μcore proj5 文档

计83 朱文雷 2008011369

June 10, 2011

Contents

1 说明 **3**

2 网络的总体结构 **3**

3 网络协议栈的初始化 **4**

4 网卡驱动 **4**
4.1 加载 PCI 驱动 5
4.2 e100 驱动加载过程 5
4.3 e1000 驱动的编写 6
4.3.1 在 Virtual Box 中开发 6

5 socket wrap 封装层 **6**

6 syscall 层 **7**
6.1 syscall 参数不够的问题 8

7 运行 **8**
7.1 qemu 版本 8
7.2 环境配置 9
7.3 IP 配置 9
7.4 测试程序 9

8 主要工作 **10**
8.1 移植胡刚之前的 ucore 10
8.2 之前的 Bug 10
8.2.1 初始化问题 10
8.2.2 调度器导致的问题 11
8.3 增加 syscall 11
8.4 编写网络测试 11
8.5 移植 FTP 11
8.5.1 为移植 FTP 增加的一些函数 11
8.5.2 为移植 FTP 做的其他改动 11
8.5.3 FTP 已知的问题 12
8.6 增加注释 12
8.7 e1000 网卡驱动编写 12
8.8 Virtual Box 自动启动 12

9 一些问题 **12**
9.1 文件系统关闭问题 12

1 说明

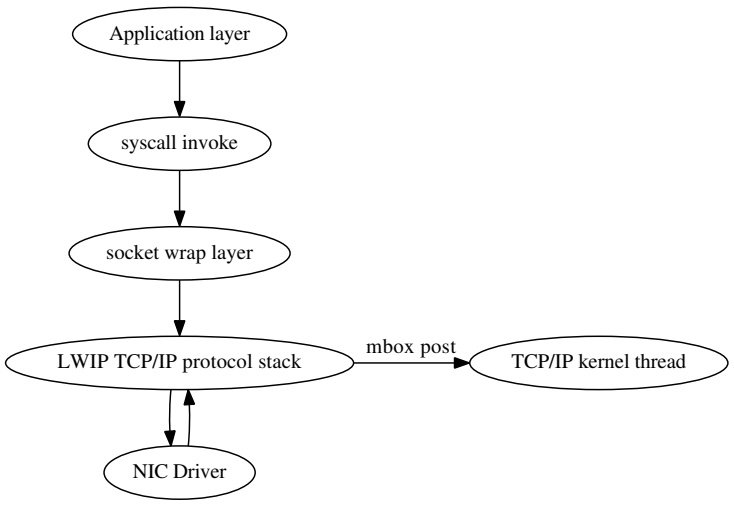
- 开发环境：Arch Linux 2.6.38
- 编译器：gcc-4.4。之前 Arch Linux 的 GCC 升级到了 4.6，然后出现了 bootblock 编译之后大于 512 个字节的问题。后来转用 gcc-4.4。陈睿同学解决了这个问题。由于担心还有其他问题，就继续使用 gcc-4.4
- VirtualBox 网卡和控制器选择：PRO/1000 MT Desktop Adapter, 82540EM Gigabit Ethernet Controller
- 相关文档：
  - LWIP 文档 trunk/doc/lwip.pdf
  - e100 驱动文档 trunk/doc/8255x_OpenSDM.pdf
  - e1000 驱动文档 trunk/doc/8254x_GBe_SDM.pdf
- 从 ucore 的 trunk/i386/lab6_filesystem/proj19 fork 出来。分支版本为 r332

2 网络的总体结构

ucore 中的网络分为 2 大块

- 网卡驱动部分，在初始化时完成，向 TCP/IP 网络协议栈提供 send 和 receive 支持。
- 网络协议栈部分，也在初始化时完成，然后向应用层提供服务。

总体结构图如下：



用户的网络应用请求首先经过一层用户层封装 (user/lib/sockets.c) 调用相应 syscall，syscall 取出参数后，大多数都会经过一层 socket wrap 层 (kern/socket/sockwrap.c) 。这个封装层的用途是把用户空间的数据安全的复制过来(用 lock_mm 和 unlock_mm 保证)。然后再调用 LWIP 层的相应函数完成操作（否则的话，进入 LWIP 之后许多操作就不在当前用户进程之内进行操作了，如果不把用户数据复制过来，一旦进程切换掉，指针就飞了）。

进入 LWIP 之后，LWIP 进行协议的相关操作，并调用网卡驱动来完成任务。实际上，操作都不是在当前用户进程中执行的，而是通过 mbox 的 post 操作传给 tcpip_thread 来进行操作的。这也就是上面需要复制用户数据的原因。

下面分别详细说明。

3 网络协议栈的初始化

在 ucore 中，TCP/IP 协议栈在网卡驱动之前进行初始化(因为网卡驱动加载的时候需要往 LWIP 的 netif 列表里加入网卡，因此让 LWIP 先初始化比较方便)。初始化代码为:

```
tcpip_init(0, 0);
```

其中的 2 个参数 0 表示没有其他 init 函数需要回调。

TCP/IP 协议栈的初始化代码在 kern/lwip/api/tcpip.c 中，分为 2 个步骤:

1. 各个网络协议的初始化。调用函数为 lwip_init。它的作用是初始化各个网络协议和一些内存申请等工作，另外网络接口链表也在这个过程里进行初始化工作。

2. 开启 tcpip 的 kernel thread。这个过程先是 new 了一个 mbox 结构，然后开启 kernel thread，运行的函数是 tcpip_thread。它的作用实际上只有一个，就是不断从 mbox 中取出消息，里面有需要执行的函数和参数，然后执行。实际上用户在调用各个网络 syscall 的时候，到了 TCP/IP 协议栈那一层的时候，都是把核心函数通过 mbox post 到这个 TCP/IP thread 里面执行的，这样就不会 block。这是 LWIP 的 operating system emulation layer 要求的。（见 lwip doc p3）

4 网卡驱动

网卡驱动的加载实际上是在 PCI 驱动对设备进行遍历的时候加载的。在 kern/init/init.c 中的初始化代码为:

```
pci_init();
```

因此下面先说 PCI 驱动的加载。

4.1 加载 PCI 驱动

在 pci_init 的时候，会调用 pci_scan_bus 函数来寻找所有的设备并加载驱动。它递归的从 root bus 开始(bus number 为 0)，遍历所有的 device id (32 个) 和 function id (8 个)，对每一个有效的设备进行 attach 操作，即分别从 CLASS 和 SUBCLASS 或者是 VENDOR 和 PRODUCT 来匹配驱动，然后执行相关驱动程序的 attach 函数。因此，对于驱动程序来说，只要把驱动加入驱动列表，然后编写相应的驱动程序加载入口函数即可。

增加 e1000 驱动之后的驱动列表如下：(kern/driver/pci.c)

```
struct pci_driver pci_attach_vendor[] = {
    { PCI_VENDOR_INTEL, PCI_PRODUCT_E100, &ether_e100_attach },
    { PCI_VENDOR_INTEL, E1000_DEV_ID_82540EM, &ether_e1000_attach},
    { 0, 0, 0 },
};
```

其中，最后的 0,0,0 是用来判断结束用的。

4.2 e100 驱动加载过程

加载的入口函数为 ether_e100_attach (kern/driver/e100.c)。
加载分为以下几个过程:

- 一些变量初始化和内存申请工作。包括 3 个 semaphore 的初始化。
- 硬件的初始化配置和读取参数工作。这包括:
  - 调用 pci_func_enable 函数，激活设备。并读取其 BARs 获取寄存器映射的 PORT IO 空间。
  - reset 工作。port soft reset，函数为 e100_reset。
  - 读取设备的 MAC 地址。打印和保存。这个没有单独的函数，在 attach 代码中。
- 加入 network interface 列表，并配置。
  - 配置 IP netmask 和 Gateway。这个工作按理说不应该在网卡驱动中来做。只是在 ucore 中简化了。放在这里比较方便。
  - 给 netif 的 send 和 receive 函数指针分别赋值为 e100_send 和 e100_receive，这样让 lwip 协议栈能够调用到网卡驱动的发送和接收函数。这也是 LWIP 和 网卡驱动的上下层交互的地方。
- 中断处理。中断处理由 picirq 来处理，它是 pic 的驱动，会将网卡中断加入其列表并设置相应的 mask 等。
- 开启一个 kernel thread 对网卡接收的数据进行处理。一旦收到数据就调用 ethernetif_input 函数将收到的包交给协议栈。

4.3 e1000 驱动的编写

qemu 和 Virtual Box 都有 e1000 网卡驱动，型号都有 82540EM ，但实际上发现有所不同。qemu 的 e1000 网卡不符合 intel 的规范。(根据文档 73 页，BAR 中 IO Register space 应该在 BAR2 或 BAR4 位置，大小为 8 byte，但 qemu 在 BAR1 ，大小为 64 byte)。并且使用 port I/O 无法正确的读取数据。同样的代码在 Virtual Box 上可以正确地读出网卡的 MAC 地址，但是在 qemu 上读出来的为全 ff。但是在 qemu 上运行 linux 是可以正确的处理网卡的。不过这样仍然没有任何帮助，因为 Linux 使用的是 Memory I/O，而非 Port I/O)

e1000 驱动比 e100 要复杂不少。由于时间的原因，目前仍然没有做完。当前进展是初始化工作基本完成。可以正确读出 MAC 地址。加入 network interface 列表和配置也完成。还差中断处理和 send receive 函数。目前 e1000 驱动已经有 1300 行代码。

4.3.1 在 Virtual Box 中开发

由于上面的缘故，开发 e1000 驱动主要在 Virtual Box 中运行测试。我写了一个脚本可以在命令行中创建 ucore 的 Virtual Box Guest OS 相关文件。

执行过程如下:

```
make
./create_ucore_vm.sh
make vb
```

其中 make vb 实际上就是执行

```
VBoxManage startvm ucore
```

如果要开启 Virtual Box 的 debug 工具的话，则需要用以下命令来运行 Virtual Box

```
VirtualBox --startvm ucore --debug
```

另外，安装 Virtual Box 后最好运行一下

```
/etc/rc.d/vboxdrv setup
```

更新一下相关内核模块。

5 socket wrap 封装层

socket wrap 的作用前面已经说过了，就是把用户空间的数据复制到内核空间来。典型过程如下:

```
lock_mm(mm);
{
    copy_from_user(mm, k_addr, addr, k_addrlen, 1);
}
unlock_mm(mm);
```

同样，如果一个函数 (比如 recv) 需要将数据返回的话，也要使用 copy_to_user 把内核空间的数据复制到用户空间去。

另外 sendmsg 函数，recvmsg 函数 和 socketpair 函数由于在 LWIP 中没有直接的支持，因此在 sockwrap 层自己实现了。

对于 sendmsg 函数，它的发送的数据在 发送过来的 msghdr 结构体中。这个结构体的定义我放在了 kern/lwip/include/lwip/sockets.h 中。它的主要工作是把 msghdr 所携带的所有 需要发送的信息复制到内核空间，然后调用 lwip_sendto 函数发送。

recvmsg 函数的工作与 sendmsg 类似。

socketpair 的用途是创建一个已连接的 socket 对，实际上它的主要作用就是 IPC，在许多地方的实现就是直接调用 pipe 函数。我按照理解实现了一个 socketpair ，过程如下：

1. 创建 server socket
2. server socket bind 到 0 端口
3. server listen，backlog 设置为 1
4. 创建 client socket
5. client 连接 server
6. server accept
7. server close
8. 返回 client 和 server accept 到的连接

但是目前发现连接有问题。原因是 accept 操作和 connnect 操作都是 block 的（lwip 不提供 non-block 的 accept 和 connect 操作），这样的话，就需要 kernel thread 才能完成。还没改。

6 syscall 层

syscall 层将所有接口以 syscall 的形式提供。主要工作是提取参数，然后根据参数和返回值情况决定是调用 socket wrap 还是直接调用 lwip 的 socket 函数。

目前除了 socketpair 函数实现还有问题，bionic 要求的 syscall 已经全部实现，列表如下：

- sys_bind
- sys_socket
- sys_recv
- sys_send
- sys_setsockopt

- sys_sockclose
- sys_listen
- sys_accept
- sys_shutdown
- sys_connect
- sys_sendto
- sys_recvfrom
- sys_getsockname
- sys_getpeername
- sys_sendmsg
- sys_recvmsg
- sys_socketpair

其中 bind, socket, recv, send, sockclose, listen, accept, connect, sendmsg, recvmsg, setsockopt, sendto, recv_from, getsockname, getpeername 目前都有程序测试并通过。

socketpair 的问题在于创建 socketpair 的时候，accept 操作和 connect 操作都是 block 的。但是 LWIP 的 connect 和 accept 都不支持 non-blocking 。目前的想法是用 kernel thread 去解决。尚未实验。

shutdown 函数未测试，因为 LWIP 没有实现半连接，shutdown 函数实际上与 close 函数效果一样(kern/lwip/api/socket.c 里的 lwip_shutdown 函数有说明)。

6.1 syscall 参数不够的问题

syscall 的 MAX_ARGS 目前设置为 5，但是 sys_sendto 和 sys_recvfrom 有 6 个参数，因此出现了一些问题。目前的解决办法是先丢掉其中的 flag 参数。以后有更好的解决方法的时候再改回去。

7 运行

7.1 qemu 版本

qemu 要使用 MIT 打过 patch 的版本。它修改了 e100 网卡的一些 bug，加上了 E100 的调试功能。

下载地址 http://pdos.csail.mit.edu/6.828/2010/tools.html

版本是 0.12.5

7.2 环境配置

环境配置的主要点在于 host 要支持 TAP networking。在我的机器上，执行 qemu -net tap 会提示说找不到网卡的错误。于是我就干脆把 bridge 配了，这样的话，从其他机器上也能访问 ucore 了。配置说明见 https://wiki.archlinux.org/index.php/QEMU 的 Tap Networking with QEMU 部分。

然后启动 qemu 的时候，要执行 qemu-ifup 脚本（在 trunk/src/etc 目录下面），这个脚本的作用就是 bring up tap0 和设置 TAP 的 IP，同时把 tap0 加到 bridge 里（如果没有配 bridge ，把 qemu-ifup 里面的 brctl addif 那句注释掉即可）。

7.3 IP 配置

前面已经提到，ucore 的网络配置在网卡驱动中 (kern/driver/e100.c 的 ether_e100_attach 函数)。目前的设置如下：ucore 的地址设置为 192.168.1.13, 网关为 192.168.1.1，子网掩码为 255.255.255.0

在 qemu-ifup 把 TAP 的地址设置为 192.168.1.12。

实际上，这个 host 中的 TAP 和 ucore 中的网络可以看作是连在一个交换机上。

配置完成后，在 host ping 192.168.1.13 应该可以 ping 通 ucore。

ucore 访问 host 则应该访问 192.168.1.12 地址。

如果还不行的话，可以试试在 host 上设置一下路由：

```
route add -net 192.168.1.13 netmask 255.255.255.255 gw 192.168.1.12
```

连接成功之后，在 ucore 上运行 tcpecho，在 host 上运行 telnet 192.168.1.13 就应该可以连通，输入任何字符回车就可以看到返回。

7.4 测试程序

现在已经有的测试程序和其功能列表：

1. tcpecho.c 能够接受一个 TCP 连接，并且把所有收到的数据都原样发回去。涉及到的函数有 socket, setsockopt, bind, accept, listen, recv, send, sockclose 测试通过。

2. print_server.c 和 tcpecho.c 类似，它只把所有收到的数据打印出来。同时，它也是 getsockname 和 getpeername 的测试程序。测试通过。

3. test_connect.c , 主要用来测试 connect 函数，另外还涉及 socket, send, sockclose, 测试通过。

4. sendmsg_test.c , 主要用来测试 sendmsg 函数，另外还涉及 socket, sockclose, connect, 测试通过。

5. recvmsg_test.c , 主要用来测试 recvmsg 函数，另外还涉及 socket, bind, accept, listen, sockclose，测试通过。

6. ftpclient.c , 网络原理实验的 FTP client 在 ucore 上面的移植。目前测试没有问题。

7. ftpserver.c , 网络原理实验的 FTP server 在 ucore 上面的移植。目前测试可用。但是在以下情形仍然有问题:
   - 在同时接受多个连接的时候，到了第 3 个链接就会卡死。经过追踪，发现问题可能与 e100 网卡驱动中的 3 个 semaphore 有关，logs/3_socket_bug.log 是调试这个问题的输出文件。
8. sendto_test.c 测试 UDP ，测试 sendto 函数。测试通过。
9. recvfrom_test.c 测试 recvfrom 函数。测试通过。
10. socketpair.c , 主要用来测试 socketpair 函数。测试不通过。

8 主要工作

8.1 移植胡刚之前的 ucore

主要改动如下:

- 修改 IDT，把 IDT 32 以上的描述符改成中断门。
- 增加 PIC 驱动，管理网卡的中断。
- 修改 syscall 里面的 sys_sem_wait, sys_sem_free, sys_mbox_free 函数名，因为和 LWIP 里面函数名冲突。
- x86.h 中增加 outl, inl, outw, inw 等函数。
- 修改 idleproc 初始化，增加 sem_queue
- 增加 PCI 驱动
- 增加 e100 网卡驱动
- 增加用户态接口。(user/libs/sockets.c)
- 增加 相应网络的 syscall 和 socket wrap
- proc_struct 中增加 lwip_timeouts 给 lwip 存储 timeouts 列表。
- 增加 LWIP

8.2 之前的 Bug

8.2.1 初始化问题

他的 sys_arch_timeouts 函数里面，lwip_timeouts 分配空间之后，其中的 next 指针没有初始化，在特定情况下导致了一些 bug。增加初始化代码后问题解决。

8.2.2 调度器导致的问题

TCP/IP 初始化的时候，开启了一个 tcpip 的 kernel thread，结果导致调度出了问题。最明显的表现是使用 ls 命令来列比较大的目录时，如 bin 目录，就会在中间 schedule 出去永远回不来。

我把这个 kernel thread 注释掉，然后就不存在这个问题。

后来把 MLFQ 调度器换成了 RR 调度器，这个问题也不存在。就一直使用 RR 调度器了。

8.3 增加 syscall

增加了 8 个 socket syscall

```
int sendmsg(int, const struct msghdr *, unsigned int)
int recvmsg(int, struct msghdr *, unsigned int)
int socketpair(int, int, int, int*)
int connect(int, struct sockaddr *, socklen_t)
int getsockname(int, struct sockaddr *, socklen_t *)
int sendto(int, const void *, size_t, int,
           const struct sockaddr *, socklen_t)
int recvfrom(int, void *, size_t, unsigned int,
             struct sockaddr *, socklen_t *)
int getpeername(int, struct sockaddr *, socklen_t *)
```

8.4 编写网络测试

现在共有 10 个测试程序。列表及功能见上面 测试程序一节。其中 tcpecho.c 是胡刚写的，其余 9 个是我写的。

8.5 移植 FTP

移植的 FTP 来自于计算机网络原理实验的 FTP 实验。

8.5.1 为移植 FTP 增加的一些函数

- fgets 获取用户输入
- atoi 将 字符串转整数
- isdigit 判断一个字符是否为数字

8.5.2 为移植 FTP 做的其他改动

- 与系统相关的一些命令，如 ls, pwd，本来是用 popen 实现的，移植到 ucore 后，采用写文件的方式，然后再读出来解决。

8.5.3 FTP 已知的问题

- ftpserver 的 DELE 命令执行过后，LIST 命令就会显示不正常。没有调查为什么。
- ftpserver 接受 2 个 client 连接之后，再开需要 data connection 的命令就会卡住，schedule 不回来。现象看起来是个活锁。

8.6 增加注释

为自己的程序和之前的胡刚的程序增加的一些注释。

8.7 e1000 网卡驱动编写

写了一半。现在已经完成了基本配置，能够读取 MAC 地址，加入 LWIP 网卡链表。还缺发送和接收函数和中断处理函数。

8.8 Virtual Box 自动启动

写了一个脚本能够自动创建 ucore 的 Virtual Box 的 OS。脚本文件为 create_ucore_vm.sh

9 一些问题

9.1 文件系统关闭问题

ucore 没有关机，所以文件系统在更改过后，如果不重新生成 fs.img 文件，再次使用就会出 panic，如下:

```
kernel panic at kern/fs/sfs/sfs_inode.c:244:
    assertion failed: ino == 0 || sfs_block_inuse(sfs, ino)
```

典型的重现过程如下:

1. 启动 ucore

   ```
   cd test
   echo 1 > testfile
   cat testfile
   ```

2. 关闭 qemu

3. 再次启动 ucore

   ```
   cd test
   cat testfile
   ```

后来我修改了 Makefile，把 $(FSIMG) 设置为 PHONY ，避过了这个问题