软件开发要点
  - 尽量用高级语言开发，少用汇编语言。
  - 局域化不可移植部分
  - 提高软件的可重用性

习题

- 嵌入式系统开发过程分为几个阶段？



3 Linux操作系统介绍

LINUX简介

Linux是一个类似于Unix的操作系统。起源于芬兰一个名为Linus Torvalds的业余爱好，但是目前已经成为流行的一款开发源码的操作系统

Linux从1991年问世到现在，短短10年的时间内已发展成为一个功能强大、设计完善的操作系统，伴随网络技术进步而发展起来的Linux OS已成为Microsoft公司Windows的强劲对手。Linux系统不仅能够运行于PC平台，还在嵌入式系统大放光芒，在各种嵌入式Linux OS迅速发展的状况下，Linux OS逐渐形成了可与Windows CE等EOS进行抗争的局面。

目前正在开发的嵌入式系统中，70%以上的项目选择Linux作为嵌入式操作系统。Linux现已成为嵌入式操作的理想选择。

LINUX简介

- Linux特点
  - 多任务和32位操作系统
  - X Window系统 X Windows是UNIX平台上，XFree86则是Linux平台上
  - 支持TCP/IP协议
  - 虚拟内存和共享库
  - Linux内核代码均为自由代码
  - Linux支持商业版UNIX全部功能
  - GNU软件支持
  - Linux符合IEEE POSIX.1标准
  - 支持多种硬件平台
  - 网络功能强大

LINUX简介

Linux操作系统有三个主要部分：Kernel、Shell、目录树

- Kernel是Linux操作系统的核心，管理计算机的所有物理资源的控制程序，包括：文件系统、设备管理、进程管理和内存管理
- Shell是Kernel和用户之间的接口

  Shell的一个主要功能是作为命令翻译器

  Shell接收键入的命令，对这些命令进行翻译，然后执行

LINUX简介

- 目录树 主要部分有root、/usr、/var及/home等
  - Root 包含与特定计算机相关内容。引导系统必备文件、挂载信息、系统修复和备份工具
  - /usr 通常操作中不需要经常修改的命令程序文件
  - /var 包含经常变化的文件，如打印机、邮件
  - /home 包含用户的主目录
  - /pro 不保存在磁盘上，操作系统在内存中创建这一文件系统。包含一些和系统相关的信息，例如CPU、DMA通道以及中断的使用信息等

Linux帐户简介

- 用户帐号

  每个用户在系统中都必须有一个帐号，用来登录系统。全部帐号在/etc/passwd文件中定义，该文件也包含了用来识别用户唯一性的元素

- Root帐号

  帐号和密码在Linux安装过程中设定。是系统管理员的登录帐号

- 登录(Logging in)

  Telnet是进入系统的入口，可以在登录屏幕输入你的登录名称和密码

LINUX命令介绍

- 开机后输入用户名和口令才可进入LINUX环境。Root用户是超级用户，其它用户均由root用户创建并赋予不同的操作权限。
- logout:注销当前用户，重新登陆。
- reboot：重新启动主机
- shutdown 0：立即关机。Shutdown 0 执行完毕后可关断电源
- [Command] - - help: 获取[command] 命令的帮助信息。
- ls {file or dir}: 查看{file}文件或者dir下所有文件列表。省略file和dir表示查看当前目录下所有文件列表。
- chmod命令来制定的文件或目录的访问权限和属性

  chmod +x file 将文件file的属性改为可执行。只有具有可执行属性文件才可被执行。 ./[app]: 执行当前目录下的app程序。

LINUX命令介绍

- ll: {file or dir}: 查看{file}文件或者dir 下所有文件的详细属性。省略file和dir表示查看当前目录下所有文件。
- cd [directory]: 进入[directory]目录。根目录用 /表示， 上级目录用..表示，当前目录用./表示；

$cd trainer

$cd /home/trainer

- pwd可以打印出当前工作目录

$pwd

- rm {-f} {-r} [file or dir] 删除file文件或者dir目录。 -f 表示无须确认强制删除。 -r 表示要删除目录，与dir配合使用。
- cp [sfile] [dfile] 拷贝sfile并命名为dfile
- mv [sfile] [dfile] 移动sfile 为dfile。兼具改名的作用

LINUX命令介绍

- mount [dev] [dir] 挂装dev设备上的文件系统到目录dir。
- umount 卸载dev。如果挂接的目录里有活动，则文件资源不可摘除。 #umount mount_ponit
- tar zxvf [file] [--directory=dir] 将压缩文件file(通常扩展名为.tar.gz)解压到dir目录下。如果没有指定[--directory=dir]，则解压到当前目录下）。

  #tar zcvf [file] [file or dir]：将[file or dir]压缩并命名为[file]文件。
- Make [-f file] 按照file文件指定的编译规则编译源代码。如果没有指定[-f file]，那么make程序自动在当前目录下寻找名makefile 或者 Makefile的文件作为编译规则文件。

VIM文本编辑器的使用

1 VIM是LINUX程序员最喜爱的编辑器，VI最大的优点是稳定可靠，功能强大，最大限度的保护了编辑成果和提高编辑效率。缺点是为了避免图形引擎的不稳定，VI采用字符界面，如果要使用VI强大的高级功能需要熟记众多的命令键。

2 进入VIM编辑器

vim file ： 打开file文件，如果file文件不存在，则创建file文件。

3 VIM编辑器的三种状态

命令态：此状态下的键盘输入被认为是VIM控制命令而非文本内容。

插入态：此状态下的键盘输入被认为是要插入光标处的文本内容。

改写态：此状态下的键盘输入被认为是改写插入光标后面旧文本的新文本。

三种状态分别在VI界面的左下脚显示 “空白” “-- REPLACE –” “-- INSERT –”

VIM文本编辑器的使用

4 VIM编辑器的三种状态的切换

“Esc”键可从当前状态切换至命令态

“Insert”键可从当前状态切换至插入态

“Delet”键可从当前切换至改写态

5 VIM编辑器的基本命令 以下命令包含“：”号，需要在命令态下输入

“:x” 保存文本内容并退出VIM

“:q” 放弃保存文本内容并退出VIM “:q！” 强制放弃保存文本内容并退出VIM

“:w” 保存文本内容但不退出VIM

“:/字符串 ” 查找文本中的字符串并加亮标记。

“:行号” 将光标跳到指定的行号。

“dd” 删除当前光标所在的那一行。

MINICOM的使用

1 minicom是linux下一个应用程序，功能与windows下的超级终端类相同，是linux环境下一个字符界面的“超级终端”

2 启动minicom：

以root身份登陆linux

minicom

3 配置minicom：

Ctrl+a o（先按Ctrl + A 再按O）。

移动上下方向键，菜单中选择serial port setup，回车确定。

按“e”键选择修改“Bps/Par/Bits ” 在弹出的菜单里按“i”键选择115200作为波特率，然后按“q”选择 8-N-1方式。回车保存修改并返回上级菜单。

按“a”键选择串口，键入“/dev/ttyS0”选择串口1或者键入“/dev/ttyS1”选择串口2（请注意大小写）。输入完成后按回车保存修改结果。

MINICOM的使用

按“f”键直到“Hardware Flow Control”后面显示为“no”

按“e”键直到“Software Flow Control”后面显示为“no”

回车保存修改并退回上级菜单。

移动上下箭头选择“Modem and dialing ”按回车进入。

按“a”键修改“Init string”，删除 “Init string”后所有字符，按回车保存

按“b”键修改“Reset string”，删除 “Reset string”后所有字符，按回车保存

按回车保存修改结果并返回上级菜单。

移动上下箭头选择“Save setup as dfl ”将所有修改保存为默认值。

移动上下箭头选择“Save setup as dfl ”将所有修改保存为默认值。

移动上下箭头选择“Exit”退出配置菜单，返回minicom界面开始使用。

（如果下次运行时不需要改变配置，可不再重复以上步骤）

MINICOM的使用

4 使用minicom

minicom　启动minicom

此时所有的键盘输入字符将通过串口发送出去，所有从串口接收到的字符将被显示到屏幕上。

5 在minicom中通过串口发送文件到远端

在minicom中，当需要向远端发送文件的时候，按“Ctrl+a o”（先按Ctrl + a 再按o）在弹出的界面中移动上下方向键选择使用的传输协议，并按回车确定。在随后的界面中我们可以看到硬盘的目录树和文件。其中带[]的是目录而[..]表示上级目录。移动上下方向键将光标想要进入的目录旁，按2次空格键进入该目录，移动上下方向键将光标想要发送的文件旁，按1次空格键选中该文件，如果再按一次空格键将取消选中标记。

MINICOM的使用

选中要发送的文件后（每一次发送请只选择一个文件），按左右方向键直到屏幕下端“[Okay]”被选中。（在以上过程中如果不想发送文件了，可以按“Esc”键退回到minicom主界面。）

按回车键，所选中的文件将立刻被发送。

如果在文件发送过程中要中止发送。可以按“ctrl－c”组合键。

6 退出minicom

在minicom主界面下按“Ctrl+a q”（先按Ctrl + a 再按q）”

4 GNU工具的使用

Richard Stallman---这个有史以来最伟大的黑客、软件自由主义的圣斗士---于1984年举起了“自由软件”的大旗。他创立了自由软件基金会(FSF)，开始了GNU计划，按照GPL(通用公共许可证)和Copyleft的原则编写和发布自由软件。

GCC编译器的使用

1 GCC是GNU项目免费提供的功能强大的C编译器

2 使用GCC

gcc hello.c -o hello_app

将hello.c 文件编译连接，将结果输出为hello_app 文件。如果不指定 -o hello_app 那么会将编译结果输出为a.out文件。

3 使用参数

GCC的整个编译过程，实质上是分四步进行的，每一步完成一个特定的工作，这四步分别是：预处理，编译，汇编和链接。它具体完成哪一步，是由GCC后面的开关选项和文件类型决定的。

GCC编译器的使用

4 常用参数和选项

-o选项表示要求编译器生成指定文件名的可执行文件；

-c选项表示只要求编译器进行编译，而不要进行链接，生成以源文件的文件名命名但把其后缀由.c或.cc变成.o的目标文件；

-g选项要求编译器在编译的时候加入调试的信息；

-E选项表示编译器对源文件只进行预处理就停止，不做编译，汇编和链接；

-S选项表示编译器只进行预处理和编译，不做汇编和链接；

-O选项是编译器对程序提供的编译优化选项，在编译的时候使用该选项，可以使生成的执行文件的执行效率提高；

-Wall 选项指定产生全部的警告信息。

-L(lib file) 连接源程序中用到的非标准库 如果前面没有 —static选项，那么默认的使用动态库。

Make工具简介

1 make是负责从项目的源代码中生成最终可执行文件和其他非源代码文件的工具。

make {-f filename}

Make 根据指定的filename文件中的规则，或者默认使用当前目录下makefile或者Makefile文件中的规则执行包括编译拷贝删除等在内的众多操作。后面可以看到Make的功能非常强大，配置和编译内核也要用到它，make可以执行所有的linux命令，甚至可以使用FTP下载文件。

2 makefile文件是一个文本文件，定义了make要执行的动作和规则

Make工具简介

3 一个实际的makefile文件 ——ad试验的makefile文件

```
KERNELDIR = /skiff/local/arm-linux/include
CGFLAGS   = -I$(KERNELDIR) -O2 -O -Wall -mtune=arm9tdmi
all:clean ad adc
ad:ad.c mx1_def.h
        arm-linux-gcc $(CGFLAGS) -c ad.c -o ad.o
adc:adc.c mx1_def.h
       arm-linux-gcc -g adc.c -o adc
.PHONY:clean
clean:
        -rm -rf ad.o
        -rm -rf adc
```

Make工具简介

KERNELDIR 和 CGFLAGS是用户定义的宏的名称，“＝”后面是宏的值（以行首开始的后面跟有等号的行是宏定义行）

当makefile中宏定义行的后面任意地方要引用该宏的值时使用 $(宏名)

all ad adc是目标体，“:””后面是该目标体依赖的依赖体列表。下面连续以“tab”开始的行定义了生成目标体的命令。（以行首开始的后面跟有冒号的行以及下面连续以“tab”开始的行是目标体定义）。目标体的依赖体可以是文件也可以是其它的目标体也可以为空。没有依赖体的目标将被认为是最新的，不被执行。

```
ad:ad.c mx1_def.h

    arm-linux-gcc $(CGFLAGS) -c ad.c -o ad.o
```

如果目标体ad不存在或两个依赖体的任意一个据上次执行make后被更新过，则执行arm-linux-gcc $(CGFLAGS) -c ad.c -o ad.o。但是本例中，目标体的命令并没有能生成目标体ad。因此，下次执行make时该目标体仍将被执行。

Make工具简介

all:clean ad adc

all是默认目标体（因为它是makefile中的第一个目标体）。All依赖于目标体clean，ad和adc。因此make会转而检查更新clean，ad和adc这三个目标体。类似目标体ad，all是个伪目标体，因为所执行的命令并没有生成all文件

clean:

```
-rm -rf ad.o

-rm -rf adc
```

伪目标体clean没有依赖体，make将认为该目标体是最新的，无需执行。除非使用 # make clean 或者将clean作为依赖体加入其它的目标体依赖体列表中。

.PHONY:clean

如果真的存在一个clean的文件，那么make会认为clean文件是clean目标体的生成文件，因为clean目标体没有依赖体，所以make会认为clean文件是最新的，因而即使运行了make clean 也不会执行clean目标体的命令。 .PHONY: clean声明禁止make检查clean是否更新而强制执行clean的命令。

GDB调试器简介

1 GDB(GNU Degugger)是GNU自带的调试工具

是一个能够运行其他程序的应用程序，能够进入到程序源码中，进行逐行单步运行，查看每条语句执行的结果，查看甚至是改变任一变量值。

2 编译可被GDB调试的二进制文件， 需要在编译时加入 —g选项

```
# gcc  -g hello.c  -o hello
```

gcc在编译的时候将加入调试信息，这些调试信息存在目标文件中，它描述了每个函数或变量的数据类型以及源码行号和可执行代码地址间对应关系，gdb正是通过这些信息使源码和机器码相关联的，它实现了源码级的调试。

3 被调试的程序应该与源代码在同一目录下

4 运行和退出gdb

```
# gdb            运行gdb
(gdb) quit       从gdb中退出 (gdb) 是gdb环境下的命令提示符
```

GDB调试器简介

5 常用GDB命令

file {name}	装入想要调试的可执行文件
list [col]	列出从指定行号或者当前行号开始的源代码
break {col or funname} {con}	设定断点或者设定满足con条件时中断
continue	继续执行
next	执行一行源代码但不进入函数内部
step	执行一行源代码而且进入函数内部
quit	中止gdb
Print var	打印变量val的值
Whatis var	查看变量val的类型
Set variable var＝val	设置变量var的值为val

5 Linux下应用程序的开发

库函数介绍

- 1 低级文件操作函数 低级文件操作函数因为没有使用缓冲机制，所以非常适合对设备进行操作

  int open(const char * path, int oflags);

- 2 标准文件函数

  FILE * fopen（const char * filename,const char * mode）

- 3 内存管理函数

  void * malloc(size_t size);

编写一个HelloWorld应用程序

- 在Linux环境下使用VIM编辑器，在当前目录下新建一个hello.c

#vim hello.c

键入如下内容：

```
#include <stdio.h>
int main() {
        printf(“Hello World!\n”);
        return 0;
}
```

编写一个HelloWorld应用程序

- 编写HelloWorld的Makefile文件

在同一目录下建立makefile文件

#vim makefile

键入如下内容：

```
CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc
KERNELDIR = /home/lh/linux-2.4.19-rmk7/include
CGFLAGS = -I$(KERNELDIR)
All:hello
Hello:hello.c
        $(CC) $(CGFLAGS) hello.c –o hello.o
.PHONY:clean
clean:
        rm –rf hello.o
```

注意，在$(CC)和rm两行前面的空格是一个Tab键制表符，不能用空格代替

编写一个HelloWorld应用程序

- 在同一个目录下，敲入make命令

#make

#ls

就可以看到名为hello.o的目标程序

Linux驱动程序开发

- 设备驱动程序说明:

设备驱动程序隐藏了设备的具体细节，对各种不同设备提供了一致的接口。一般来说是把设备映射为一个特殊的设备文件，用户程序可以象对其它文件一样对此设备文件进行操作

- 设备驱动程序的分类

1. 块设备驱动：通过块（字符）特别设备文件存取的设备称为块（字符）设备或具有块（字符）设备接口，如ram驱动，flash驱动
2. 字符设备驱动 支持面向字符的I/O操作，它不经过系统的快速缓存，所以它们负责管理自己的缓冲区结构，如：led灯驱动
3. 网络设备驱动

Linux驱动程序开发

- Linux驱动程序组成

设备驱动程序设备号：

1.主设备号 主设备号唯一标识了设备类型，即设备驱动程序类型，它是块设备表或字符设备表中设备表项的索引

2.次设备号 仅由设备驱动程序解释，一般用于识别在若干可能的设备中，I/O请求所涉及到的那个设备

设备驱动程序组成部分：

1.自动配置和初始化子程序 负责检测所要驱动的硬件设备是否存在和是否能正常工作，如：init_module()

2.服务于I/O请求的子程序 调用这部分是由于系统调用的结果。这部分程序在执行的时候，系统仍认为是和进行调用的进程属于同一个进程，只是由用户态变成了核心态

Linux驱动程序开发

- 中断服务子程序

  在Linux系统中，并不是直接从中断向量表中调用设备驱动程序的中断服务子程序，而是由Linux系统来接收硬件中断，再由系统调用中断服务子程序

- 设备驱动程序存取

  设备的存取通过一组固定的入口点来进行，这组入口点是由每个设备的设备驱动程序提供的。一般来说，字符型设备驱动程序能够提供如下几个入口点：

  open入口点：打开设备准备I/O操作。对字符特别是设备文件进行打开操作，都会调用设备的open入口点。Open子程序必须对将要进行的I/O操作做好必要的准备工作，如：初始化等

Linux驱动程序开发

- 设备驱动程序存取

read入口点

从设备上读数据。对于有缓冲区的I/O操作，一般是从缓冲区里读数据，对字符特别设备文件进行读操作将调用read子程序。

write入口点

往设备上写数据。对于有缓冲区的I/O操作，一般是把数据写入缓冲区里，对字符特别设备文件进行写操作将调用write子程序

ioctl入口点

执行自定义的命令

select入口点

检查设备，看数据是否可读或设备是否可用于写数据

Linux驱动程序开发

- 驱动程序入口点定义

在Linux系统里，设备驱动程序所提供的这组入口点由一个结构来向系统进行说明，此结构定义如下：

```
#include <linux/fs.h>
struct file_operations {
        int (*lseek)(struct inode *inode,struct file *filp, off_t off,int pos);
        int (*read)(struct inode *inode,struct file *filp,char *buf, int count);
        int (*write)(struct inode *inode,struct file *filp,char *buf,int count);
        int (*readdir)(struct inode *inode,struct file *filp,struct dirent *dirent,int
                        count);
        int (*select)(struct inode *inode,struct file *filp,int sel_type,select_table
       *wait);
        int (*ioctl) (struct inode *inode,struct file *filp,unsigned int cmd,unsigned
       int arg);
        int (*mmap) (void);
        int (*open) (struct inode *inode, struct file *filp);
        void (*release) (struct inode *inode, struct file *filp);
        int (*fsync) (struct inode *inode, struct file *filp);
```

Linux驱动程序开发

- Linux驱动程序注册

注册字符型设备驱动程序：通过调用register_chrdev向系统注册字符型设备驱动程序。reqister_chrdev定义为：

```
#include＜linux/fs.h>
#indude ＜linux/errno.h>
int register_chrdev（unsigned int major,const char *name，
struct file_operations *fops）；
```

major是为设备驱动程序向系统申请的主设备号，如果为0则系统为此驱动程序动态地分配一个主设备号。name是设备名，fops就是前面所说的对各个调用的入口点的说明。此函数返回0表示成功，返回-EINVAL表示申请的主设备号非法

register_chrdev操作成功，设备名就会出现在 / proc/devices文件里

Linux驱动程序开发

- Linux驱动程序加载和卸载
  - 设备驱动程序加载和卸载

设备驱动程序加载

insmod命令加载，用lsmod命令来查看所有已加载的设备驱动程序的状态

int init_module(void) 在加载设备驱动程序的时候自动调用，负责进行设备驱动程序的初始化工作

init_module返回0以表示初始化成功，返回负数表示失败

设备驱动程序卸载

rmmod命令来卸载，用lsmod命令来查看是否已经成功卸载

void cleanup_module (void)，在设备驱动程序被卸载时调用，负责进行设备驱动程序的除工作

  - 设备映射文件

设备映射文件：

成功的向系统注册了设备驱动程序后（调用register_chrdev成功后）用mknod命令来把设备映射为一个设备文件，其它程序使用这个设备的时候，只要对此设备文件进行操作就行了

Linux驱动程序开发

- Hello驱动程序初始化

```
#define MODULE
#include <linux/module.h>

int init_module(void)
{
    printk("Hello,world\n");
    return 0;
}
```

说明：

hello驱动程序初始化入口函数，当hello驱动程序被加载时被执行，用insmod hellodrv.o命令加载

在超级终端中显示： Hello,world

Linux驱动程序开发

- Hello驱动程序关闭

```
#define MODULE
#include <linux/module.h>

void cleanup_module(void)
{
    printk("Goodbye,world\n");
}
```

说明：

hello驱动程序退出入口函数，当hello驱动程序被卸载时被执行，用rmmod命令去卸载hello驱动程序

在超级终端中显示： Goodbye,world

Linux驱动程序开发

- 编写Hello驱动程序makefile文件

```
KERNELDIR = /home/lh/linux-2.4.19-rmk7/include
CGFLAGS   = -I$(KERNELDIR)
all:hellodrv.o
hellodrv.o: hellodrv.c
    /usr/local/arm/2.95.3/bin/arm-linux-gcc     $(CGFLAGS)     -c
    hellodrv.c -
                                                o hellodrv.o
.PHONY:clean
clean:
    rm -rf hellodrv.o
```

注意：arm-linux-gcc 的选项是-c

Linux驱动程序开发

- 编译Hello驱动程序

在同一目录下，运行make命令

#make

#ls

就可以看到名为hellodrv.o的驱动程序

Linux驱动程序开发

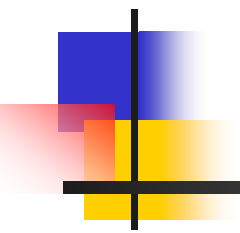
- 加载Hello驱动程序

将hellodrv.o下载到实验箱上的/tmp目录下

加载hello驱动程序，用命令：insmod hellodrv.o

显示当前已经加载的驱动程序，用命令：lsmod

加载hello驱动程序，用命令：rmmod hellodrv.o



本部分内容介绍结束