華東理工大學

《嵌入式系统》

实验报告本

2023 年到 2024 学年第 二 学期

专　业	计算机科学与技术
班　级	计 213
姓　名	郑铠涛
学　号	21013122
指导教师	罗飞

计算机实验教学中心

2024　年　6 月

《嵌入式系统》实验报告 1

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/4/26

成绩： _____ **指导教师：** 罗飞

实验名称：嵌入式 Linux 基础实验	**实验地点**：信息楼 216
实验仪器： ECS 云实验环境--Linux 实验环境	

一、实验目的：

1、 掌握 Linux 各类命令的使用方法；
2、 熟悉 Linux 操作环境
3、 了解 Makefile 的基本概念和基本结构
4、 初步掌握编写简单 Makefile 的方法
5、 了解递归 Make 的编译过程
6、 初步掌握利用 GNU Make 编译应用程序的方法

二、实验内容：

1. 基于实验指导手册，练习使用 Linux 常用命令，简述主要命令的含义

ls: 列出目录中的文件以及子目录。
mkdir: 创建新的目录
cd: 进入目录
vi: 打开文本编辑器(vi)
cp: 复制文件
less: 查看文本文件内容
grep: 文件中搜索指定的模式，并输出包含该模式的行
tar: 创建、查看或提取 tar 归档文件
rm: 删除文件或目录
date: 显示以及设置系统日期以及时间
free: 显示系统内存的使用情况
df: 显示磁盘空间的使用情况

完成实验指导手册中的内容二--四，进而完成如下工作：

2. 新建项目目录 project-1，完成如下功能：

1）在该项目中编写 hello.c 程序，该程序输出“Hello,world！”；
2）通过 gcc 命令编译该文件，运行该可执行程序。

简述实验步骤。

使用 mkdir project-1 创建目录，然后 cd project-1 进入目录。
使用 vim hello.c 创建文件，输入 i 进入编辑模式，输入以下程序。

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");

    return 0;
```

}

输入 ESC 退出编辑模式，使用命令:wq 保存文件

使用命令 gcc -c hello.c 编译，使用命令 gcc hello.o -o hello 链接，最后 ./hello 运行文件

3. 新建项目目录 project-2，完成如下功能：

1）在该项目中编写 makefile-hello.c 程序；

2）该程序输出“Hello,world for Makefile！”；

3）编写 Makefile，通过 Makefile 编译 makefile-hello.c，并运行可执行程序。

简述实验步骤。

使用 cd .. 返回上一层，同时使用命令创建目录 mkdir project-2，并使用 cd project-2 进入

使用命令 vim makefile-hello.c 创建文件，文件内容如下：

```
#include <stdio.h>

int main() {
    printf("Hello, world for Makefile!\n");

    return 0;
}
```

同样退出保存后，使用 vim makefile 新建 makefile 文件，内容如下：

```
CC=gcc
CFLAGS=

OBJS=hello.o
all: hello
hello: $(OBJS)
 $(CC) $(CFLAGS) $^ -o $@
hello.o: makefile-hello.c
 $(CC) $(CFLAGS) -c $< -o $@

clean:
 rm -rf hello*.o
```

保存退出

使用命令 make 编译文件，然后执行 ./hello 命令

4. 新建项目目录 project-3，完成如下功能：

1) 编写 C 语言文件 file1.c、file2.c 和相关的头文件；

2）file1.c 输出：“This is the first file!”；

3）file2.c 输出：“This is the second file!”；

4）编写 Makefile，编译该工程，并运行可执行程序，依次输出：

This is the first file!

This is the second file!

简述实验步骤。

同样使用 mkdir project-3 新建文件夹，并进入

创建以下文件:

file1.c:

```
#include <stdio.h>
#include "file1.h"

void showFile1(void) {
    printf("This is the first file!\n");
}
```

file1.h:

```
void showFile1(void);
```

file2.c:

```
#include <stdio.h>
#include "file2.h"

void showFile2(void) {
    printf("This is the second file!\n");
}
```

file2.h:

```
void showFile2(void);
```

main.c:

```
int main() {
    showFile1();
    showFile2();
}
```

makefile:

```
CC=gcc
CFLAGS=

CFILES=$(wildcard *.c)
OBJS=$(CFILES:%.c=%.o)
all: main
main: $(OBJS)
    $(CC) $(CFLAGS) -o main $(OBJS)
.c.o:
    $(CC) -c $<
clean:
```

rm -rf *.o

使用命令 make 编译，然后使用./main 运行

5. Linux 命令综合：根据各人学号 or 姓名，为自己新建一个工作目录，并利用 pwd 命令查看自己的工作目录；将前述工程 project-1 到 project-3 打包成 back.tar.g；简述实验步骤：

使用命令 mkdir 21013122 新建工作目录，使用命令 pwd 21013122 查看工作目录

使用命令 tar zcvg back.tar.g project-1 project-2 project-3 打包

使用命令 mv back.tar.g 21013122 将打包文件移动到工作目录

《嵌入式系统》实验报告 2

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/5/17

成绩： _____ **指导教师：** 罗飞

实验名称：嵌入式 Linux 开发环境	**实验地点**：信息楼 216
实验仪器：Linux 实验环境	

一、实验目的：

1、熟悉命令行操作方式进行用户和用户组管理
2、掌握 vi 编辑器的进入与退出方法
3、了解文本编辑器的三种模式
4、熟练掌握使用 vi 编辑器进行编辑、选择及操作文本文件的命令
5、安装和交叉编译环境
6、理解和掌握 bootloader 的使用方法

二、实验内容：

1. 创建用户（以 zhangsan 为例）

(1) 创建一个新用户 zhangsan
(2) 查看/etc/passwd 文件的最后一行内容，并记录
(3)查看/etc/shadow 文件的最后一行内容，并记录
(4) 给新建的用户设置密码
(5) 查看/etc/shadow 文件的最后一行内容，记录并说明变化
(6) 使用 zhangsan 用户登录系统，测试能否登录成功

简述实验的详细步骤

使用 sudo useradd zhangsan 新建用户;
使用 cd /etc 进入到 etc 目录
使用 tail –n 1 passwd 查看，内容如下: zhangsan:x:1002:1002::/home/zhangsan
使用 sudo tail –n 1 shadow(需要管理员身份) 查看 zhangsan:!:19860:0:99999:7:::
使用 sudo passwd zhangsan 设置密码，此时最新的/etc/shadow 文件如下:
Zhangsan:6GfI/vyLH$E/TqCj……../:19860:0/99999:7::: ，在用户名后面多了加密密码字段
使用 su zhangsan 命令进行登录

2. 创建用户 2

（1）使用 1 的步骤创建新用户 user
（2）更改 zhangsan 所属群组为 root
（3）查看/etc/passwd 文件，记录 zhangsan 用户和 user 用户的属组情况
（4）删除用户 user

简述实验的详细步骤。

在 ecust 用户中 使用命令 sudo useradd user 创建用户
使用命令 sudo usermod –g root zhangsan 修改用户群组
使用命令 tail –n 2 passwd, 其中 zhangsan 用户的组为 0, user 用户的组为 1003
使用命令 sudo userdel –r user 删除用户

3. 组的管理

（1）创建一个新组，组名为 stuff

（2）查看/etc/group 文件的最后一行内容，并记录

（3）创建一个新账户 test，并将其起始组合附属组都设置为 stuff

（4）查看/etc/group 文件中的最后一行内容，记录并说明变化

（5）设置 stuff 组密码

（6）在 stuff 组中删除用户 test

（7）查看/etc/group 文件中的最后一行，记录并说明变化

简述实验的详细步骤。

sudo groupadd stuff 新建组

tail –n 1 group 查看，如下: stuff:x:1005

使用 sudo useradd test 创建账号，显示已经存在 test 组，选择创建 test1 账号

sudo useradd test1 创建账号，sudo usermod –aG stuff test1 设置组(包括附属组)

使用 cat group 命令，最后两行为 stuff:x:1005:test1, test1:x:1003，增加了账号 test1

使用命令 sudo gpasswd stuff 给 stuff 组设置密码

使用命令 sudo gpasswd -d test1 stuff 组中删除用户

此时使用 cat group 查看 group 文件，最后两行为 stuff:x:1005:, test1:x:1003:, stuff 组没有了 test1

4. vim 编辑器的使用

（1）进入 vim 编辑器，建立一个文件，如 file.c；进入插入方式，输入一个 C 语言程序的各行内容，故意制造几处错误。最后，将该文件存盘。回到 shell 状态下。

（2）运行 gcc file.c -o myfile，编译该文件，会发现错误提示。说明其含义。

（3）重新进入 vim 编辑器，对该文件进行修改。然后存盘，退出 vi 编辑器。重新编译该文件，如果编译通过了，可以利用./myfile 运行该程序。

（4）运行 man date > file10，然后 vi file10。使用 x，dd 等命令删除某些文本行。使用 u 命令复原此前的情况。使用 c、r、s 等命令修改文本内容。使用检索命令进行给定模式的检索。说明以上各个命令的作用。

简述实验的详细步骤。

建立 file.c, 内容如下

```
#include <stdio.h>
#include <stdlib>
int mian() {
    print("Hello World!");
    printf("\n")
    return 0;
}
```

第一次错误:

stdlib: No such file or directory, 找不到头文件,应该为 stdio.h

第二次: In function mian: expected ';' before ruturn ,缺少分号

第三次: undefined reference to 'mian', 'print'

没有启动函数，print 函数找不到

mian 改为 main, print 改为 printf

修改错误后，成功编译

运行 man date > file10 后，使用 vi file10 打开

x 命令： 删除当前字符； dd 命令: 删除当前行；

r 命令: 移除当前字符或者单词 (remove)； c 命令：改变当前字符或者单词(change),比如 ci”

s 命令: 替换,substitute

/命令，查找，比如/-f, 查找字符”-f”

进一步完成实验指导手册中的实验，并完成如下内容：

5. 简述安装 ToolChain 的主要步骤（说明 source 命令的作用）。

首先使用 tar jxvf /home/ecust/samba_share/embed/ToolChain/toolchain-4.5.1-farsight. tar.bz2 解压文件

然后对 toolchain-4.5.1-farsight 创建名为 toolchain 的快捷方式，使用以下命令:

ln -s toolchain-4.5.1-farsight toolchain

然后修改 bash shell 的配置文件(.bashrc)，将 toolchain/bin 添加环境变量

最后实现配置文件的更改，使用命令 source ~/.bashrc

source 在这里的作用是将对配置文件的更改应用到当前 shell 会话中，而不用关闭会话

6．编辑并编译 C 语言 hello.c，该执行文件可输出：“Hello, world!”；分别写出基于 X86、ARM 架构的编辑、编译步骤，并用 file 查看 X86、ARM 架构下可执行文件的格式，说明其区别。

首先 vim hello.c 创建 hello.c 文件，内容如下:

```
#include <stdio.h>

int main() {
    printf(“Hello, world!\n”);

    return 0;
}
```

X86 架构编译: gcc hello.c -o helloX86

ARM 架构的编译: arm-linux-gcc hello.c -o helloArm

其中 helloX86 的可执行文件格式为: x86-64,

helloArm 为 ARM

两个文件的格式不一样，前者可以直接运行在 linux 中，后者不能直接通过./helloArm 运行

7．bootLoader 程序的编译

说明编译 Bootloader 程序的步骤：

使用 tar -xvf /home/ecust/samba_share/embed/U-Boot/uboot-fs210_V5.3.tar 解压文件

将文件中的 Makefile 中 ROSS_COMPILE 的 toolchain 路径更改为当前 linux 中的路径

清楚非必要的文件 make distclean,

然后使用命令 make fs210_nand_config 构建 fs210_nand_config 目标，最后使用 make 构建

查看 bootloader 源码，说明 include/configs/fs210_nand.h 和 include/configs/fs210_sd.h 文件的区别：

首先进入到 uboot-fs210_V5.3/include/configs 文件夹，

然后使用命令 diff fs210_nand.h fs210_sd.h 查看区别

这两个文件的是在 NAND 闪存以及 SD 卡相关接口的差异 都是宏定义上的区别

首先是 root 参数配置宏 CONFIG_BOOTARGS 的区别

前者是 /dev/mtdblock3 rootfstype=yaffs2, 后者是 /dev/mmcblk0p2 rootfstype=ext4

再次就是 CONFIG_BOOTCOMMAND 的不同，后者会多读取一次

8. 如何将编译好的 bootloader 下载到目标板中？请说明主要步骤。

1. 使用适当的接口和电缆将目标机器连接到计算机。
2. 打开目标机器并进入 bootloader 模式
3. 使用与目标机器架构兼容的工具将编译好的 bootloader 传输到目标机
4. 配置 nand 或者 sd 的必要设置
5. 将编译好的二进制文件传输到目标机器的内存中
6. 验证是否可以成功运行

《嵌入式系统》实验报告 3

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/5/31

成绩： _____ **指导教师：** 罗飞

实验名称：Linux 内核编译实验	**实验地点**：信息楼 216
实验仪器：ECS 云计算环境（提供 Linux 实验环境）	

一、实验目的：

1、 了解 Linux 内核源代码的目录结构以及各目录的相关内容
2、 了解 Linux 内核各配置选项内容和作用
3、 掌握 Linux 内核配置文件 config.in 的作用
4、 掌握 Linux 内核的编译过程
5、 掌握将新增内核代码加入到 Linux 内核结构中的方法

二、实验内容：

完成实验指导手册实验，进而完成如下内容：

1.. 简述基于内核源码编译嵌入式 Linux 内核的步骤

1) 安装编译 toolchain(根据上次实验的内容，貌似隔一阵子都会清空环境，得重头开始编译...)

2) 使用命令 tar -xvf /home/ecust/samba_share/embed/Linux/linux-3.2.tar.bz2 解压内核源码

3) 进入 linux-3.2 目录，编辑 Makefile 文件，使用 /CONFIG_CROSS_COMPILE 命令查找，在第 195 行左右的位置更改目标板结构以及所使用的交叉编译器，修改结果如下:

ARCH ?= arm
CROSS_COMPILE ?= arm-none-linux-gnueabi-

4) 复制标准板配置文件到工作目录下

使用命令 cp arch/arm/configs/s5pv210_defconfig .config

5) 使用 make menuconfig 启动图形化配置界面，查看配置是否正确

一般能启动意味着配置不会有太大错误

6) 使用命令 make zImage 编译内核镜像

hostname 输入 zearo，其他选项为默认值

当前的 CPU 类型：

使用命令 lscpu 进行查看

当前的 CPU 类型为 x86_64

内核编译得到的内核文件有哪些？分别在什么目录？

本次实验使用 make zImage 编译后，得到 vmlinux 文件，在当前源代码目录下.

一般的，内核编译得到的内核文件通常包括以下几种：

1. zImage：这是一种压缩的内核镜像文件，适用于嵌入式系统或旧版本的 Linux 内核。
2. Image：这是未经压缩的内核镜像文件，通常用于较新版本的 Linux 内核 。
3. vmlinuz：这是一个指向压缩的内核镜像文件的符号链接。
4. System.map：这是内核符号表文件，包含了内核中所有符号的地址信息。
5. config：这是内核编译配置文件，记录了编译内核时使用的配置选项。

通常可以在内核源代码目录的"arch/<架构>/boot"目录 下找到这些文件。

三、完成思考题

1、 分析 make config、make menuconfig、make xconfig 三个 linux 内核配置界面的区别

1. make config:

- make config 是最基本的配置界面，以文本方式显示配置选项，用户需要逐个选择或取消选项来配置内核。
- make config 界面简单，适合在终端环境下使用，但对于大型的内核配置选项不够友好。

2. make menuconfig:

- make menuconfig 提供了一个基于文本的菜单界面，用户可以使用方向键和回车键来选择配置选项，具有一定的交互性。
- make menuconfig 相比 make config 更加直观和易用，可以方便地浏览和配置内核选项。

3. make xconfig:

- make xconfig 提供了一个基于图形界面的配置工具,用户可以通过鼠标和窗口操作来配置内核选项。
- make xconfig 界面更加直观和用户友好,适合对内核配置有较高要求或需要进行复杂配置的用户使用。

2、 指出 linux 内核编译命令 make, make zImage, make bzImage 的区别

1. make:

- 使用 make 命令编译 Linux 内核时，会生成未压缩的内核镜像文件 Image。
- 这种内核镜像文件适用于较新版本的 Linux 内核。

2. make zImage:

- 使用 make zImage 命令编译 Linux 内核时，会生成压缩的内核镜像文件 zImage。
- zImage 适用于嵌入式系统或旧版本的 Linux 内核，因为其体积较小。

3. make bzImage:

- 使用 make bzImage 命令编译 Linux 内核时，会生成压缩的内核镜像文件 bzImage。
- bzImage 适用于较新版本的 Linux 内核，与 zImage 相比，bzImage 支持更大的内核镜像大小。

《嵌入式系统》实验报告 4

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/6/20

成绩： _____ **指导教师：** 罗飞

实验名称：嵌入式文件系统的构造	**实验地点**：信息楼
实验仪器：ECS 云计算环境（提供 Linux 实验环境）	

一、实验目的：

1、 了解嵌入式操作系统中文件系统的类型和作用
2、 了解 JFFS2 文件系统的优点及其在嵌入式系统中的作用
3、 掌握利用 BusyBox 软件制作嵌入式文件系统的方法
4、 掌握嵌入式 Linux 文件系统的挂载过程

二、实验内容

注：如果 chrome 打不开，下载云平台上的 firefox 并安装，可以打开虚拟仿真实验。

1.简述利用 BusyBox 制作嵌入式文件系统的步骤。

获取 BusyBox 源码，使用命令

tar -xvf /home/ecust/samba_share/embedbusybox/busybox-1.17.3.tar.bz2 解压

进入源码目录，配置源码，使用 make menuconfig 命令配置。

选择 Busybox Settings 之下的 Build Options

勾选 Build BusyBox as a static binary (no shared libs)

将默认勾选的 Build with Large File Support (for accessing files > 2 GB)取消勾选

然后配置 Cross Compiler prefix 的值，设为 arm-none-linux-gnueabi-

使用命令 make 编译 BusyBox

然后使用命令 make install 安装 busybox， 默认安装在当前目录的 _install 下

进入_install 目录，使用 ls 查看文件夹，可以创建其他需要的目录

makdir dev etc mnt proc var tmp sys root

添加库，将工具链的 lib 拷贝到 _install 目录下

cp /home/ecust/workplace/toolchain/arm-none-linux-gnueabi/lib/ ./ -a

最后删除静态库以及各种不需要的库文件，确保最后的库大小不超过 4MB

可以使用 du -mh lib/查看大小

添加系统启动文件，在 etc 目录下创建文件 inittab

```
#this is run first except when booting in single-user mode.
:: sysinit:/etc/init.d/rcS
# /bin/sh invocations on selected ttys
#start an "askfirst" shell on the console (whatever that may be)
::askfirst:-/bin/sh
```

```
# stuff to do when restarting the init process
::restart:/sbin/init
# stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
```

在 etc 添加文件 fstab， vim fstab

```
#device        mount-point    type        options              dump        fsck order
proc            /proc          proc        defaults        0       0
tmpfs          /tmp           tmpfs       defaults          0             0
sysfs          /sys           sysfs       defaults          0             0
tmpfs    /dev           tmpfs       defaults          0             0
```

然后修改内核配置，添加 tmpfs 的支持， 在内核配置中使用 make menuconfig 配置
最后重新编译内核

三、思考：

1、 什么是根文件系统？查看当前所使用系统的文件系统目录与所制作的根文件系统目录，说明它们的异同；能否在建立根文件系统时中构建 sys、proc 子目录？请说明原因。

根文件系统是 Linux 内核启动后首先挂载的文件系统，它包含了系统启动时所必需的文件和目录。根文件系统提供了文件系统的顶层目录结构，是整个文件系统树的根目录。在嵌入式系统中，根文件系统通常存储在只读存储介质上，以确保系统的稳定性。

系统的文件系统目录有很多目录，home 等等，但是所制作的根文件系统目录没有这么多，只有一些最基本的。但是同时，两者都包含了最基本的目录。

可以在建立根文件系统时中构建 sys, proc 子目录，但通常它们会是空的或者只包含一些基本的初始化文件。因为 sys, proc 是特殊的伪文件系统，不占用实际的磁盘空间，而是在系统运行时由内核动态创建的。

《嵌入式系统》实验报告 5

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/6/14

成绩： _____ **指导教师：** 罗飞

实验名称：嵌入式驱动程序的构造及应用程序调试	**实验地点**： 信息楼

实验仪器：实验云平台

一、实验目的：

1．了解 Linux 驱动程序的结构
2．掌握 Linux 驱动程序常用结构体和操作函数的使用方法
3．初步掌握 Linux 驱动程序的编写方法及过程
4．掌握 gcc 和 gdb 工具进行编译和调试程序的使用方法

二、实验内容：

（一）驱动程序的构造

举例说明一个内核驱动模块程序的构造过程；特别地，加载该模块时，输出“Hello driver-module!”，卸载该模块时，输出“Goodbye driver-module!”。

编写模块文件 hello.c，内容如下

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>

MODULE_LICENSE(“Dual BSD/GPL”);

static int __init dri_arch_init_module(void){
    printk(“Hello driver-module!\r\n”);
    return 0;
}

static void __exit dri_arch_cleanup_module(void) {
        printk(“Goodbye driver-module!\r\n”);
}

module_init(dri_arch_init_module);
module_exit(dri_arch_cleanup_module);
```

将 linux 内核解压，配置，编译 (实验平台定时清空之前的操作，所以得重复实验三的内容)

PS: 实验平台的 linux 编译出来会显示缺少部分.h 文件。。。

解压缩 arm-cortex_a8-linux-gnueabi-命令

tar –xvf /home/ecust/samba_share/embed/Toolchain/arm-cortex_a8-linux-gnueabi.tar.bz2

打开~/.bashrc, vim ~/.bashrc，输入 GG 跳至最后一行，继续输入 o，新建一行，添加如下:

PATH=/home/ecust/Desktop/zearo/zrm-cortex_a8/bin:$PATH

其中/zearo/目录是解压缩后的目录

执行 source ~/.bashrc 命令应用

得解压缩 linux 内核到 samba_share/embed 中

编写 Makefile 文件，内容如下:

```
Obj-m := hello.o
KDIR:=/home/ecust/samba_share/embed/linux-3.2-FS210/
PWD := $(shell pwd)
default:
    make -C $(KDIR) M=$(PWD) modules ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabi-
clean:
    rm –rf *.o *.ko *.order *.mod.c *.symvers
```

这时候编译 make V=1

生成 hello.ko 文件 使用 insomd hello.ko 加载模块，使用 rmmod hello 卸载模块

所得到的驱动模块文件路径及其文件名为：

在当前工作目录下得到驱动模块文件，文件名为 hello.ko，以及相应的 Module.symvers, modules.order

进一步说明，驱动模块的基本结构包括那些内容？

模块加载函数 module_init()

模块卸载函数 module_exit()

模块许可证声明 MODULE_LICENSE()

模块作者以及描述信息说明 (可有可无)

以上为必要的基本结构

（二）应用程序的调试

1．构建工作目录并进入工作目录

2. 创建目标实验代码 greeting.c

实验代码内容如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static char buff [256];
static char* string;
```

```
int main (void) {
    char welcome[] = “Linux C Tools: GCC and GDB”;
    printf(“This is the experiment to learn %s \n!”, welcome);
    my_print();
}

void my_print()
{
    printf (“Please input a string: ”);
    gets (string);
    printf ("\nYour string is: %s\n", string);
}
```

3．创建 makefile

makefile 内容如下：

```
CC = gcc
CFLAGS = -Wall -g

greeting: greeting.o
    $(CC) $(CFLAGS) greeting.o –o greeting
greeting.o:   greeting.c:
    $(CC) $(CFLAGS) –c greeting.c –o greeting.o

clean:
    rm –f greeting.o
```

4. 基于上述 makefile 编译并运行该程序

程序运行错误，主要错误原因是：

segmentation fault

访问为初始化的指针， static char* string; 未初始化，不能使用 gets()访问

5. 使用 gdb 进行调试，结合 where, list, print, break, run, set 等调试命令进行调试，记录其过程。

使用命令 gdb ./greeting

使用命令 list ，查看代码(包括每一行)

可以发现 my_print()在第 11 行

9,10 行不可能有问题，所以直接使用 break 11 命令，标注第一个调试点

使用 run ，运行

可以发现运行到 my_print()这一行

使用 list – 继续查看代码

这时候继续增加调试点 显然 16 行打印 printf(“Please input a string:”);不可能有问题

break 17

break 18

使用命令 continue

输入完后可以明确发现第 18 行的问题 Segmentation fault.

发现问题后，退出 quit

三、思考：

1．驱动的加载使用主要有哪些方法？它们的差别是什么？

1. 静态加载：在操作系统启动时加载所有必要的驱动程序。这种方法的优点是系统启动速度快，但缺点是占用了系统资源，包括内存和处理器时间。

2. 动态加载：在系统运行时根据需要加载驱动程序。这种方法可以减少系统资源的占用，但可能会导致一些延迟，因为驱动程序需 要在需要时加载。

3. 模块化加载：将驱动程序分为多个模块，每个模块在需要时加载。这种方法结合了静态加载和动态加载的优点，可以灵活地管理 系统资源。

《嵌入式系统》实验报告 6

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/6/21

成绩： _____ **指导教师：** 罗飞

实验名称：嵌入式驱动程序的移植	**实验地点：** 信息楼
实验仪器：ECS 云计算环境（提供 Linux 实验环境）；开发板	

一、实验目的：

1．了解 Linux 驱动程序的结构
2．掌握 Linux 驱动程序常用结构体和操作函数的使用方法
3．初步掌握 Linux 驱动程序的移植方法及过程
4．掌握 Linux 驱动程序的加载方法

二、实验内容：

查看自己虚拟机 ip： 使用命令 ip addr 查看，可以发现虚拟机 ip 为 172.18.120.53

基于/home/ecust/samba_share/embed/Linux/linux-3.2.tar.bz2 来完成

1. 准备代码

使用命令 tar -jxvf /home/ecust/samba_share/embed/Linux/linux-3.2.tar.bz2 解压缩文件

使用 vim Makefile, 更改目标板结构以及所使用的交叉编译器

ARCH ?= arm

CROSS_COMPILE ?= arm-none-linux-gnueabi-

分别使用命令 curl -o regs-nand.h http://172.18.120.26:10005/experiments/experiment/downloadFileResource?fileId=4281, curl -o s3c_nand.c http://172.18.120.26:10005/experiments/experiment/downloadFileResource?fileId=4282 下载文件到 arch/arm/mach-s5pv210/include/mach/和 drivers/mtd/nand/中

由于实验平台直接下载文件会出现错误代码 9999，在这里采用以下方式解决:

新建用户 jiguang, sudo adduser jiguang

然后使用 window powershell, 使用命令 ssh jiguang@172.18.120.53 进入 linux 系统

然后使用命令下载文件，最后将文件移动到响应位置(跨用户)

这样在 window 终端下也可以复制代码

2. 修改平台代码：

简述主要步骤及关键代码含义。

在解压缩完的 linux 内核目录下,使用命令 vim arch/arm/mach-s5pv210/mach-smdkv210.c

添加相应头文件

```
#ifdef CONFIG_MTD_NAND
#include <linux/mtd/mtd.h>
```

#include <linux/mtd/nand.h>

#include <linux/mtd/nand_ecc.h>

#include <linux/mtd/partitions.h>

#include <plat/nand.h>

#endif

然后添加平台设备

关键代码含义:

定义了一个名为 s5pv210_partition_info 的结构体数组，它是 struct mtd_partition 类型的。这个数组用于定义 NAND 闪存的分区信息。每个分区都包含以下字段：

.name：分区的名称，用于在日志和配置中识别分区。

.offset：分区开始的位置，相对于 NAND 闪存的起始地址。

.size：分区的尺寸，以字节为单位。

平台设备结构体 s5pv210_device_nand，用于注册 NAND 控制器

添加 s5pv210-nand 的平台设备，并指定了设备 ID 为-1（这意味着它没有特定的 ID）。它还定义了一个资源数组 s5pv210_nand_resource，这个数组包含了 NAND 控制器的硬件资源信息，例如 I/O 端口地址和中断号。

最后将 s5pv210_device_nand 添加到平台设备数组 smdkv210_devices 中

static struct platform_device *smdkv210_devices[] __initdata = {

……

#if defined(CONFIG_MTD_NAND)

&s5pv210_device_nand,

#endif

};

这部分代码使用了条件编译（conditional compilation），这意味着如果在内核配置中启用了 CONFIG_MTD_NAND（即启用了 MTD（Memory Technology Device）子系统支持），那么 &s5pv210_device_nand 会被添加到数组中。如果没有启用 MTD 子系统，这段代码将被编译器忽略，&s5pv210_device_nand 不会被添加到数组中，从而避免了在内核中注册不必要的平台设备。

3. 修改arch/arm/plat-samsung/include/plat/nand.h

简述主要步骤：

在 nand.h 中添加结构体：

struct s3c_nand_mtd_info {

uint chip_nr;

uint mtd_part_nr;

struct mtd_partition *partition;

};

4. 修改include/linux/mtd/partitions.h

简述主要步骤：

在 partitions.h 中，可以在最后#endif 前添加函数

int add_mtd_partitions(struct mtd_info *, const struct mtd_partition *, int);

5. 修改arch/arm/mach-s5pv210/clock.c中结构体

简述主要步骤：

使用 vim clock.c(得提前进入该目录)打开文件，使用命令/static struct clk init_clocks_off[] 找到结构体，在结构体数组的最后添加一个结构体

```
{
        .name= "nandxl",
        .id= -1,
        .parent= &clk_hclk_psys.clk,
        .enable= s5pv210_clk_ip1_ctrl,
        .ctrlbit= (1 << 24),
},
```

6. 修改drivers/mtd/nand/Kconfig

简述主要步骤和关键代码含义：

进入目录后使用 vim Kconfig 打开文件，输入命令:213，在输入 o 新建一行，添加以下配置

```
config MTD_NAND_S3C
  tristate "NAND Flash support for S3C SoC"
  depends on MTD_NAND && (ARCH_S5PC1XX || ARCH_S5PC11X || ARCH_S5PV2XX || ARCH_S5PV210)
  help
    This enables the NAND flash controller on the S3C.
    No board specfic support is done by this driver, each board
    must advertise a platform_device for the driver to attach.

config MTD_NAND_S3C_DEBUG
  bool "S3C NAND driver debug"
  depends on MTD_NAND_S3C
  help
    Enable debugging of the S3C NAND driver

config MTD_NAND_S3C_HWECC
  bool "S3C NAND Hardware ECC"
  depends on MTD_NAND_S3C
  help
    Enable the use of the S3C's internal ECC generator when
    using NAND. Early versions of the chip have had problems with
    incorrect ECC generation, and if using these, the default of
    software ECC is preferable.
    If you lay down a device with the hardware ECC, then you will
    currently not be able to switch to software, as there is no
    implementation for ECC method used by the S3C
```

含义:

定义了三个内核配置选项，用于控制 S3C SoC（System on Chip）的 NAND 闪存支持

MTD_NAND_S3C：这个选项允许或禁用 S3C SoC 的 NAND 闪存控制器支持。它依赖于 MTD（Memory Technology Device）子系统支持，并且仅在 S3C 的特定架构（ARCH_S5PC1XX、ARCH_S5PC11X、ARCH_S5PV2XX、ARCH_S5PV210）上可用。启用这个选项会使得 S3C 的 NAND 控制器工作，但不会为特定板提供支持，每个板都需要提供平台设备以供驱动连接。

MTD_NAND_S3C_DEBUG：这个选项启用或禁用 S3C NAND 驱动的调试功能。它依赖于 MTD_NAND_S3C 的配置，即只有当 NAND 支持被启用时，调试选项才可用。启用这个选项可以用于调试 S3C NAND 驱动的问题。

MTD_NAND_S3C_HWECC：这个选项允许或禁用 S3C NAND 驱动使用硬件 ECC（Error Correction Code）生成器。它也依赖于 MTD_NAND_S3C 的配置。启用这个选项可以让 S3C 的内部 ECC 生成器处理 NAND 数据校验，但早期芯片版本存在 ECC 生成问题，使用软件 ECC 可能更安全。如果您使用硬件 ECC，则无法切换到软件 ECC，因为没有为 S3C 的 ECC 方法实现软件支持。

7. 修改drivers/mtd/nand/Makefile

简述主要步骤和关键代码含义：

进入到对应目录后使用命令 vim Makefile，/CONFIG_MTD_NAND_S3C2410，然后在下面添加

obj-$(CONFIG_MTD_NAND_S3C) += s3c_nand.o

当内核配置中启用了 S3C NAND 控制器支持时，s3c_nand.o 模块会被编译。

8. 将S3C NAND配置到内核中

简述主要步骤：

使用命令 make menuconfig 启动图形菜单配置界面

在 Device Drivers 使用回车键

在 Memory Technology Device (MTD) support, 使用空格勾选，然后使用回车键

在 NAND Device Support 使用空格勾选，然后使用回车键

勾选 Support for generic platform NAND driver 以及 Support for NAND Flash Simulator

9. 编译

使用命令 make zImage 编译，跟之前 linux 内核的编译类似

三、思考

1. 在内核中使用 S3C NAND 驱动有几种方式？分别如何操作？

两种方式，第一种，在编译内核前直接将 S3C NAND 驱动源码写入，之后再完整编译内核，即完成 S3C NAND 驱动的安装，

第二种，将 NAND 驱动视为模块动态加载入内核，

安装完驱动后需要挂载，挂载后即可使用。

任课教师签名：　　2024 年 6 月 21 日

《嵌入式系统》实验报告 7

学号： 21013122 **姓名：** 郑铠涛 **班级：** 计 213 **日期：** 2024/6/21

成绩： _____ **指导教师：** 罗飞

实验名称：Openwrt 编译/嵌入式系统的烧写实验	**实验地点**： 信息楼
实验仪器：Linux 实验环境	

一、实验目的：

1、 了解 Openwrt 源代码的目录结构以及各目录的相关内容
2、 了解 Openwrt 各配置选项内容和作用
3、 掌握 Openwrt 配置文件 feeds 的作用
4、 掌握 Openwrt 的编译过程
5、 掌握 Openwrt 的烧录过程
6、 掌握嵌入式 Linux 镜像的烧写过程

二、实验内容：

一>镜像编译

Openwrt 用到了 feeds，其作用是什么？如何组织？

OpenWrt 的 feeds 机制是一种扩展包管理器，它允许用户从 OpenWrt 的官方服务器或其他第三方源获取额外的软件包。这些软件包可以是系统工具、网络服务、应用程序等，它们通常不被包括在 OpenWrt 的核心源代码中，但是可以通过 feeds 机制方便地集成到 OpenWrt 系统中。

feeds 的组织方式通常是通过一个或多个 feeds 配置文件，这些文件位于 OpenWrt 源代码的 feeds.conf.default 或自定义的 feeds.conf 文件中。每个配置文件包含了一系列的 feeds 源地址，指向不同的软件包集合。

./scripts/feed update -a： 这个命令会更新所有的 feeds 源，同步最新的软件包信息。这个操作相当于运行 git pull 对于每个配置文件中指定的 feeds 源，确保你的本地源代码树中的 feeds 是最新的。

./scripts/feed install -a： 这个命令会安装所有之前通过 update 命令更新的软件包。安装实际上意味着将这些软件包的元数据添加到 OpenWrt 的构建系统中，使得它们在后续的编译过程中可以被包括在内。

所用目标板为 mt7688，如何进行配置的？（通过配置来验证目标板是 mt7688）

使用命令 make menuconfig

在这个配置界面中，导航到 Target System 选择 MediaTek Ralink MIPS，然后在 Subtarget 选择 MT7628，最后在 Target Profile 选择具体设备型号。

这时候可以看到目标板已经设置为 mt7688

特别的，编译完后，使用命令 cat /proc/cpuinfo，同样可以验证目标板型号。

特别地，通过 make menuconfig，说明以下配置别代表了什么，且其选项下的选项主要都有些什么作用？

Target System: 表示要构建固件的目标系统架构

Subtarget: 表示目标系统的具体子架构或类型，如 Generic、Raspberry Pi 等

Target Profile: 用于选择固件的配置文件，包含了预定义的硬件配置和软件包

Target Images: 选择要生成的固件镜像格式，如.img、.bin 等

Kernel Modules: 用于选择要包含在内核中的模块，以支持不同的硬件设备

Languages: 选择要包含在固件中的语言包

Luci: 选择是否包含 LuCI Web 界面管理工具

Network: 配置网络相关的选项，如无线网络、网络协议等

Sound: 配置声音相关的选项，如音频驱动、音频编解码器等