

学生实验报告

成绩:

- **管理硬件资源：** 内核控制并分配计算机的硬件资源，例如 CPU、内存、I/O 设备等。它确保每个应用程序都能公平地访问这些资源，并防止它们相互冲突。
- **提供系统调用接口：** 应用程序通过系统调用与内核交互。系统调用是应用程序请求内核执行某些操作的函数，例如打开文件、读取数据或创建进程。
- **处理中断：** 当硬件或软件发生需要内核注意的事件时，就会产生中断。内核负责

处理中断并相应地采取措施。

- 调度进程： 当多个进程同时运行时，内核负责决定哪个进程应该获得 CPU 时间。这称为进程调度。
- 管理内存： 内核负责管理计算机的内存。它分配和释放内存给应用程序，并确保每个应用程序都能安全地访问其内存。
- 提供文件系统支持： 内核支持各种文件系统，例如 ext4、Btrfs 和 ZFS。它允许应用程序以统一的方式访问这些文件系统上的文件。
- 实现网络功能： 内核提供网络功能，例如 TCP/IP 协议栈。这允许计算机相互通信并访问网络资源。

Linux 内核架构：

- Linux 内核采用模块化设计，这意味着它由许多独立的模块组成。每个模块都负责特定的功能，例如管理特定类型的硬件设备或提供特定的系统服务。这种模块化设计使内核易于扩展和维护。

我们本次实验使用 linux-3.5 内核，我们将使用该内核的源码进行内核编译学习，然后使用内核中的内置驱动，进行设备驱动学习。

2. 什么是 makefile？为什么需要 makefile？

什么是 makefile？或许很多 Windows 的程序员都不知道这个东西，因为那些 Windows 的 IDE 都为你做了这个工作，但我觉得要作一个好的和 professional 的程序员，makefile 还是要懂。这就好像现在有这么多的 HTML 的编辑器，但如果想成为一个专业人士，还是要了解 HTML 的标识的含义。特别在 Unix 下的软件编译，你就不能不自己写 makefile 了，会不会写 makefile，从一个侧面说明了一个人是否具备完成大型工程的能力。

因为，makefile 关系到了整个工程的编译规则。一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile 就像一个 Shell 脚本一样，其中也可以执行操作系统的命令。

makefile 带来的好处就是——“自动化编译”，一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。make 是一个命令工具，是一个解释 makefile 中指令的命令工具，一般来说，大多数的 IDE 都有这个命令，比如：Delphi 的 make，Visual C++ 的 nmake，Linux 下 GNU 的 make。可见，makefile 都成为了一种在工程方面的编译方法。

3. 什么是驱动程序？

Linux 内核将设备驱动分为三大类：

- 字符设备驱动
- 块设备驱动
- 网络设备驱动

(1) 字符设备驱动

字符设备驱动用于管理和控制字符设备，即以字节流方式进行读写操作的设备。常见的字符设备包括串口、键盘、鼠标、打印机等。

字符设备驱动的特点：

- 按字节流进行读写操作
- 通常用于与用户进行交互的设备
- 驱动程序使用 file_operations 结构体实现文件操作函数

字符设备驱动的典型应用场景：

- 串口驱动程序：用于管理和控制串口设备，实现串口通信
- 键盘驱动程序：用于管理和控制键盘设备，处理键盘输入事件
- 鼠标驱动程序：用于管理和控制鼠标设备，处理鼠标移动和按键事件
- 打印机驱动程序：用于管理和控制打印机设备，实现打印操作

(2) 块设备驱动

块设备驱动用于管理和控制块设备，即以块为单位进行读写操作的设备。常见的块设备包括硬盘、固态硬盘、光盘驱动器等。

块设备驱动的特点：

- 按块进行读写操作
- 通常用于存储设备
- 驱动程序使用 `block_device_operations` 结构体实现块设备操作函数

块设备驱动的典型应用场景：

- 硬盘驱动程序：用于管理和控制硬盘设备，实现文件读写、分区管理操作
- 固态硬盘驱动程序：用于管理和控制固态硬盘设备，实现文件读写、分区管理等操作
- 光盘驱动器驱动程序：用于管理和控制光盘驱动器设备，实现光盘读写、刻录等操作

(3) 网络设备驱动

网络设备驱动用于管理和控制网络设备，例如网卡、无线网卡等。网络设备驱动使计算机能够与网络进行通信。

网络设备驱动的特点：

- 发送和接收网络数据包
- 实现网络协议栈
- 通常使用 `net_device_ops` 结构体实现网络设备操作函数

网络设备驱动的典型应用场景：

- 以太网驱动程序：用于管理和控制以太网网卡，实现以太网通信
- Wi-Fi 驱动程序：用于管理和控制无线网卡，实现无线网络连接
- 蓝牙驱动程序：用于管理和控制蓝牙设备，实现蓝牙通信

Linux 设备驱动是操作系统与硬件设备之间通信的桥梁，在现代操作系统中扮演着至关重要的角色。

如果想要驱动程序加载到内核运行环境中，那么就要使用这两种方法其中之一：**编译驱动程序到内核中**和**编译成模块并挂载到内核**，但两者之间存在一些关键的区别：

特性	编译到内核中	编译成模块
加载方式	自动加载	手动加载
灵活性	较低	较高
安全性	较低	较高
大小	较大	较小

而本次实验将要开发简单的字符设备驱动程序，也就是第一类驱动程序，并且将使用编译成模块并挂在到内核中的方式，加载到内核运行环境中。

实验内容：

1. 准备工作

- (1) 连接好开发板，将开发板串口与主机串口相连
- (2) 打开终端，激活 sudo 权限

```
oseasy@PC05:~$ sudo su
[sudo] oseasy 的密码:
root@PC05:/home/oseasy# sudo tar -xjvf ./arm-linux-gcc-4.6.4-arm-x86_64.tar.bz2
-C /
```

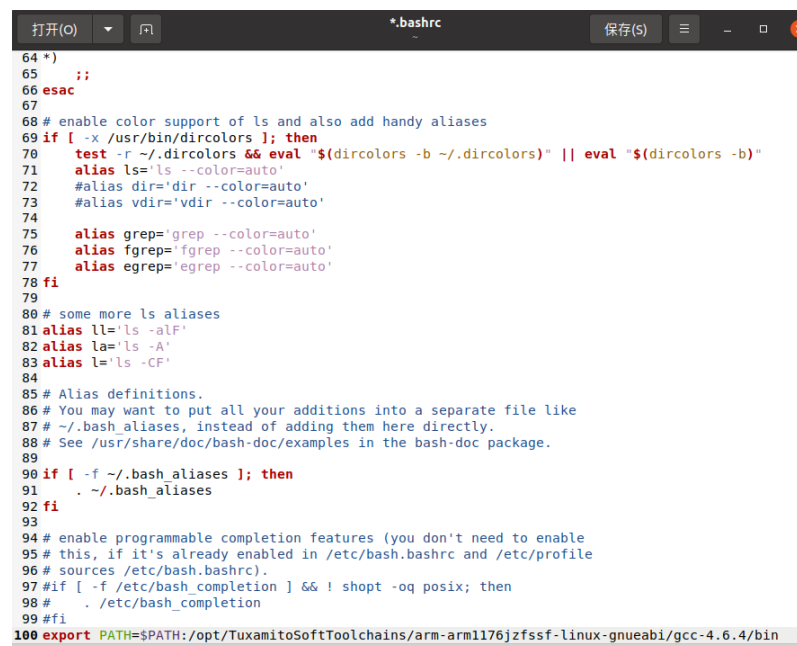
2. 配置交叉编译环境

- (1) 使用 tar arm-linux-gcc-4.5.1-v6-vfp-20120301.tar -C /命令将其解压到/opt 路径下；
- (2) 将 arm-linux-gcc 加入环境变量中，使用命令 gedit ~/.bashrc，打开编辑模型；

```
root@PC05:/home/oseasy# sudo tar -xjvf ./arm-linux-gcc-4.6.4-arm-x86_64.tar.bz2
-C /
opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/
opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin/
opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin/arm-non
e-linux-gnueabi-elfedit
opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin/arm-non
e-linux-gnueabi-addr2line
opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin/arm-arm
```

```
root@PC05:/home/oseasy# gedit ~/.bashrc
(gedit:11132): Tepl-WARNING **: 19:32:50.823: GVfs metadata is not supported. Fa
llback to TeplMetadataManager. Either GVfs is not correctly installed or GVfs me
tadata are not supported on this platform. In the latter case, you should config
ure Tepl with --disable-gvfs-metadata.
root@PC05:/home/oseasy#
```

- (3) 在文件末尾追加 PATH=\$PATH:/opt/FriendlyARM/toolchain/4.5.1/bin，即加入 PATH 环境中；



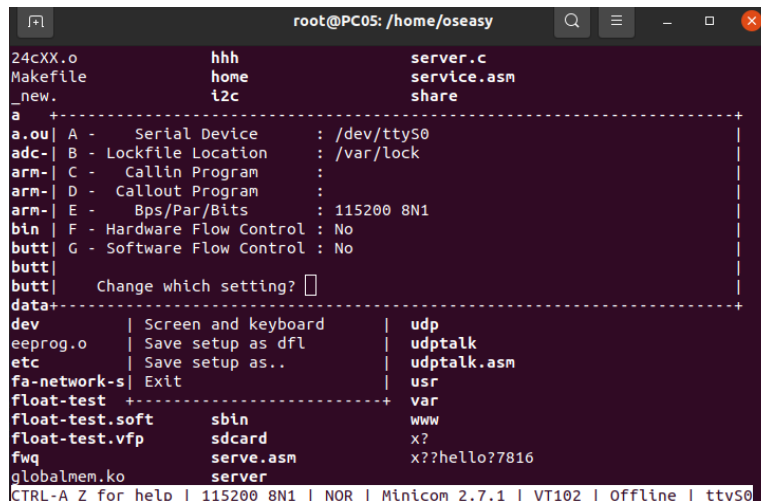
```
*.bashrc
64 *)
65 ;;
66 esac
67
68 # enable color support of ls and also add handy aliases
69 if [ -x /usr/bin/dircolors ]; then
70     test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
71     alias ls='ls --color=auto'
72     #alias dir='dir --color=auto'
73     #alias vdir='vdir --color=auto'
74
75     alias grep='grep --color=auto'
76     alias fgrep='fgrep --color=auto'
77     alias egrep='egrep --color=auto'
78 fi
79
80 # some more ls aliases
81 alias ll='ls -alF'
82 alias la='ls -A'
83 alias l='ls -CF'
84
85 # Alias definitions.
86 # You may want to put all your additions into a separate file like
87 # ~/.bash_aliases, instead of adding them here directly.
88 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
89
90 if [ -f ~/.bash_aliases ]; then
91     . ~/.bash_aliases
92 fi
93
94 # enable programmable completion features (you don't need to enable
95 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
96 # sources /etc/bash.bashrc).
97 if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
98     . /etc/bash_completion
99 fi
100 export PATH=$PATH:/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin
```

- (4) 使用 `source ~/.bashrc` 更新环境变量, `echo $PATH` 显示编译器逻辑则正确;

```
root@PC05:/home/oseasy# arm-linux-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gcc
COLLECT_LTO_WRAPPER=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/
gcc-4.6.4/libexec/gcc/arm-arm1176jzfssf-linux-gnueabi/4.6.4/lto-wrapper
Target: arm-arm1176jzfssf-linux-gnueabi
Configured with: /work/builddir/src/gcc-4.6.4/configure --build=x86_64-build_unk
nown-linux-gnu --host=x86_64-build_unknown-linux-gnu --target=arm-arm1176jzfssf-
linux-gnueabi --prefix=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnuea
bi/gcc-4.6.4 --with-sysroot=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-
gnueabi/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/sysroot --enable-languages=c,c
++ --with-arch=armv6zk --with-cpu=arm1176jzf-s --with-tune=arm1176jzf-s --with-f
pu=vfp --with-float=softfp --with-pkgversion='crosstool-NG hg:unknown-20130521.1
54019 - tc0002' --disable-sjlj-exceptions --enable-__cxa_atexit --disable-libmud
flap --disable-libgomp --disable-libssp --disable-libquadmath --disable-libquadm
ath-support --with-gmp=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-mpfr=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-mp
c=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-ppc=/work/bui
lddir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-cloog=/work/builddir/arm
-arm1176jzfssf-linux-gnueabi/buildtools --with-libelf=/work/builddir/arm-arm1176
jzfssf-linux-gnueabi/buildtools --with-host-libstdcxx='-static-libgcc -Wl,-Bstat
ic,-lstdc++,-Bdynamic -lm' --enable-threads=posix --enable-target-optspace --wit
```

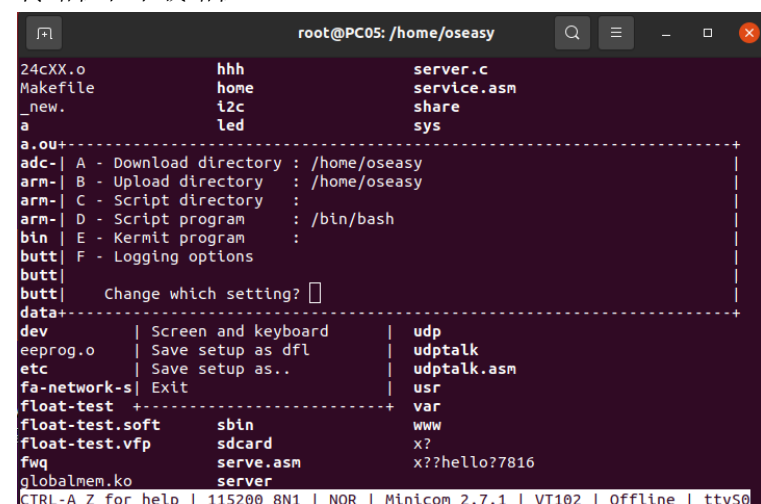
3. 设置 minicom 的配置

- (1) 修改 serial device 和波特率, 关闭硬件控制流



```
root@PC05:/home/oseasy
24cXX.o      hhh      server.c
Makefile     home     service.asm
_new.        i2c      share
a            led
a.ou+-----+
adc-| A - Serial Device      : /dev/ttyS0
arm-| B - Lockfile Location  : /var/lock
arm-| C - Callin Program    :
arm-| D - Callout Program   :
bin  | E - Bps/Par/Bits     : 115200 8N1
butt | F - Hardware Flow Control : No
butt | G - Software Flow Control : No
butt |
butt | Change which setting? [ ]
data+-----+
dev   | Screen and keyboard | udp
eeprog.o | Save setup as dfl      | udptalk
etc    | Save setup as..       | udptalk.asm
fa-network-s| Exit                  | usr
float-test +-----+ var
float-test.soft  sbin      www
float-test.vfp   sdcard    x?
fwq             serve.asm  x??hello?7816
globalmem.ko     server
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0
```

- (2) 配置上传路径和下载路径



```
root@PC05:/home/oseasy
24cXX.o      hhh      server.c
Makefile     home     service.asm
_new.        i2c      share
a            led
a.ou+-----+
adc-| A - Download directory : /home/oseasy
arm-| B - Upload directory  : /home/oseasy
arm-| C - Script directory  :
arm-| D - Script program    : /bin/bash
bin  | E - Kermit program   :
butt | F - Logging options   :
butt |
butt | Change which setting? [ ]
data+-----+
dev   | Screen and keyboard | udp
eeprog.o | Save setup as dfl   | udptalk
etc    | Save setup as..    | udptalk.asm
fa-network-s| Exit              | usr
float-test +-----+ var
float-test.soft  sbin      www
float-test.vfp   sdcard    x?
fwq             serve.asm  x??hello?7816
globalmem.ko     server
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0
```

- (3) 重启开发板, 更新配置, 检测是否连接成功

```
root@PC05: /home/oseasy
press CTRL-A Z for help on special keys

root@FriendlyARM /]# ls
??@?          fwq          server
              globalmem.ko server.asm
4cXX.o        hello        server.c
makefile      hhh          service.asm
CPserve       home         share
DP            i2c          sys
new.          led          term
              led-player  test
              led1     test.sh
dc-test       lib          test.sh-
rm-udp        linuxrc     th
rm-udptalk    lost+found  tiny4412_buttons.ko
rm-udptalkzk  mnt         tiny4412_leds.ko
in            opt         tiny4412_leds.mod.c
button        p           tiny4412_pwm.ko
button_test   proc        tmp
buttons_test  pthread     udisk
ata           pthread.o   udp
ev            pwm         udptalk
eprog.o       pwm_test    udptalk.asm
```

实验内容 1-globalmem 驱动

4. 编译开发板的 Linux 内核

使用以下命令解压内核的压缩包

```
#tar -zxvf linux-3.5-20150929.tgz
```

复制 tiny4412_linux_deconfig 为.config

```
#cp tiny4412_linux_deconfig .config
```

执行 Makefile 文件，编译内核

```
# make ARCH=arm CROSS_COMPILE=arm-linux-
```

```
oseasy@PC05:~/下载/linux-3.5$ sudo su
[sudo] oseasy 的密码:
root@PC05:/home/oseasy/下载/linux-3.5# make ARCH=arm CROSS_COMPILE=arm-linux-
CHK      include/linux/version.h
CHK      include/generated/utsrelease.h
make[1]: "include/generated/mach-types.h"已是最新。
CC      kernel/bounds.s
GEN      include/generated/bounds.h
CC      arch/arm/kernel/asm-offsets.s
GEN      include/generated/asm-offsets.h
CALL     scripts/checksyscalls.sh
CC      scripts/mod/empty.o
HOSTCC   scripts/mod/mk_elfconfig
MKELF    scripts/mod/elfconfig.h
HOSTCC   scripts/mod/file2alias.o
```

等待约 10 分钟:

```
arch/arm/boot/compressed/head.S:1068: Warning: (null)
arch/arm/boot/compressed/head.S:1075: Warning: (null)
arch/arm/boot/compressed/head.S:1107: Warning: (null)
GZIP     arch/arm/boot/compressed/piggy.gzip
AS       arch/arm/boot/compressed/piggy.gzip.o
CC       arch/arm/boot/compressed/misc.o
CC       arch/arm/boot/compressed/decompress.o
CC       arch/arm/boot/compressed/string.o
SHIPPED  arch/arm/boot/compressed/lib1funcs.S
AS       arch/arm/boot/compressed/lib1funcs.o
SHIPPED  arch/arm/boot/compressed/ashldi3.S
AS       arch/arm/boot/compressed/ashldi3.o
LD       arch/arm/boot/compressed/vmlinux
OBJCOPY  arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
Building modules, stage 2.
MODPOST 3 modules
CC       crypto/ansi_cprng.mod.o
LD [M]   crypto/ansi_cprng.ko
CC       drivers/char/tiny4412_hello_module.mod.o
LD [M]   drivers/char/tiny4412_hello_module.ko
CC       drivers/scsi/scst_wait_scan.mod.o
LD [M]   drivers/scsi/scst_wait_scan.ko
root@PC05:/home/oseasy/下载/linux-3.5# [
```

(截图显示编译成功)

5. 驱动程序编写

完善相关代码，globalmem.c 源程序如下：

```
1. #include <linux/module.h> // 包含模块的头文件
2. #include <linux/types.h> // 包含类型定义的头文件
3. #include <linux/fs.h> // 包含文件系统的头文件
4. #include <linux/errno.h> // 包含错误码的头文件
5. #include <linux/mm.h> // 包含内存管理的头文件
6. #include <linux/sched.h> // 包含调度的头文件
7. #include <linux/init.h> // 包含初始化宏的头文件
8. #include <linux/cdev.h> // 包含字符设备的头文件
9. #include <asm/io.h> // 包含 I/O 操作的头文件
10. #include <linux/slab.h> // 包含内存分配的头文件
11. #include <linux/version.h> // 包含 Linux 版本的头文件
12.
13. #if LINUX_VERSION_CODE > KERNEL_VERSION(3, 3, 0)
14. #include <asm/switch_to.h> // 包含特定版本的头文件
15. #else
16. #include <asm/system.h> // 包含特定版本的头文件
17. #endif
18. #include <asm/uaccess.h> // 包含用户空间访问的头文件
19. #define GLOBALMEM_SIZE 0x1000 /*全局内存最大 4K 字节*/
20. #define MEM_CLEAR 0x1 /*清 0 全局内存*/
21. #define GLOBALMEM_MAJOR 150 /*预设的 globalmem 的主设备 \号*/
22. static int globalmem_major = GLOBALMEM_MAJOR; // 定义设备号
23.
24. /*globalmem 设备结构体*/
25. struct globalmem_dev
26. {
27.     struct cdev cdev; /*cdev 结构体*/
28.     unsigned char mem[GLOBALMEM_SIZE]; /*全局内存*/
29. };
30.
31. struct globalmem_dev *globalmem_devp; /*设备结构体指针*/
32.
33. // 打开设备的函数
34. int globalmem_open(struct inode *inode, struct file *filp)
35. {
36.     filp->private_data = globalmem_devp;
37.     return 0;
38. }
```



```

39.
40.// 释放设备的函数
41.int globalmem_release(struct inode *inode, struct file *fil
    p)
42.{
43.    return 0;
44.}
45.
46.// 设备的ioctl 函数
47.long globalmem_ioctl(struct file *filp, unsigned int cmd, u
    nsigned long arg)
48.{
49.    struct globalmem_dev *dev = filp->private_data;
50.    /*
51.    获得设备结构体指针
52.    */
53.    switch (cmd)
54.    {
55.    case MEM_CLEAR:
56.        memset(dev->mem, 0, GLOBALMEM_SIZE); // 清零全局内存
57.        printk(KERN_INFO "globalmem is set to zero\n");
58.        break;
59.    default:
60.        return -EINVAL;
61.    }
62.    return 0;
63.}
64.
65.// 设备的read 函数
66.static ssize_t globalmem_read(struct file *filp, char __use
    r *buf, size_t size, loff_t *ppos)
67.{
68.    unsigned long p = *ppos;
69.    unsigned int count = size;
70.    int ret = 0;
71.    struct globalmem_dev *dev = filp->private_data; /*获得设
    备结构体指针*/
72.
73.    /* 分析和获取有效的写长度*/
74.    if (p >= GLOBALMEM_SIZE)
75.        return count ? -ENXIO : 0;
76.    if (count > GLOBALMEM_SIZE - p)
77.        count = GLOBALMEM_SIZE - p;
78.

```



```

79.     /*内核空间->用户空间*/
80.     if (copy_to_user(buf, (void *) (dev->mem + p), count))
81.     {
82.         ret = -EFAULT;
83.     }
84.     else
85.     {
86.         *ppos += count;
87.         ret = count;
88.         printk(KERN_INFO "read %d bytes(s) from %d\n",
            count, p);
89.     }
90.     return ret;
91. }
92.
93. // 设备的write 函数
94. static ssize_t globalmem_write(struct file *filp, const cha
    r __user *buf, size_t size, loff_t *ppos)
95. {
96.     unsigned long p = *ppos;
97.     unsigned int count = size;
98.     int ret = 0;
99.     struct globalmem_dev *dev = filp->private_data; /*获得设
        备结构体指针*/
100.
101.     /*分析和获取有效的写长度*/
102.     if (p >= GLOBALMEM_SIZE)
103.         return count ? -ENXIO : 0;
104.     if (count > GLOBALMEM_SIZE - p)
105.         count = GLOBALMEM_SIZE - p;
106.
107.     /*用户空间->内核空间*/
108.     if (copy_from_user(dev->mem + p, buf, count))
109.         ret = -EFAULT;
110.     else
111.     {
112.         *ppos += count;
113.         ret = count;
114.         printk(KERN_INFO "written %d bytes(s) from %d
            \n", count, p);
115.     }
116.     return ret;
117. }
118.

```

```
119. // 设备的llseek 函数
120. static loff_t globalmem_llseek(struct file *filp, loff_t
    offset, int orig)
121. {
122.     loff_t ret = 0;
123.     switch (orig)
124.     {
125.     case 0: /*相对文件开始位置偏移*/
126.         if (offset < 0)
127.         {
128.             ret = -EINVAL;
129.             break;
130.         }
131.         if ((unsigned int)offset > GLOBALMEM_SIZE)
132.         {
133.             ret = -EINVAL;
134.             break;
135.         }
136.         filp->f_pos = (unsigned int)offset;
137.         ret = filp->f_pos;
138.         break;
139.     case 1: /*相对文件当前位置偏移*/
140.         if ((filp->f_pos + offset) > GLOBALMEM_SIZE)
141.         {
142.             ret = -EINVAL;
143.             break;
144.         }
145.         if ((filp->f_pos + offset) < 0)
146.         {
147.             ret = -EINVAL;
148.             break;
149.         }
150.         filp->f_pos += offset;
151.         ret = filp->f_pos;
152.         break;
153.     default:
154.         ret = -EINVAL;
155.         break;
156.     }
157.     return ret;
158. }
159.
160. // 定义设备的操作函数
161. static const struct file_operations globalmem_fops =
```

```
162.     {
163.         .owner = THIS_MODULE,
164.         .llseek = globalmem_llseek,
165.         .read = globalmem_read,
166.         .write = globalmem_write,
167.         .unlocked_ioctl = globalmem_ioctl,
168.         .open = globalmem_open,
169.         .release = globalmem_release,
170.
171.     };
172.
173. // 初始化字符设备结构体，并将设备添加到系统中
174. static void globalmem_setup_cdev(struct globalmem_dev *dev, int index)
175. {
176.     int err, devno = MKDEV(globalmem_major, index);
177.
178.     cdev_init(&dev->cdev, &globalmem_fops);
179.     dev->cdev.owner = THIS_MODULE;
180.     dev->cdev.ops = &globalmem_fops;
181.     err = cdev_add(&dev->cdev, devno, 1);
182.     if (err)
183.         printk(KERN_NOTICE "Error %d adding LED%d", err, index);
184. }
185.
186. // 驱动程序的初始化函数
187. int globalmem_init(void)
188. {
189.     int result;
190.     dev_t devno = MKDEV(globalmem_major, 0);
191.
192.     /* 申请设备号*/
193.     if (globalmem_major)
194.         result = register_chrdev_region(devno, 1, "globalmem");
195.     else /* 动态申请设备号 */
196.     {
197.         result = alloc_chrdev_region(&devno, 0, 1, "globalmem");
198.         globalmem_major = MAJOR(devno);
199.     }
200.     if (result < 0)
201.         return result;
```

```

202.
203.     /* 动态申请设备结构体的内存*/
204.     globalmem_devp = kzalloc(sizeof(struct globalmem_dev)
    , GFP_KERNEL);
205.     if (!globalmem_devp) /*申请失败*/
206.     {
207.         result = -ENOMEM;
208.         goto fail_malloc;
209.     }
210.     memset(globalmem_devp, 0, sizeof(struct globalmem_dev
    ));
211.     globalmem_setup_cdev(globalmem_devp, 0);
212.     return 0;
213. fail_malloc:
214.     unregister_chrdev_region(devno, 1);
215.     return result;
216. }
217.
218. // 驱动程序的退出函数
219. void globalmem_exit(void)
220. {
221.     cdev_del(&globalmem_devp->cdev);
    /*注销 cdev*/
222.     kfree(globalmem_devp);
    /*释放设备结构体内存*/
223.     unregister_chrdev_region(MKDEV(globalmem_major, 0), 1
    ); /*释放设备号*/
224. }
225.
226. MODULE_AUTHOR("Song Baohua");           // 模块作者
227. MODULE_LICENSE("Dual BSD/GPL");         // 模块许可证
228. module_param(globalmem_major, int, S_IRUGO); // 模块参数
229. module_init(globalmem_init);             // 模块初始化
    函数
230. module_exit(globalmem_exit);             // 模块退出函
    数

```

6. Makefile 编写, 编译生成 globalmem.ko

编写 Makefile 文件, 编译 globalmem.c 文件, 生成 globalmem.ko。

Makefile 的编写如下

```

1. obj-m += globalmem.o
2.
3. KERNELDIR ?= /home/oseasy/linux-3.5

```

```

4. PWD := $(shell pwd)
5.
6. all:
7.  $(MAKE) -C $(KERNELDIR) M=$(PWD)
8.
9. clean:
10. rm -rf *.o *~ core .depend *.cmd *.ko.* *.mod.c .tmp_versions

```

```

1 obj-m += tiny4412_leds.o
2
3 KERNELDIR ?= /home/oseasy/linux-3.5
4 PWD := $(shell pwd)
5
6 all:
7  $(MAKE) -C $(KERNELDIR) M=$(PWD)
8
9 clean:
10  rm -rf *.o *~ core .depend *.cmd *.ko.* *.mod.c .tmp_versions

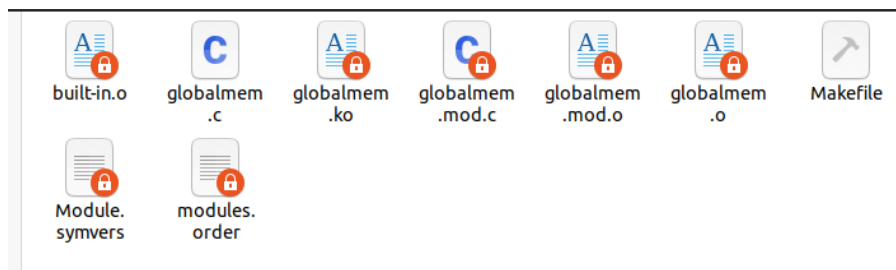
```

```

root@PC05:/home/oseasy/globalmem# make
make -C /home/oseasy/linux-3.5 M=/home/oseasy/globalmem
make[1]: 进入目录"/home/oseasy/linux-3.5"
CC [M] /home/oseasy/globalmem/globalmem.o
/home/oseasy/globalmem/globalmem.c: In function 'globalmem_read':
/home/oseasy/globalmem/globalmem.c:82:9: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
/home/oseasy/globalmem/globalmem.c: In function 'globalmem_write':
/home/oseasy/globalmem/globalmem.c:107:9: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
Building modules, stage 2.
MODPOST 1 modules
CC /home/oseasy/globalmem/globalmem.mod.o
LD [M] /home/oseasy/globalmem/globalmem.ko
make[1]: 离开目录"/home/oseasy/linux-3.5"
root@PC05:/home/oseasy/globalmem#

```

执行 Makefile, 允许效果如上图所示, 生成.ko 文件



7. 上传驱动文件到开发板

```
root@PC05: /home/oseasy
24cXX.o      hhh      service.asm
Makefile     home     share
TCPserve     i2c      sys
UDP          led      term
_new.        led-player test
a            led1     test.sh
a.out        +-----[zmodem upload - Press CTRL-C to quit]-----+
adc-test     |Sending: globalmem.ko
arm-udp      |Bytes Sent: 99237  BPS:11005
arm-udptalk  |
arm-udptalk  |Transfer complete
bin          |
button       |READY: press any key to continue...
button_test  |
button_test  |-----+
data         |      pwm      udptalk
dev          |      pwm_test udptalk.asm
eeprog.o     |      pwm_test usr
etc          |      root     var
fa-network-service sbin      www
float-test   |      sdcard  x?
float-test.soft serve.asm x??hello?7816
float-test.vfp server
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tty50
```

8. 驱动安装（模块的方式）与验证

```
[root@FriendlyARM /]# insmod ./globalmem.ko
[ 17.675000] globalmem: module license 'Dual BSD/GPL' taints kernel.
[ 17.675000] Disabling lock debugging due to kernel taint
[ 17.675000] Unable to handle kernel paging request at virtual address 01140004
[ 17.675000] pgd = ed12c000
[ 17.675000] [01140004] *pgd=00000000
[ 17.675000] Internal error: Oops: 5 [#1] PREEMPT SMP ARM
[ 17.675000] Modules linked in: globalmem(P0) libertas_sdio(0) libertas(0) zd6
[ 17.675000] CPU: 0 Tainted: P O (3.5.0-FriendlyARM #1)
[ 17.675000] PC is at module_put+0x48/0xfc
[ 17.675000] LR is at sys_init_module+0xf84/0x1b44
[ 17.675000] pc : [<c0082750>] lr : [<c0084f90>] psr: a0000013
[ 17.675000] sp : ec787ee8 ip : c0905688 fp : 00000028
[ 17.675000] r10: bf2605a4 r9 : 00000299 r8 : bf26055c
[ 17.675000] r7 : ec786028 r6 : c0084f90 r5 : bf26055c r4 : 00000000
[ 17.675000] r3 : 00000004 r2 : c08e6c30 r1 : 01140000 r0 : bf26055c
[ 17.675000] Process insmod (pid: 283, stack limit = 0xec7862f8)
[ 17.675000] Stack: (0xec787ee8 to 0xec788000)
[ 17.675000] 7ee0: ecb84600 00000000 ec9f1fc0 ec9f1fe4 00000
[ 17.675000] 7f00: bf260568 00007fff c008157c c0008460 ec786000 00000000 000ae
[ 17.675000] 7f20: f0fcf10c f0fc95c4 00000000 00000000 00000000 00000000 00000
[ 17.675000] 7f40: 00000000 f0fb7000 000183a5 f0fc95c4 f0fc9474 f0fcf10c 00008
[ 17.675000] 7f60: 00000000 00000000 00000022 00000023 0000000d 00000000 00000
[ 17.675000] 7f80: 00000003 00000001 00000069 beea6e04 00000080 c00136e8 ec780
[ 17.675000] 7fa0: 00000000 c0013540 00000001 00000069 000bc040 000183a5 000af
[ 17.675000] 7fc0: 00000001 00000069 beea6e04 00000080 beea6e08 000afa95 beea0
[ 17.675000] 7fe0: 00000001 beea6aac 0001cb50 b6e40664 60000010 000bc040 6d7e1
[ 17.675000] [<c0082750>] (module_put+0x48/0xfc) from [<c0084f90>] (sys_init_)
[ 17.675000] [<c0084f90>] (sys_init_module+0xf84/0x1b44) from [<c0013540>] (r)
[ 17.675000] Code: e34c208e e590316c e7921101 e2833004 (e7932001)
[ 18.360000] ---[ end trace 5c91106212da6272 ]---
[ 18.365000] note: insmod[283] exited with preempt_count 1
segmentation fault
[root@FriendlyARM /]#
```

这里安装显示是段错误，但是实际上安装成功了，详细看后面的验证，段错误的原因还有待考证。

执行 `cat /proc/devices` 命令 显示 150 为我们所安装的 `globalmem` 驱动

执行 `echo "hello world globalmem" > /dev/globalmem` 命令，正常显示

```

110 rtc
128 ptm
136 pts
150 globalmem
180 usb
188 ttyUSB
189 usb_device
204 ttySAC
216 rfcomm
251 ttyGS
252 watchdog

```

```

[ root@FriendlyARM / ]# echo ?hello world globalmem? > /dev/globalmem
[ root@FriendlyARM / ]# cat /dev/globalmem
"hello world globalmem"
[ root@FriendlyARM / ]#

```

实验内容 2-Tiny4412 设备驱动学习

1. 从/linux-3.5/drivers/char 下拷贝 tiny4412_leds.c (驱动的源码) 到新建目录/leds,并分析此 leds 驱动源文件

Tiny4412_leds.c 代码如下 (逐行加入了注释)

```

1. #include <linux/kernel.h> // 包含了一些核心库的定义
2. #include <linux/module.h> // 包含了模块的相关定义
3. #include <linux/miscdevice.h> // 杂项设备的相关定义
4. #include <linux/fs.h> // 文件系统的相关定义
5. #include <linux/types.h> // 包含了一些基本类型的定义
6. #include <linux/moduleparam.h> // 模块参数的相关定义
7. #include <linux/slab.h> // 内存管理的相关定义
8. #include <linux/ioctl.h> // ioctl 的相关定义
9. #include <linux/cdev.h> // 字符设备的相关定义
10. #include <linux/delay.h> // 延时函数的相关定义
11.
12. #include <linux/gpio.h> // GPIO 的相关定义
13. #include <mach/gpio.h> // 机器相关的 GPIO 定义
14. #include <plat/gpio-cfg.h> // 平台相关的 GPIO 配置定义
15.
16. #define DEVICE_NAME "leds" // 定义设备名称为"leds"
17.
18. // 定义了四个 LED 灯对应的 GPIO 引脚
19. static int led_gpios[] = {
20.     EXYNOS4X12_GPM4(0),
21.     EXYNOS4X12_GPM4(1),

```



```
22.     EXYNOS4X12_GPM4(2),
23.     EXYNOS4X12_GPM4(3),
24. };
25.
26. #define LED_NUM ARRAY_SIZE(led_gpios) // 定义LED 灯的数量
27.
28. // 定义了一个ioctl 函数, 用于处理用户空间对设备的操作请求
29. static long tiny4412_leds_ioctl(struct file *filp, unsigned int c
    md,
30.                                unsigned long arg)
31. {
32.     switch (cmd)
33.     {
34.     case 0:
35.     case 1:
36.         if (arg > LED_NUM)
37.         {
38.             return -EINVAL;
39.         }
40.
41.         gpio_set_value(led_gpios[arg], !cmd); // 设置LED 灯的状态
42.         break;
43.
44.     default:
45.         return -EINVAL;
46.     }
47.
48.     return 0;
49. }
50.
51. // 定义了设备的操作函数集
52. static struct file_operations tiny4412_led_dev_fops = {
53.     .owner = THIS_MODULE,
54.     .unlocked_ioctl = tiny4412_leds_ioctl,
55. };
56.
57. // 定义了一个杂项设备
58. static struct miscdevice tiny4412_led_dev = {
59.     .minor = MISC_DYNAMIC_MINOR,
60.     .name = DEVICE_NAME,
61.     .fops = &tiny4412_led_dev_fops,
62. };
63.
64. // 定义了模块的初始化函数
```

```
65. static int __init tiny4412_led_dev_init(void)
66. {
67.     int ret;
68.     int i;
69.
70.     for (i = 0; i < LED_NUM; i++)
71.     {
72.         ret = gpio_request(led_gpios[i], "LED"); // 请求GPIO 资源
73.         if (ret)
74.         {
75.             printk("%s: request GPIO %d for LED failed, ret = %d\n",
76.                 DEVICE_NAME,
77.                 led_gpios[i], ret);
78.             return ret;
79.         }
80.         s3c_gpio_cfgpin(led_gpios[i], S3C_GPIO_OUTPUT); // 设置
81.         gpio_set_value(led_gpios[i], 1); // 设置LED 灯的初始状态为
82.         亮
83.     }
84.     ret = misc_register(&tiny4412_led_dev); // 注册杂项设备
85.
86.     printk(DEVICE_NAME "\tinitialized\n"); // 打印初始化信息
87.
88.     return ret;
89. }
90.
91. // 定义了模块的退出函数
92. static void __exit tiny4412_led_dev_exit(void)
93. {
94.     int i;
95.
96.     for (i = 0; i < LED_NUM; i++)
97.     {
98.         gpio_free(led_gpios[i]); // 释放GPIO 资源
99.     }
100.
101.     misc_deregister(&tiny4412_led_dev); // 注销杂项设备
102. }
103.
104. module_init(tiny4412_led_dev_init); // 指定模块的初始化函数
105. module_exit(tiny4412_led_dev_exit); // 指定模块的退出函数
```

106.

```
107. MODULE_LICENSE("GPL"); // 指定模块的许可证为GPL
```

```
108. MODULE_AUTHOR("FriendlyARM Inc."); // 指定模块的作者为  
    FriendlyARM Inc.
```

代码分析:

这 tiny4412_leds.c 代码是一个 Linux 内核模块,用于控制 Tiny4412 开发板上的 LED 灯。主要逻辑是:

- (1) 定义了四个 LED 灯对应的 GPIO 引脚,以及 LED 灯的数量。
- (2) 定义了一个 ioctl 函数,用于处理用户空间对设备的操作请求。在这个函数中,根据用户传入的命令和参数,控制对应的 LED 灯亮或灭。
- (3) 定义了设备的操作函数集,其中包含了上面定义的 ioctl 函数。
- (4) 定义了一个杂项设备,设备名为“leds”,并指定了设备的操作函数集。
- (5) 定义了模块的初始化函数,在这个函数中,请求 GPIO 资源,设置 GPIO 为输出模式,设置 LED 灯的初始状态为亮,然后注册杂项设备。
- (6) 定义了模块的退出函数,在这个函数中,释放 GPIO 资源,并注销杂项设备。
- (7) 指定了模块的初始化函数和退出函数。
- (8) 指定了模块的许可证为 GPL,作者为 FriendlyARM Inc.。

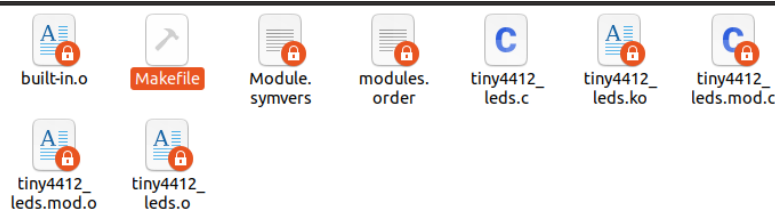
这个模块在加载时会初始化四个 LED 灯,设置为亮状态,并注册一个名为“leds”的杂项设备。用户空间可以通过 ioctl 操作这个设备来控制 LED 灯的亮灭。在卸载模块时,会释放 GPIO 资源,并注销杂项设备。

2. 编辑 Makefile

与上一个自己编写的模块驱动的编译逻辑没有什么区别所以,makefile 文件也几乎一样。

```
1. obj-m += tiny4412_leds.o  
2.  
3. KERNELDIR ?= /home/oseasy/linux-3.5  
4. PWD := $(shell pwd)  
5.  
6. all:  
7. $(MAKE) -C $(KERNELDIR) M=$(PWD)  
8.  
9. clean:  
10. rm -rf *.o *~ core .depend *.cmd *.ko.* *.mod.c .tmp_versions
```

3. 编译驱动生成 tiny4412_leds.ko



4. 使用 minicom 把 tiny4412_leds.ko 上传到 tiny4412 开发板

```
root@PC05: /home/oseasy
65 sd
6+-----[Select one or more files for upload]-----+
6|Directory: /home/oseasy/upload
6| [...]
6| globalmem.c
7| globalmem.ko
7| tiny4412_leds.ko
12|
12|
13|
13|
13|
13|
13|
13|
17|
25|
[r]
[r] ( Escape to exit, Space to tag )
+h+-----+
[root@FriendlyARM /]# [ 320.505000] saved f6400004 value 0000ffbb
[ [Goto] [Prev] [Show] [Tag] [Untag] [Okay]
[ 320.505000] saved f6400000 value 0000b200
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0
```

```
root@PC05: /home/oseasy
65 sd
66 sd
67 sd
68 sd
69 sd
70 sd
71 sd +-----[zmodem upload - Press CTRL-C to quit]-----+
128 sd |Sending: tiny4412_leds.ko
129 sd |sz: skipped: tiny4412_leds.ko
130 sd |
131 sd |Transfer complete
132 sd |
133 sd |READY: press any key to continue...
134 sd |
135 sd +-----+
179 mmc
254 device-mapper
[root@FriendlyARM /]# echo ?hello world globalmem? > /dev/globalmem
[root@FriendlyARM /]# cat /dev/globalmem
"hello world globalmem"
[root@FriendlyARM /]# [ 320.505000] saved f6400004 value 0000ffbb
[ 320.505000] saved f6400000 value 00008000
[ 320.505000] saved f6400000 value 0000b200
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0
```

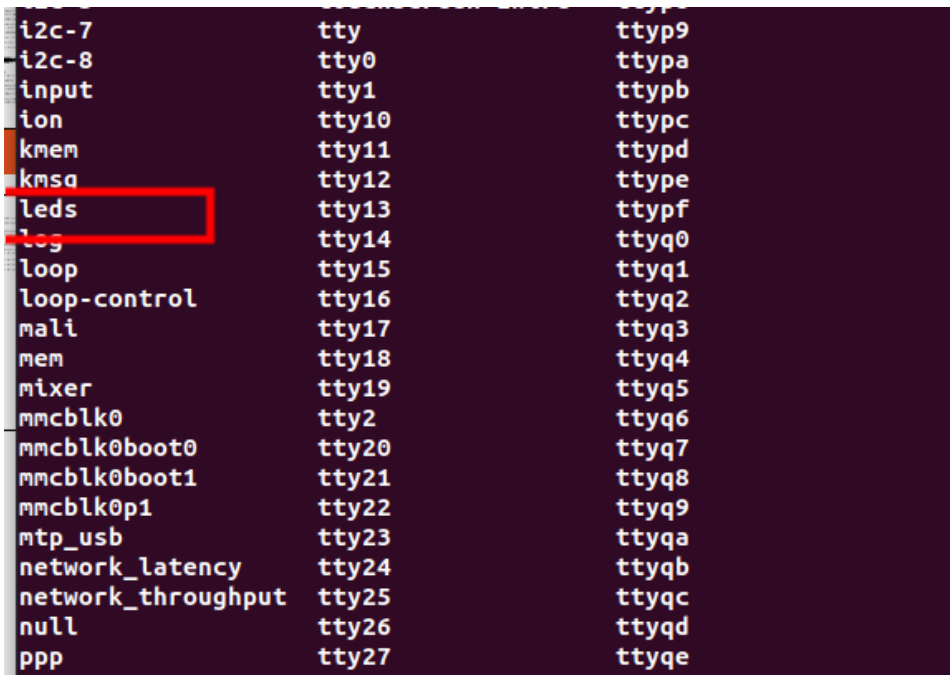
5. 驱动安装并观察结果

使用 insmod 模块化安装驱动

```
etc pwn_test usr
fa-network-service root var
float-test sbin www
float-test.soft sdcard x?
float-test.vfp serve.asm x??hello?7816
[root@FriendlyARM /]# insmod tiny4412_leds.ko
^C^C
```

```
111.580000] Stack: (0xec457ee8 to 0xec458000)
111.580000] 7ee0: ecbddf00 00000000 ec74a980 ec74a9a4 00000
111.580000] 7f00: bf264568 00007fff c008157c c0008460 ec456000 00000000 000ae
111.580000] 7f20: f0fe110c f0fdb5c4 00000000 00000000 00000000 00000000 00000
111.580000] 7f40: 00000000 f0fc9000 000183a5 f0fdb5c4 f0fdb474 f0fe110c 00008
111.580000] 7f60: 00000000 00000000 00000022 00000023 0000000d 00000000 00000
111.580000] 7f80: 00000003 00000001 00000069 bead5e04 00000080 c00136e8 ec450
111.580000] 7fa0: 00000000 c0013540 00000001 00000069 000bc040 000183a5 000af
111.580000] 7fc0: 00000001 00000069 bead5e04 00000080 bead5e08 000afa95 bead0
111.580000] 7fe0: 00000001 bead5aac 0001cb50 b6dc6664 60000010 000bc040 fffff
111.580000] [<c0082750>] (module_put+0x48/0xfc) from [<c0084f90>] (sys_init_)
111.580000] [<c0084f90>] (sys_init_module+0xf84/0x1b44) from [<c0013540>] (r)
111.580000] Code: e34c208e e590316c e7921101 e2833004 (e7932001)
112.260000] ---[ end trace 781548b4369a7b7d ]---
112.265000] note: insmod[313] exited with preempt_count 1
segmentation fault
```

最后的结果和上一个驱动安装一样，还是报了段错误，但是实际上还是安装成功了。通过 `cat /proc/devices` 命令显示 Linux 内核中注册的设备列表。



i2c-7	tty	ttyp9
i2c-8	tty0	ttypa
input	tty1	ttypb
ion	tty10	ttypc
kmem	tty11	ttypd
kmsa	tty12	ttype
leds	tty13	ttypf
log	tty14	ttyq0
loop	tty15	ttyq1
loop-control	tty16	ttyq2
mali	tty17	ttyq3
mem	tty18	ttyq4
mixer	tty19	ttyq5
mmcblk0	tty2	ttyq6
mmcblk0boot0	tty20	ttyq7
mmcblk0boot1	tty21	ttyq8
mmcblk0p1	tty22	ttyq9
mtp_usb	tty23	ttyqa
network_latency	tty24	ttyqb
network_throughput	tty25	ttyqc
null	tty26	ttyqd
ppp	tty27	ttyqe

可以发现还是存在的安装好的驱动的，所以本驱动程序已经安装集成到了开发板上并且处于运行状态。

实验环境（含主要设计设备，器材，软件等）

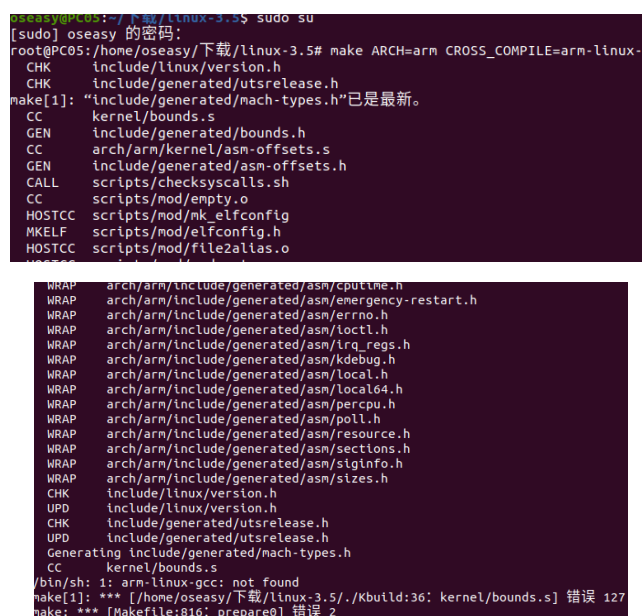
硬件：Ubuntu 主机、Tiny4412 开发板

软件：X86 架构的 Linux 系统，arm 架构的 linux 系统，arm-linux-gcc,minicom, 开发板 Linux 内核

实验结果与总结：

1. 实验中出现问题总结

(1) 内核编译出错



```
oseasy@ubuntu:~/下载/linux-3.5$ sudo su
[sudo] oseasy 的密码:
root@PC05:/home/oseasy/下载/linux-3.5# make ARCH=arm CROSS_COMPILE=arm-linux-
CHK include/linux/version.h
CHK include/generated/utsrelease.h
make[1]: "include/generated/mach-types.h"已是最新。
CC kernel/bounds.s
GEN include/generated/bounds.h
CC arch/arm/kernel/asm-offsets.s
GEN include/generated/asm-offsets.h
CALL scripts/checksyscalls.sh
CC scripts/mod/empty.o
HOSTCC scripts/mod/mk_elfconfig
MKELF scripts/mod/elfconfig.h
HOSTCC scripts/mod/file2alias.o
HOSTCC scripts/mod/strip.o
WRAP arch/arm/include/generated/asm/cputime.h
WRAP arch/arm/include/generated/asm/emergency-restart.h
WRAP arch/arm/include/generated/asm/errno.h
WRAP arch/arm/include/generated/asm/toctl.h
WRAP arch/arm/include/generated/asm/irq_regs.h
WRAP arch/arm/include/generated/asm/kdebug.h
WRAP arch/arm/include/generated/asm/local.h
WRAP arch/arm/include/generated/asm/local64.h
WRAP arch/arm/include/generated/asm/percpu.h
WRAP arch/arm/include/generated/asm/poll.h
WRAP arch/arm/include/generated/asm/resource.h
WRAP arch/arm/include/generated/asm/sections.h
WRAP arch/arm/include/generated/asm/siginfo.h
WRAP arch/arm/include/generated/asm/sizes.h
CHK include/linux/version.h
UPD include/linux/version.h
CHK include/generated/utsrelease.h
UPD include/generated/utsrelease.h
Generating include/generated/mach-types.h
CC kernel/bounds.s
/bin/sh: 1: arm-linux-gcc: not found
make[1]: *** [/home/oseasy/下载/linux-3.5/./kbuild:36: kernel/bounds.s] 错误 127
make: *** [Makefile:816: prepare0] 错误 2
```

后来发现是源码有问题，需要修改一点。之后就编译成功了。

(2) 编译驱动失败

```
oseasy@PC05:~/globalmem$ sudo su
[sudo] oseasy 的密码:
root@PC05:/home/oseasy/globalmem# make
cc -c -o globalmem.o globalmem.c
globalmem.c:5:11: fatal error: linux/mm.h: 没有那个文件或目录
  5 | #include <linux/mm.h>
    |           ^~~~~~
compilation terminated.
make: *** [<内置>: globalmem.o] 错误 1
root@PC05:/home/oseasy/globalmem#
```

出现这种情况一般是 makefile 的编写有问题。

这里我们可以分析一下正确的 makefile 的逻辑。

1. `obj-m += globalmem.o` # 将 `globalmem.o` 添加到要编译的模块列表中
- 2.
3. `KERNELDIR ?= /home/oseasy/linux-3.5` # 如果环境变量 `KERNELDIR` 未设置，则默认为 `/home/oseasy/linux-3.5`
4. `PWD := $(shell pwd)` # 将当前目录赋值给 `PWD` 变量
- 5.
6. `all:` # 定义 `all` 目标，它是默认目标
7. `$(MAKE) -C $(KERNELDIR) M=$(PWD)` # 在 `KERNELDIR` 目录下执行 `make`，其中 `M` 变量指定了模块的源代码位置
- 8.
9. `clean:` # 定义 `clean` 目标，用于清理编译生成的文件
10. `rm -rf *.o *~ core .depend *.cmd *.ko.* *.mod.c .tmp_versions` # 删除所有 `.o`、`~`、`core`、`.depend`、`.cmd`、`.ko.*`、`.mod.c` 文件和 `.tmp_versions` 目录

前面一些定义我们可以看注释就可以明白，关键是

All 目标的编译指令

`$(MAKE) -C $(KERNELDIR) M=$(PWD)`

我们仔细分析一下这个指令

- `$(MAKE)`: 这部分调用 `make` 工具，它负责根据 `Makefile` 文件构建和管理软件项目。
- `-C $(KERNELDIR)`: `-C` 标志指定在执行 `make` 命令之前要切换到到的目录。在本例中，`$(KERNELDIR)` 扩展为已定义的路径 (`/home/oseasy/linux-3.5`)，指示 `make` 工具应将工作目录更改为 Linux 内核源代码目录。
- `M=$(PWD)`: `M=` 变量赋值将 `M` 变量的值设置为当前工作目录 (`$(PWD)`)。 `PWD` 变量扩展为 `Makefile` 所在目录的实际路径

所以该指令在此处的含义就是：

- 使用 `-C` 标志将目录更改为 Linux 内核源代码目录 (`/home/oseasy/linux-3.5`)。
- 使用 `=` 运算符将 `M` 变量设置为当前工作目录 (`$(PWD)`)。

- 在内核源代码目录中执行 `modules` 目标，这可能涉及内核模块的编译和构建。

但是细心的同学可能会发现 `-c` 和 `m` 不是冲突了吗？又把目录设置到内核把目录设置到驱动源文件目录？

将当前工作目录设置为 `M` 变量的值并同时在工作目录更改为 Linux 内核源代码目录看似矛盾，但实际上这两种操作是相辅相成的，共同确保了模块的正确编译和加载。

a) 传递模块路径：

将当前工作目录设置为 `M` 变量的值，主要目的是传递模块的路径信息。`make` 工具通常使用 `M` 变量来确定要编译的模块源文件所在的目录。例如，如果模块的源文件位于当前工作目录的 `src` 子目录中，则 `M=$(PWD)` 命令会将 `$(PWD)/src` 路径传递给 `make` 工具。

b) 进入内核源代码目录：

将工作目录更改为 Linux 内核源代码目录是必要的，因为内核模块需要在特定的环境下进行编译和构建，该环境通常位于内核源代码树中。将工作目录更改到该目录可以确保 `make` 工具能够找到必要的头文件、库和其他资源。

c) 工作目录的意义：

当前工作目录是指您在命令行中执行 `make` 命令所在的目录。该目录通常包含 `Makefile` 文件和其他与模块构建相关的文件。将 `M` 变量设置为当前工作目录，意味着 `make` 工具会从该目录及其子目录中查找模块源文件。

d) 协同工作：

这两个操作看似矛盾，但实际上它们协同工作，确保了模块的正确编译和加载。

- `M=$(PWD)` 命令将模块的路径信息传递给 `make` 工具，指示它在何处查找模块源文件。
- 将工作目录更改为 Linux 内核源代码目录，为模块的编译和构建提供了必要的环境。

所以这样才能确保编译成功。

再来看之前出现的错误，可以发现找不到头文件，所以一定是编译的环境没有在内核的目录下，要在 `makefile` 中指出才行，修改之后就编译成功了。

```
root@PC05:/home/oseasy/globalmem# make
make -C /home/oseasy/linux-3.5 M=/home/oseasy/globalmem
make[1]: 进入目录"/home/oseasy/linux-3.5"
CC [M] /home/oseasy/globalmem/globalmem.o
/home/oseasy/globalmem/globalmem.c: In function 'globalmem_read':
/home/oseasy/globalmem/globalmem.c:82:9: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
/home/oseasy/globalmem/globalmem.c: In function 'globalmem_write':
/home/oseasy/globalmem/globalmem.c:107:9: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
Building modules, stage 2.
MODPOST 1 modules
CC /home/oseasy/globalmem/globalmem.mod.o
LD [M] /home/oseasy/globalmem/globalmem.ko
make[1]: 离开目录"/home/oseasy/linux-3.5"
root@PC05:/home/oseasy/globalmem#
```

2. 实验结果

通过本实验,成功完成了以下工作:

- (1) 配置了 Ubuntu 主机上的 `arm-linux-gcc` 交叉编译环境和 `minicom` 串口工具。
- (2) 编译了 Linux 3.5 内核,生成了 `zImage` 映像文件。
- (3) 基于 Linux 内核源码的 `char` 驱动程序框架,编写并编译了一个简单的字符设备驱动程序 `globalmem.ko`。
- (4) 将 `globalmem.ko` 驱动模块加载到 Tiny4412 开发板上,通过文件节点 `/dev/globalmem` 进行读写操作,验证了驱动的正确性。

(5) 分析并编译了 Tiny4412 开发板上的 LED 驱动程序 `tiny4412_leds.ko`,并成功加载到开发板上运行。

实验结果良好均符合预期。

3. 实验总结感悟

通过本次设备驱动实验,我获益匪浅,不仅实现了预期的实验目标,而且对嵌入式 Linux 驱动开发有了更加全面和深入的理解。

首先,我掌握了 `Makefile` 文件的编写方法,熟悉了内核驱动程序的代码架构和主要函数实现。通过编写简单的字符设备驱动 `globalmem.ko`,然后将其加载到 Tiny4412 开发板上进行读写测试,我了解了驱动模块编译和加载到内核的全过程。

其次,分析并实践了 Tiny4412 开发板上的 LED 驱动程序 `tiny4412_leds.ko`,使我对块设备驱动及其特性有了全面认知。通过这一过程,我掌握了嵌入式系统开发的基本理论知识,以及安装、调试、运行驱动程序的基本方法。

更重要的是,本次实验让我进一步深入理解了嵌入式系统开发的整体过程和所使用的工作工具。我意识到了嵌入式计算机与传统 PC 机在硬件架构和软件运行环境上的显著差异,对嵌入式系统有了更为深刻的认知。

作为本学期嵌入式系统开发实践环节的最后一次实验,通过四次实验的锻炼,我可谓是打开了通往嵌入式世界的大门。我学习并掌握了嵌入式系统开发的基础知识,对 Linux 系统下的嵌入式开发流程、工具、方法和原则等有了全面的理解。

我要衷心感谢老师们长期以来的辛勤付出,精心准备实验内容,耐心讲解知识点,悉心教导实践技能,使我在嵌入式开发这一全新领域有了质的突破。本门课程坚定了我要深入计算机底层、研究嵌入式系统的决心,争取在嵌入式开发领域有进一步的挖掘和提升。我将继续努力,做到软硬件知识的有机结合,实现嵌入式开发技术的更上一层楼。