《嵌入式系统》实验报告 2

学号：21013134 姓名：徐昊博 班级：计 213 日期：2024.5.11

成绩：_____ 指导教师：罗飞

实验名称：嵌入式 Linux 开发环境	**实验地点**：
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

（1）创建一个新用户 zhangsan：*sudo useradd zhangsan*

（2）查看/etc/passwd 文件的最后一行内容：*tail -n 1 /etc/passwd* 内容如下：

zhangsan:x:1002:1002::/home/zhangsan:

（3）查看/etc/shadow 文件的最后一行内容：*sudo tail -n 1 /etc/shadow* 内容如下：

zhanqsan:!:19860:0:99999:7:::

（4）给新建的用户设置密码：*sudo passwd zhangsan*

（5）查看/etc/passwd 文件的最后一行内容：*tail -n 1 /etc/passwd* 内容如下：

zhangsan:6KfEgZ0h3$KNRNZEGCVIUSEIrUiUEUUe5mb3u2c7pTard.xh6gwTcnaF2NUfLZBPNGarNX48UvRb2Jbeqwg7akIw2.nlsi1:19860:0:99999:7::: 可以看到设置好的密码被加密存储到了文件 shadow 文件当中。

（6）使用 zhangsan 用户登录系统：*su - zhangsan*，遇到错误：No directory logging in with HOME=/ 这说明新建的 zhangsan 用户时无法找到其家目录所以无法登陆

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。

（1）使用 1 的步骤创建新用户：usersudo useradd *user*

（2）更改 zhangsan 所属群组为 root ： *sudo usermod -g root zhangsan*

（3）查看/etc/passwd 文件，记录 zhangsan 用户和 user 用户的属组情况：

cat /etc/passwd | grep zhangsan

cat /etc/passwd | grep user

记录内容如下：zhangsan 用户的属组情况：*zhangsan:x:1002:0::/home/zhangsan:*

user 用户的属组情况：hplip:x:114:7:HPLIP system user,,,:/var/run/hplip:/bin/false

user:x:1003:1003::/home/user:

（4）删除用户 user：*sudo userdel -r user*

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。

（1）创建一个新组，组名为 stuff：*sudo groupadd stuff*

（2）查看/etc/group 文件的最后一行内容，并记录：*tail -n 1/etc/group* 内容如下：

stuff:x:1004:

（3）创建一个新账户 test，并将其起始组合附属组都设置为 stuff：*sudo useradd -g stuff -G stuff test*

（4）查看/etc/group 文件中的最后一行内容，记录并说明变化：tail -n l /etc/group 内容如下：

stuff:x:1004:test，可见新的用户已经加入了组当中

（5）设置 stuff 组密码：*sudo gpasswd stuff*，输入指令后按照提示操作如下：

changing the password for group stuff

New Password:

Re-enter new password:

（6）在 stuff 组中删除用户 test：*sudo deluser test stuff*

（7）查看/etc/group 文件中的最后一行，记录并说明变化：*tail -n 1 /etc/group* 内容如下：

stuff:x:1004:，可以看到用户 test 已经被删除。

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。

（1）进入 vim 编辑器：vim file.c

（2）输入带有错误的 c 语言代码：

```
#include<stdio.h>
int main()
{
```

```
        print("Hello,World!\n");
        return 0        //缺少分号
}
```

按 esc 后输入：wq 退出编辑模式

（2）运行 *gcc file.c -o myfile* 出现以下错误信息

```
file.c: In function ‘main’:
file.c:5:17: error: expected ‘;’ before ‘}’ token
    5 |         return 0
      |                 ^
      |                 ;
    6 | }
      | ~
```

这表明在 main 函数当中 return0 语句后缺少一个分号

（3）重新进入 vim 编辑器，对该文件进行修改，更正 return 0 后面的分号。然后存盘，退出 vi 编辑器。重新编译该文件，编译通过，利用./myfile 运行该程序，得到结果：在终端上显示 Hello，World！

（4）说明以上各个命令的作用。

删除某些文本行：

x：删除当前光标所在的字符。

dd：删除当前光标所在的整行。

复原删除操作：

u：撤销前一个操作。

修改文本内容：

c：进入更改模式。例如 cw 可以更改当前单词。

r：替换当前字符。例如 rX 将当前字符替换为 X。

s：删除当前字符并进入插入模式。

检索给定模式：

/pattern：向前搜索 pattern。

?pattern：向后搜索 pattern。

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）。

建立工作目录：

```
ecust@Ubuntu:~$ mkdir   /home/ecust/workplace
ecust@Ubuntu:~$ cd   /home/ecust/workplace
```

ToolChain 安装：

```
ecust@Ubuntu:~/workplace$ tar jxvf /home/ecust/samba_share/embed/ToolChain/toolchain-4.5.1-farsight.tar.bz2
ecust@Ubuntu:~/workplace$ ln -s toolchain-4.5.1-farsight toolchain
ecust@Ubuntu:~/workplace$ vim ~/.bashrc
```

在该文件的末尾输入：

```
export PATH=/home/ecust/workplace/toolchain/bin:$PATH
```

保存退出

source 命令的作用：

source 命令用于在当前 Shell 环境中读取并执行文件中的命令。它通常用于重新加载 Shell 配置文件（如.bashrc、.bash_profile 或.zshrc）以使更改立即生效，而不需要重新启动终端。具体作用如下：

立即应用对配置文件的更改。

在当前 Shell 会话中执行脚本，而不创建新的子 Shell。

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。

（1）X86 架构：

1. 编辑 hello.c

使用 vim 或其他文本编辑器创建并编辑 hello.c 文件：*vim hello.c*

输入以下 C 语言代码并保存：

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

2. 编译 hello.c

使用 gcc 编译 hello.c 文件：

gcc hello.c -o hello_x86

3. 查看可执行文件格式

使用 file 命令查看生成的可执行文件格式：*file hello_x86*

hello_x86: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=e5c1c5d73c5aef963245fceeb3b63e1f25b95b0c, not stripped

（2）ARM 架构

1. 编辑 hello.c

可以使用与 x86 架构相同的代码：

vim hello.c

输入以下 C 语言代码并保存：

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

2. 编译 hello.c

使用交叉编译工具链 arm-linux-gnueabi-gcc）编译 hello.c 文件：*arm-linux-gnueabi-gcc hello.c -o hello_arm*

3. 查看可执行文件格式

使用 file 命令查看生成的可执行文件格式：*file hello_arm*

预期输出示例：

hello_arm: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, not stripped

（3）两个架构区别：

1. 架构差异：

X86 架构：一般是指 Intel 和 AMD 等公司生产的 CPU，使用的是 CISC（复杂指令集计算机）架构，文件输出为‘x86-64’或‘i386’。

ARM 架构：一般用于移动设备和嵌入式系统，使用的是 RISC（精简指令集计算机）架构，文件输出为‘ARM’。

2. ELF 格式：

X86 架构：可执行文件为 ELF 64-bit（或 32-bit）格式，适用于‘x86-64’架构。

ARM 架构：可执行文件为 ELF 32-bit 格式，适用于‘ARM’架构。

3. 动态链接与静态链接：

X86 架构：通常是动态链接，文件输出中会有‘dynamically linked’标识。

ARM 架构：在嵌入式系统中可能更多使用静态链接，文件输出中会有‘statically linked’标识。

4. 解释器路径：

X86 架构：解释器路径为‘/lib64/ld-linux-x86-64.so.2’。

ARM 架构：没有特定解释器路径（因为可能是静态链接）。

7．bootLoader 程序的编译

说明编译 Bootloader 程序的步骤：

（1）获取源代码

首先，从官方仓库或其他可信来源获取 Bootloader 的源代码。以 U-Boot 为例：

git clone https://gitlab.denx.de/u-boot/u-boot.git

cd u-boot

（2）安装必要的依赖

确保系统中安装了必要的工具和依赖包，包括编译器、工具链和其他开发工具。以 ARM 平台为例，可能需要安装交叉编译工具链：

sudo apt-get update

sudo apt-get install gcc-arm-none-eabi

（3）配置 U-Boot

为目标平台配置 U-Boot。U-Boot 源代码中包含许多配置选项，针对不同的硬件平台进行配置。对于特定板子（如 raspberrypi_4）配置如下：

make rpi_4_defconfig

这将生成默认配置文件，位于 configs 目录下。

（4）编译 U-Boot

使用 make 命令编译 U-Boot。这一步会生成二进制文件和相关的目标文件。

编译完成后，将在 u-boot 目录下看到生成的二进制文件，如 u-boot.bin 或 u-boot.img。

（5）验证和调试

将 SD 卡插入目标板中，启动设备，验证 U-Boot 是否正常工作。如果出现问题，可以通过串行控制台或其他调试工具进行调试。

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

‘include/configs/fs210_nand.h’和‘include/configs/fs210_sd.h’是两个不同的头文件，用于配置不同的引导方式和存储介质。

1. ‘fs210_nand.h’是针对 NAND Flash 存储器的引导配置文件。NAND Flash 是一种非易失性存储器，通常用于嵌入式系统中作为引导介质。该头文件中可能包含了 NAND Flash 相关的配置参数，如引导加载地址、内存布局等信息。

2.‘fs210_sd.h’是针对 SD 卡存储器的引导配置文件。SD 卡是一种常见的可移动存储介质，也可以用于嵌入式系统的引导。该头文件中可能包含了 SD 卡相关的配置参数，如引导加载地址、内存布局等信息。

这两个文件的区别主要在于它们针对的存储介质不同，分别是 NAND Flash 和 SD 卡。在实际应用中，根据系统的需求和硬件平台的特点选择适合的引导配置文件。

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

（1）准备硬件和软件环境

目标板：确保目标板已经连接好，并且可以通过串口、JTAG 或其他接口与主机通信。

编译好的 Bootloader 文件：如 u-boot.bin 或 u-boot.img。

串口通信软件：如 minicom、putty 或 screen。

下载工具：如 dfu-util、fastboot、tftp 或其他特定平台的工具。

（2）连接目标板

连接串口：使用串口线连接目标板和主机，并启动串口通信软件。

sudo minicom -D /dev/ttyUSB0 -b 115200

连接电源：为目标板接通电源。

（3）选择下载方法

根据目标板支持的下载方法，选择合适的下载工具这里使用 U-Boot 和 TFTP：

1、启动 U-Boot：在串口通信软件中启动目标板，进入 U-Boot 命令行界面。

2、设置网络环境：在 U-Boot 命令行中设置网络参数，如 IP 地址、服务器 IP 地址等。

setenv ipaddr 192.168.1.100

setenv serverip 192.168.1.1

3、使用 TFTP 下载 Bootloader：

tftp 0x82000000 u-boot.bin

4、将 Bootloader 写入 Flash：

nand erase 0 0x40000

nand write 0x82000000 0 0x40000

（4）验证 Bootloader

重启目标板：重新启动目标板，观察串口输出，验证 Bootloader 是否正常运行。

检查输出：确保看到 Bootloader 启动信息，确认其正常加载和执行。