

计算机系统综合实践报告

梁盾 2012011373

高博 2012012139

August 14, 2015

Contents

1	综述	2
1.1	试验目标及完成情况	2
1.2	实验分工	2
1.3	试验内容	2
2	网口收发包	2
2.1	内核态初始化	2
2.2	收发数据包	2
3	ucore 内网络栈的实现	3
3.1	使用方法	3
4	基于 ucore 结构实现文件系统	3
4.1	原有文件系统分析	3
4.2	具体实现	3
4.3	使用方法	4
4.4	解决的问题	4
5	试验成果展示	4
5.1	文件传输	5
5.2	文本/信息传输	5
6	试验中遇到的问题	6
6.1	pyserial 无法正常工作	6
6.2	网口寄存器读写不正常	6
6.3	网口不能收发数据包	6
6.4	ucore 运行不稳定	6
6.5	Flash 读写校验异常	6
6.6	TCP 链接不响应	6
7	References	6

1 综述

试验进程维护在 git: <https://github.com/cjld/armcpu>

1.1 试验目标及完成情况

1. 在 ThinPad II 上实现精简的 MIPS32 指令系统, 能运行教学操作系统 ucore
 - (a) 实现网口
 - (b) 实现文件系统
2. 操作系统移植
3. 将 decaf 编译后, 通过网络接口上载到教学机并运行

1.2 实验分工

1. DM9000AEP verilog 代码: 梁盾
2. DM9000AEP 驱动中断编写: 梁盾
3. 链路层协议栈: 梁盾
4. TCP/IP 实现: 高博
5. Http 获取文件: 高博
6. ucore 中文件系统完善: 高博

1.3 试验内容

本实验基于<https://github.com/jia-kai/armcpu>。

1. 前人成果的重现
2. 网口收发包
3. ucore 内网络栈的实现
4. 基于 ucore 结构实现可以动态创建文件的文件系统
5. 利用 tcp 将编译后的 decaf 程序通过网络接口上载到教学机并运行

2 网口收发包

主要参考 DM9000AEP application note, 本部分代码在 ucore/ours/ethernet.c 中有实现。

2.1 内核态初始化

在 ucore/driver/eth.c 中包含了对硬件的初始化代码, 在初始化完成以后, 可以发现以太网接口的右侧黄灯亮, 代表了我们成功建立了以太网 100M 全双工的链接, 在 pc 端使用 `ethtool -i eth0` 可以发现 link 已经建立好

2.2 收发数据包

进入 term 以后运行 server, 不加参数, 这时候会只监听网络上的 arp 包和 ICMP 包, 可以使用 `arping -I eth0 192.168.2.2` 可以正确的受到回复, 关于 ip 和 mac 的配置请详见代码

3 ucore 内网络栈的实现

本部分实现在 ucore/ours/network/中, 以及往 user/libs 中移植了一份代码。

本部分参考了<https://github.com/blahgeek/MadeAComputerIn20Days>网络部分。

由于内核态的网络栈每次改动都需要重新烧写内核, 十分影响效率, 因而我们将网络栈写在用户态以方便调试。在网络栈中实现了对 link 层、ip 层、传输层表头的解析, 同时我们完善了 tcp 协议的逻辑, 完成了标准的 tcp “握手-传输-分手” 过程, 可以与 python 的 HTTP server 对接, 发送 HTTP get 请求, 并接受 HTTP server 发送的数据/文件, 并可以对其进行简单的解析, 可以显示文本/图片资源, 实现了简单的浏览器功能。

提供了便于使用的用户态库函数, 详见 user/libs/network.h, 函数 tcp_send(cmd, callback, timeout), 提供了便捷的异步 IO, 并且提供了超时退出的功能。

增加了系统调用 wait_eth, 用于等待数据包的传入, 并且方便将内核态与用户态的程序隔离开来。

3.1 使用方法

由于我们目前链路层的 MAC 地址直接 hardcode 成了我们自己网口的 MAC, 所以还需要手动在代码中修改 MAC 地址。

另外板子使用的 ip 为 192.168.2.2, 主机使用的 ip 为 192.168.2.1。

然后可以使用如下指令启动一个 Http server, 端口号为 8888

```
python -m SimpleHTTPServer 8888
```

按照贾开学长中的方法, 进入 term 以后, 可以看到一个应用程序 wget, 该程序接受两个参数第一个参数为远端文件的文件名, 第二个为本地保存的文件名

比如如下指令, 可以列出远端服务器有哪些文件

```
wget list
cat list
```

该段代码的含义为吧根目录的文件列表保存到名字叫 list 的文件中, 然后 cat 该文件, 显示出 list

```
wget myls ls
mysls
```

以上代码可以在教学机上新建文件 myls, 将 PC 机上名为 ls 的程序通过网口传到教学机上。在运行 myls, 发现可以列出文件列表。

4 基于 ucore 结构实现文件系统

本部分实现主要在 ucore 内核 sfs_inode.c 中。

4.1 原有文件系统分析

原有文件系统支持文件的读写, 但不能动态创建文件。通过对 Makefile 等文件的分析, 我们发现原系统在编译 ucore 的同时使用 ucore/tools/mksfs.c 将特定文件 (Makefile 中的 USER_APPLIST) 信息读出, 创建 file entry 和 inode, 并写入文件 initrd.img, 加到 kernel 中并烧入 flash, 在 ucore 运行时再从 disk 中读出 entry 和 inode。ucore_lab8 中的文件系统也是这样生成的。

4.2 具体实现

通过对原有文件系统的分析, 我们发现需要做的就是将 mksfs.c 的动态版本实现出来, 可以随时创建 entry 和 inode 并将其写入 disk。使用 ucore 内部已有的动态空间分配机制, 实现了 sfs_create(), 为新文件分配 inode。

4.3 使用方法

提供用户态新建/拷贝文件的程序: testfile。

Usage: testfile dst_file_name [src_file_name]

新建 dst_file, 并将 src_file 拷贝进去, 如果只有一个参数, 只会新建空文件。

```
testfile myls ls
```

会将 ls 拷贝到新文件 myls 中, 之后再输入 myls, 发现它也可以列出文件列表。

4.4 解决的问题

由于本部分使用 qemu 调试, 在调试过程中我们发现一些原本应该是空的 block 也写入了 inode。后来发现, 使用刚刚编译出来的 .img 文件不存在这个问题, 在 qemu 中跑过之后再次模拟就会出问题。因为在 qemu 模拟过程中会将创建的文件写入 disk, 而在 qemu 退出后, 这些信息并不会消失。果然在板子操作的时候没有发生类似问题。

5 试验成果展示

```
user sh is running!!!
$ ls
'.' is [directory] 2(hlinks) 11(blocks) 2816(bytes)
[d] 2(h) 11(b) 2816(s) .
[d] 2(h) 11(b) 2816(s) ..
[-] 1(h) 22(b) 87531(s) wget
[-] 1(h) 19(b) 76854(s) ls
[-] 1(h) 19(b) 75653(s) run
[-] 1(h) 1(b) 17(s) prog_holder
[-] 1(h) 1(b) 318(s) hello
[-] 1(h) 19(b) 74940(s) cat
[-] 1(h) 19(b) 77007(s) sh
[-] 1(h) 19(b) 75100(s) snake
[-] 1(h) 17(b) 69424(s) slideshow
lsdir: step 4
$ wget myls ls
msglen: 11, msg: GET /ls

TCP handshake initiated
handle arp
TCP handshake complete
tcp_handle send msg: G, len: 11
TCP_CLOSED
call_rcv_len: 76854
$ ls
'.' is [directory] 2(hlinks) 12(blocks) 2816(bytes)
[d] 2(h) 12(b) 2816(s) .
[d] 2(h) 12(b) 2816(s) ..
[-] 1(h) 22(b) 87531(s) wget
[-] 1(h) 19(b) 76854(s) ls
[-] 1(h) 19(b) 75653(s) run
[-] 1(h) 1(b) 17(s) prog_holder
[-] 1(h) 1(b) 318(s) hello
[-] 1(h) 19(b) 74940(s) cat
[-] 1(h) 19(b) 77007(s) sh
[-] 1(h) 19(b) 75100(s) snake
[-] 1(h) 17(b) 69424(s) slideshow
[-] 1(h) 19(b) 76854(s) myls
lsdir: step 4
$ myls
'.' is [directory] 2(hlinks) 12(blocks) 2816(bytes)
[d] 2(h) 12(b) 2816(s) .
[d] 2(h) 12(b) 2816(s) ..
[-] 1(h) 22(b) 87531(s) wget
[-] 1(h) 19(b) 76854(s) ls
[-] 1(h) 19(b) 75653(s) run
[-] 1(h) 1(b) 17(s) prog_holder
[-] 1(h) 1(b) 318(s) hello
[-] 1(h) 19(b) 74940(s) cat
[-] 1(h) 19(b) 77007(s) sh
[-] 1(h) 19(b) 75100(s) snake
[-] 1(h) 17(b) 69424(s) slideshow
[-] 1(h) 19(b) 76854(s) myls
lsdir: step 4
$
```

Figure 1: 文件传输教学机端

5.1 文件传输

如 Fig. 1所示, 首先调用 `ls`, 文件系统内并没有 `mys` 文件。然后输入 `wget myls ls`, 将电脑上的 `ls` 文件 (是提前编译好的 `mips` 可执行文件) 传输到教学机上, 然后再在教学机上输入 `mys`, 发现它具有 `ls` 的功能, 列出了文件列表, 并且文件列表中出现了新文件 `mys`。文件的传输和存储完成。如果传过来的文件是图片, 我们还可以用 `wgetimg` 将其显示出来。

5.2 文本/信息传输

```
user sh is running!!!
$ wget info
msglen: 9, msg: GET /

TCP handshake initiated
TCP handshake complete
tcp_handle send msg: G, len: 9
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="google.pic">google.pic</a>
<li><a href="logo.png">logo.png</a>
<li><a href="logo11w.png">logo11w.png</a>
<li><a href="ls.pic">ls.pic</a>
<li><a href="ls.png">ls.png</a>
<li><a href="prime.png">prime.png</a>
<li><a href="tsinghua.pic">tsinghua.pic</a>
</ul>
<hr>
</body>
</html>
$ ls
.  is [directory] 2(hlinks) 12(blocks) 2816(bytes)
[d] 2(h) 12(b) 2816(s) .
[d] 2(h) 12(b) 2816(s) ..
[-] 1(h) 22(b) 87396(s) wget
[-] 1(h) 19(b) 76854(s) ls
[-] 1(h) 19(b) 75653(s) run
[-] 1(h) 1(b) 17(s) prog_holder
[-] 1(h) 1(b) 318(s) hello
[-] 1(h) 19(b) 74940(s) cat
[-] 1(h) 19(b) 77007(s) sh
[-] 1(h) 19(b) 75100(s) snake
[-] 1(h) 17(b) 69424(s) slideshow
[-] 1(h) 1(b) 442(s) info
lsdir: step 4
$ cat info
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="google.pic">google.pic</a>
<li><a href="logo.png">logo.png</a>
<li><a href="logo11w.png">logo11w.png</a>
<li><a href="ls.pic">ls.pic</a>
<li><a href="ls.png">ls.png</a>
<li><a href="prime.png">prime.png</a>
<li><a href="tsinghua.pic">tsinghua.pic</a>
</ul>
<hr>
</body>
</html>
```

Figure 2: 文本/信息传输

如 Fig. 2所示, 输入 `wget info`, 可以获得电脑端 `server` 的信息, 显示出来, 并存到文件 `info` 中 (如果不存在会创建一个)。之后会发现创建了文件 `info`, 并可以使用 `cat` 来查看其信息。

如 Fig. 3所示, 使用

```
python -m SimpleHTTPServer 8888
```

在 PC 端搭建一个 TCP 服务器, 可以与教学机对接并接受教学机的 TCP 请求 (192.168.2.2 为教学机 IP)。

```
[cjd@cjld-Y400] - [~/summer_project/armcpu/ucore/obj/user] - [2015-08-14 04:13:19]
[1] <git:(master 30a790a*) > python -m SimpleHTTPServer 8888
Serving HTTP on 0.0.0.0 port 8888 ...
192.168.2.2 - - [14/Aug/2015 16:16:31] "GET /ls" 200 -
192.168.2.2 - - [14/Aug/2015 16:26:26] "GET /ls" 200 -
192.168.2.2 - - [14/Aug/2015 16:36:13] "GET /" 200 -
192.168.2.2 - - [14/Aug/2015 16:54:14] "GET /ls" 200 -
192.168.2.2 - - [14/Aug/2015 17:06:36] "GET /ls" 200 -
```

Figure 3: PC 端 server

6 试验中遇到的问题

6.1 pyserial 无法正常工作

使用 pip 安装 pyserial ≥ 1.7 可以解决这个问题

或者通过手动设置停止位的方法解决这个问题, 详见文件 serial_fix.py

6.2 网口寄存器读写不正常

症状为读写的数据发生跳跃, 正常数据为 123456, 会出现 133466 的数据

严格按照 spec 中给的时序来实现, eth_cmd, eth_iow, 几个信号的 timing 会影响到读写寄存器, 详见文件 phy_mem_ctrl.v 中的网口控制代码段, 以及 datasheet 中关于 timing 的描述。

6.3 网口不能收发数据包

以太网驱动初始化以后, 发现无法正常建立 100M 全双工的链接, 只能建立 10M 半双工的链接, 该问题为硬件问题。

6.4 ucore 运行不稳定

会产生各种不同的 panic, 更换不同的板子有所好转, 偶然发现使用小米充电器的电源线有明显好转, 尚未试验证明, 很有可能电源的最大电流过低会导致板子工作不正常, 有待考证。

6.5 Flash 读写校验异常

由于地址对齐问题, 发回来的 flash 校验程序会多发一个 0xff 在行末, 使用 mydiff.py 可以解决这个问题, 详见 flash_file.sh 文件。

6.6 TCP 链接不响应

最开始我们每次建立连接的时候, 发现第一次可以正确建立 TCP 链接, 而第二次链接需要等待一会才可以使用, 发现这是因为 linux 存在一个叫 Ephemeral_port 的东西, 一个 Ephemeral_port 在使用过后需要过一段时间才能继续使用, 为了解决这个问题, 我们每次随机 client 的端口号, 该问题得到解决, wget 应用程序因此可以重复使用。

7 References

- <https://github.com/jia-kai/armcpu>
- <https://github.com/blahgeek/MadeAComputerIn20Days>
- DM9000AEP datasheet

- DM9000AEP Application Note
- Ephemeral port