Fall 2005 Handout 4

Startup scripts

(1) A running program is called a *process.* Each one has a ''process identification number'', or *PID.* For example, the kernel is dressed up to look like a process and is given PID number 0. When the computer is *booted* (turned on), the kernel gives birth to exactly one child, named **`init`**, whose PID number is 1:

```
1$ ps -Af | awk 'NR == 1 || $2 == 0 || $2 == 1'
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   May 15 ?        0:04 sched
    root     1     0  0   May 15 ?       58:21 /etc/init -v
```

(2) A newly-booted computer first awakens into *single user mode* and then into *multi-user mode.* After that, everybody can log in. For the eight *run levels* (modes) in our version of Unix, see **`init`**(1M), **`inittab`**(4), and **`shutdown`**(1M) Most of the time, we're in **`S`**, **`2`**, or **`3`**.

`S or s`: single user
`2`: multi-user
`3`: multi-user, making local resources available over the network

`0`: go into firmware
`1`: system administrator mode for installing packages
`4`: alternative multi-user environment, ''usually not used''
`5`: shut down
`6`: reboot

The file **`/etc/inittab`** tells the **`init`** process what shellscript to run when the system enters each *run level.* For example, when **`init`** enters run level **`S`** it will execute the shellscript **`/sbin/rcS`**. **`sbin`** stands for ''system binaries''; the uppercase **`S`** in **`rcS`** stands for ''single user''. The **`rc`** in the name of many Unix startup scripts stands for ''run command''. It's taken from F. J. Corbató's Compatible Time-Sharing System CTSS (MIT, 1962), the first time sharing operating system.

```
2$ man -s 4 inittab                    Only Solaris needs the -s.
3$ awk -F: '$2 == "s"' /etc/inittab
sS:s:wait:/sbin/rcS          >/dev/msglog 2<>/dev/msglog </dev/console

4$ cd /sbin
5$ ls -l rc* | more
-rwxr--r--   3 root     sys         2792 Apr  6  2002 rc0
-rwxr--r--   1 root     sys         3177 Apr  6  2002 rc1
-rwxr--r--   1 root     sys         2922 Apr  6  2002 rc2
-rwxr--r--   1 root     sys         2403 Apr  6  2002 rc3
-rwxr--r--   3 root     sys         2792 Apr  6  2002 rc5
-rwxr--r--   3 root     sys         2792 Apr  6  2002 rc6
-rwxr--r--   1 root     sys         9934 Apr  6  2002 rcS
```

(3) The shellscript **`/sbin/rcS`** runs all the shellscripts starting with uppercase **`S`** in the **`/etc/rcS.d`** directory. Scripts in the **`/etc/rc*.d`** directories that start with **`S`** are run when the

computer is started, i.e., when the system comes up into the runlevel of the directory. Scripts that start with **K** are run when the computer is killed, i.e., when the system comes down into the runlevel of the directory.

```
7$ cd /etc
8$ ls -ld rc*.d | more
drwxr-xr-x   2 root     sys         1024 Feb 20  2005 rc0.d
drwxr-xr-x   2 root     sys         1024 May 14  2004 rc1.d
drwxr-xr-x   2 root     sys         1536 Feb 20  2005 rc2.d
drwxr-xr-x   3 root     sys          512 Feb 20  2005 rc3.d
drwxr-xr-x   2 root     sys         1024 May 14  2004 rcS.d
```

The shellscript **/sbin/rcS** is written in the language of the Bourne shell, not the language of the Korn shell. It therefore has only one pair of **[**square brackets**]** around the logical expression of the **if** statements in lines 343 and 345. The **-d** in line 343 tests if the following directory exists; the **-s** in line 345 tests if the following file is not empty. For these and other goodies within the square brackets, go to

http://i5.nyu.edu/~mm64/man/

and print out the manual page for **test**(1).

The last dot in line 347 executes a shellscript by the current shell, not a subshell. This is how a shellscript that creates environment variables must be run. If the environment variables had been created by a subshell, they would have disappeared when the subshell died. In Unix, environment variables are passed from a process to its descendants, not from a process to its ancestors. See p. 7 of the manual page for **sh**(1).

```
9$ cat -n /sbin/rcS | sed -n '1p;343,352p'
     1  #!/sbin/sh
   343      if [ -d /etc/rcS.d ]; then
   344          for f in /etc/rcS.d/S*; do
   345              if [ -s $f ]; then
   346                  case $f in
   347                      *.sh)   .    $f ;;
   348                      *)  /sbin/sh $f start ;;
   349                  esac
   350              fi
   351          done
   352      fi
```

(4) The **/etc/rcS.d/S*** in the above line 344 loops through the **S** shellscripts in alphabetical order. But in the **/etc/rcS.d** directory, every **S** shellscript has exactly two digits after the **S**. Since all the names have the same number of digits, alphabetical order is the same as ascending numerical order:

```
10$ cd /etc/rcS.d
11$ ls S* | pr -3 -l4          minus lowercase L
S30network.sh           S35svm.init             S50devfsadm
S30rootusr.sh           S40standardmounts.sh    S70buildmnttab.sh
S33keymap.sh            S41cachefs.root         S95picld
S35cacheos.sh           S42coreadm
```

```
12$ ls K* | pr -3 -l13 -t            minus lowercase L
K03samba                K28kdc.master           K40nscd
K03sshd                 K28nfs.server           K40slpd
K05appserv              K33audit                K40syslog
K05volmgt               K34IIim                 K40xntpd
K06mipagent             K34ncalogd              K41autofs
K07dmi                  K34svm.sync             K41directory
K07snmpdx               K36sendmail             K41ldap.client
K10dtlogin              K36utmpd                K41rpc
K15imq                  K36wbem                 K42inetsvc
K16apache               K37power                K43inet
K21dhcp                 K39lp                   K50pppd
K27boot.server          K39spc                  K52llc2
K28kdc                  K40cron
```

(5) As you just saw, one of the scripts in the directory **/etc/rcS.d** is named **S30network.sh**. It and all the other **S** and **K** scripts for each runlevel are hardlinked to the directory **/etc/init.d** so that you can see them all in one place. They are also hardlinked to one of the **/etc/rc*.d** directories to show the runlevel(s) at which they are executed and the order in which they are executed within each runlevel.

```
13$ cd /etc/rcS.d
14$ ls -li S30network.sh                                        inode number
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 S30network.sh

    15$ find /etc -inum 5832 -print
    /etc/init.d/network
    /etc/rcS.d/S30network.sh

16$ ls -li ‘find /etc -inum 5832 -print‘
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/init.d/network
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/rcS.d/S30network.sh
```

▼ Homework 4.1: at what runlevel is each server launched on i5.nyu.edu?

Use **grep** to examine all the scripts in the directory **/etc/init.d** to find the one that launches the secure shell server **sshd**. When you have found it, use **find** to find the other directories that hold this script, and the name that the script has in each directory. Look for a name that starts with **S**, not **K**. At what runlevel is **sshd** launched? At what runlevel are the following servers launched? If two or more servers are launched at the same runlevel, what order are they launched in?

sshd	*Secure Shell dæmon*
httpd	*Hypertext Transport Protocol dæmon*
in.routed	*routing dæmon*
in.named	*DNS dæmon*
inetd	*Internet dæmon*
sendmail	*mail dæmon*

▲

The dæmonic (DÆMONIC) ancestry of a process

Read from the bottom up to see the ancestry of a Korn Shell process. It may be different on other versions of Unix. **STIME** is starting time; **TIME** is cumulative elapsed time. The **C** field is obsolete. **sched** is the kernel. **sshd** is the Secure Shell dæmon. The dash in front of **ksh** means that this **ksh** is the shell that was launched when you logged in, not a shell that's being used to run a shellscript.

```
1$ ps -Af | more                          output doctored by hand
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   May 15 ?        0:04 sched
    root     1     0  0   May 15 ?       58:21 /etc/init -v
    root   430     1  0   May 15 ?       11:56 /usr/local/sbin/sshd
    root 28483   430  0 08:53:08 ?        0:00 /usr/local/sbin/sshd
    mm64 28487 28483  0 08:53:12 ?        0:11 /usr/local/sbin/sshd
    mm64 28489 28487  0 08:53:12 pts/6    0:01 -ksh
```

▼ Homework 4.2: print out the ancestry of your shell

Direct the output of **`ps -Af`** into a file. Use **`vi`** to remove all of the processes except your shell and all of its ancestors. Reorder these processes in order of increasing PID number, as in the above example.

```
1$ cd
2$ pwd

3$ ps -Af > temp
4$ vi temp                                Doctor the output by hand.

5$ lpr temp
6$ rm temp                    or use the escape-. in X52.9545 Handout 2, p. 11
```

▲

inetd: the Internet dæmon

A *dæmon* runs forever and has no controlling terminal. See pp. 129−133. I had to get the **`TTY`** with **`substr`** instead of **`$7`** because the **`STIME`** is not always two words.

```
1$ ps -Af | awk 'NR == 1 || substr($0, 34, 1) == "?"' | head -15
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   May 15 ?        0:04 sched
    root     1     0  0   May 15 ?       58:21 /etc/init -v
    root     2     0  0   May 15 ?        0:32 pageout
    root     3     0  1   May 15 ?       10022:32 fsflush
    root    57     1  0   May 15 ?        0:00 /usr/lib/sysevent/syseventd
    root    65     1  0   May 15 ?        0:01 /usr/lib/picl/picld
    root   150     1  0   May 15 ?        0:00 /usr/sbin/rpcbind
    root   173     1  0   May 15 ?        0:01 /usr/sbin/inetd -s
    root   193     1  0   May 15 ?        8:59 /usr/sbin/syslogd
    root   195     1  0   May 15 ?       24:32 /usr/sbin/cron
    root   214     1  0   May 15 ?        0:03 /usr/lib/lpsched
    root   234     1  0   May 15 ?        0:29 /usr/lib/utmpd
   smmsp   238     1  0   May 15 ?        0:00 /usr/lib/sendmail -Ac -q15m
    root   261     1  0   May 15 ?        0:03 /usr/lib/inet/xntpd
```

Are we running the original **`inetd`** or the extended **`xinetd`**?

```
2$ ps -Af | awk 'NR == 1 || /inetd/'
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root   173     1  0   May 15 ?        0:01 /usr/sbin/inetd -s
```

`/etc/rc2.d/S72inetsvc` is also known as **`/etc/init.d/inetsvc`**.

```
3$ cd /etc
4$ ls -li rc2.d/S72inetsvc
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 rc2.d/S72inetsvc

5$ find . -inum 392 -print
./init.d/inetsvc
./rc0.d/K42inetsvc
./rc1.d/K42inetsvc
./rc2.d/S72inetsvc
./rcS.d/K42inetsvc

6$ ls -li 'find . -inum 392 -print'
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 ./init.d/inetsvc
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 ./rc0.d/K42inetsvc
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 ./rc1.d/K42inetsvc
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 ./rc2.d/S72inetsvc
392 -rwxr--r--   5 root     sys         7353 Mar 18  2004 ./rcS.d/K42inetsvc
```

For the Service Access Facility (SAF), see the **-s** option of **inetd** in **inetd**(1M).

```
7$ cat -n /etc/rc2.d/S72inetsvc | sed -n '263,$p'
   263  #
   264  # Run inetd in "standalone" mode (-s flag) so that it doesn't have
   265  # to submit to the will of SAF.  Why did we ever let them change inetd?
   266  #
   267  /usr/sbin/inetd -s
```

Does **inetd** have any children?

```
8$ ps -Af | awk 'NR == 1 || $2 == 173 || $3 == 173' | head
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root   173     1  0   May 15 ?        0:01 /usr/sbin/inetd -s
```

The above login names give the *effective user ID* of each process, which tells what the process has permission to do. The *real user ID* of each process tells who owns it. Sometimes they're different:

Most of the lines in our **/etc/inetd.conf** are commented out:

```
9$ awk '/^#+ *(echo|daytime|ftp|telnet)/' /etc/inetd.conf
## echo    stream tcp6 nowait root internal
## echo    dgram  udp6 wait   root internal
## daytime stream tcp6 nowait root internal
## daytime dgram  udp6 wait   root internal
## telnet  stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd
## ftp     stream tcp6 nowait root /usr/sbin/in.ftpd    in.ftpd-a
```

What's not commented out? Let's look for lines where the first non-whitespace character is not a **#**. Inside the first pair of **[**square brackets**]**, there is one blank and one tab.

```
10$ grep '^[   ]*[^#]' /etc/inetd.conf
100134/1    tli    rpc/ticotsord wait   root /usr/lib/krb5/ktkt_warnd      ktkt_warnd
bpcd        stream tcp           nowait root /usr/openv/netbackup/bin/bpcd bpcd
vnetd       stream tcp           nowait root /usr/openv/bin/vnetd          vnetd
vopied      stream tcp           nowait root /usr/openv/bin/vopied         vopied
bpjava-msvc stream tcp           nowait root /usr/openv/netbackup/bin/bpjava-msvcbpjava
```

Because of the following line in **/etc/nsswitch.conf**, the **inetd** looks up the port number of each service in the file **/etc/services**. See Handout 2, p. 17.

```
11$ awk '$1 == "services:"' /etc/nsswitch.conf
services:   files
```

Inside the **[**square brackets**]**, there is one blank and one tab.

```
12$ awk '$1 ~ /^[    ]*(echo|daytime|ftp|telnet|talk|finger)/' /etc/services
echo      7/tcp
echo      7/udp
daytime   13/tcp
daytime   13/udp
ftp-data  20/tcp
ftp       21/tcp
telnet    23/tcp
finger    79/tcp
talk      517/udp
```

Attach a new host to an existing network (without using DHCP)

(1) Give the host a hostname. The name of our host is **i5**.

(2) Give the host an IP address. The IP address of our host is **128.122.253.152**.

(3) It is not necessary to give a name to a network interface: it already has a name derived from its chipset. The names of our interfaces are **lo0** (loopback) and **ge0** (Gigabit Ethernet). But the host's interfaces must be configured with the program **ifconfig**.

(4) Give the host a default route with **route**.

(5) Point the host to a DNS domain name server.

(6) Configure NIS to tell the host where to get information from: for example, local files such as **/etc/hosts** vs. DNS.

The configuration files

Human beings usually start counting at ''one'', but **sort** believes that the fields are numbered starting with zero. The **+4n** option therefore directs **sort**'s attention to the file sizes in what we could call ''column five''. As we will see, **notrouter** only has to exist. It doesn't have to contain anything.

```
1$ cd /etc
2$ ls -l | tail +2 | grep '^-' | sort +4n | head
-rw-r--r--   1 root     other          0 Oct  7  2003 notrouter
-rw-rw-r--   1 root     sys            0 Aug 18  2003 dumpdates
-rw-r--r--   1 root     root          11 Jan  9  2004 hostname.ge0
-rw-r--r--   1 root     root          11 Jan  9  2004 nodename
-rw-r--r--   1 root     other         16 Dec 19  2003 defaultrouter
-rw-r--r--   1 root     other         38 May 13  2003 scrollkeeper.conf
-rw-r--r--   1 root     sys           39 Aug 18  2003 ioctl.syscon
-rw-r--r--   1 root     bin           50 Aug 18  2003 auto_home
-rw-r--r--   1 root     other         77 May 13  2003 esd.conf
-rw-r--r--   1 root     sys           78 Aug 18  2003 project
```

Set the hostname.

i5.nyu.edu knows its own name:

```
1$ /sbin/uname -a
SunOS i5.nyu.edu 5.9 Generic_112233-12 sun4u sparc SUNW,Ultra-Enterprise
```



```
2$ /sbin/uname -X
System = SunOS
Node = i5.nyu.edu
Release = 5.9
KernelID = Generic_112233-12
Machine = sun4u
BusType = <unknown>
Serial = <unknown>
Users = <unknown>
OEM# = 0
Origin# = 1
NumCPU = 4
```

You have to create the file **/etc/nodename**:

```
3$ cd /etc
4$ ls -l nodename
-rw-r--r--   1 root     root          11 Jan  9  2004 nodename

5$ cat nodename
i5.nyu.edu
```

The above file is read by **/etc/init.d/network** (a.k.a. **/etc/rcS.d/S30network.sh**) when the machine starts up.

```
6$ cd /etc/rcS.d
7$ ls -li S30network.sh                                    inode number
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 S30network.sh

8$ find /etc -inum 5832 -print
/etc/init.d/network
/etc/rcS.d/S30network.sh

9$ ls -li 'find /etc -inum 5832 -print'
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/init.d/network
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/rcS.d/S30network.sh
```

Here are excerpts from **/etc/init.d/network**. **-z** is true if the following string is of length zero; see **test**(1).

```
1 hostname='cat /etc/nodename'
2
3 if [ -z "$hostname" ]
4 then
5     hostname=unknown
6 fi
7
8 /sbin/uname -S $hostname            #Only the superuser can give -S
9 echo Hostname: '/sbin/uname -n'
```

Set the IP address.

```
1$ cd /etc
2$ ls -l hosts
lrwxrwxrwx   1 root     root          12 Aug 18  2003 hosts -> ./inet/hosts

3$ cd ./inet
4$ ls -l hosts
-r--r--r--   1 root     sys          128 Jan  9  2004 hosts

5$ cat -n /etc/hosts | more
     1  #
     2  # Internet host table
     3  #
     4  127.0.0.1   localhost
     5  ## 128.122.253.152   i6.home.nyu.edu  loghost
     6  128.122.253.152  i5.nyu.edu   i5 loghost
```

The file **`/etc/hosts`** will be read by the **`/etc/rcS.d/S30network.sh`** shellscript to find the IP address of i5.nyu.edu . **`S30network.sh`** will then give the IP address to an **`ifconfig`** command.

See the network interfaces.

The maximum transmission unit for the loopback interface is 8232 = 8 × 1024 + 20 + 20. This allows 8K of data, plus the 20-byte IP header (p. 14), plus the 20-byte TCP header (p. 19). These headers are 20 bytes each if they include no options.

```
1$ /sbin/ifconfig -a                    "all"
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
    inet 127.0.0.1 netmask ff000000
ge0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
    inet 128.122.253.152 netmask ffffffc0 broadcast 128.122.253.191
```

Let's translate the interface flags from hexadecimal to binary:

```
2$ bc
obase=2                          Specify the output base before the input base!
ibase=16
1000849
1000000000000100001001001       Five flags are on.
1000843
1000000000000100001000011       Five flags are on.
control-d
3$
```

The C header file **`/usr/include/net/if.h`** has one **`IFF_`** macro for each interface flag. The **`[`**square brackets**`]`** contain one blank and one tab.

```
4$ egrep \
'#define[    ]*IFF_(UP|BROADCAST|LOOPBACK|RUNNING|MULTICAST|IPV4|PROMISC)' \
/usr/include/net/if.h
#define IFF_UP        0x000000001/* interface is up */
#define IFF_BROADCAST 0x000000002/* broadcast address valid */
#define IFF_LOOPBACK  0x000000008/* is a loopback net */
#define IFF_RUNNING   0x000000040/* resources allocated */
#define IFF_PROMISC   0x000000100/* receive all packets */
#define IFF_MULTICAST 0x000000800/* supports multicast */
#define IFF_IPV4      0x001000000/* IPv4 interface */
```

On my Caldera Linux, connected to PPP but not to Ethernet, the output of **`ifconfig -a`** is

```
lo  Link encap:Local Loopback
    inet addr:127.0.0.1  Mask:255.0.0.0
    UP LOOPBACK RUNNING  MTU:3924  Metric:1
    RX packets:54 errors:0 dropped:0 overruns:0 frame:0
    TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0

eth0Link encap:Ethernet  HWaddr 00:40:05:43:1C:57
    unspec addr:[NONE SET]  Bcast:[NONE SET]  Mask:[NONE SET]
    BROADCAST MULTICAST  MTU:1500  Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:100
    Interrupt:10 Base address:0xfc80

ppp0Link encap:Point-to-Point Protocol
    inet addr:216.165.4.189  P-t-P:216.165.0.18  Mask:255.255.255.255
    UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
    RX packets:12762 errors:2 dropped:0 overruns:0 frame:2
    TX packets:19311 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:10
```

Underneath Mac OS X,

```
ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    inet6 ::1 prefixlen 128
    inet6 fe80::1 prefixlen 64 scopeid 0x1
    inet 127.0.0.1 netmask 0xff000000

gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    inet6 fe80::20a:95ff:fec2:e6bc prefixlen 64 scopeid 0x4
    inet 192.168.20.230 netmask 0xffffff00 broadcast 192.168.20.255
    ether 00:0a:95:c2:e6:bc
    media: autoselect (10baseT/UTP <half-duplex>) status: active
    supported media: none autoselect 10baseT/UTP <half-duplex>
10baseT/UTP <full-duplex> 10baseT/UTP <full-duplex,hw-loopback>
100baseTX <half-duplex> 100baseTX <full-duplex>
100baseTX <full-duplex,hw-loopback> 1000baseTX <full-duplex>
1000baseTX <full-duplex,hw-loopback> 1000baseTX <full-duplex,flow-control>
1000baseTX <full-duplex,flow-control,hw-loopback>

fw0: flags=8822<BROADCAST,SMART,SIMPLEX,MULTICAST> mtu 4078
    lladdr 00:0a:95:ff:fe:c2:e6:bc
    media: autoselect <full-duplex> status: inactive
    supported media: autoselect <full-duplex>
```

On i5.nyu.edu,


```
5$ /bin/netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      3281335 0     3281335 0     0      0
ge0   1500 i5.nyu.edu    i5             309586338 0     366806801 0     0      0
```

The **-n** option (for ''numeric'') shows the IP addresses instead of the fully qualified domain names. Use this early in the booting process before DNS is up:

```
6$ /bin/netstat -in
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 127.0.0.0     127.0.0.1      3281335 0     3281335 0     0      0
ge0   1500 128.122.253.128 128.122.253.152 309586347 0     366806818 0     0      0
```

Configure the network interfaces

The network interfaces are configured with **ifconfig** commands in the file

```
1$ cd /etc/init.d
2$ ls -li network                                                    inode number
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 network

    3$ find /etc -inum 5832 -print
    /etc/init.d/network
    /etc/rcS.d/S30network.sh

4$ ls -li 'find /etc -inum 5832 -print'
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/init.d/network
5832 -rwxr--r--   2 root     sys        20336 Jan 13  2004 /etc/rcS.d/S30network.sh

5$ cd /etc/init.d
6$ cat -n network | sed -n 558,562p
   558  #
   559  # Configure the software loopback driver. The network initialization is
   560  # done early to support diskless and dataless configurations.
   561  #
   562  /sbin/ifconfig lo0 plumb 127.0.0.1 up
```

/etc/init.d/network uses the following one-word file to find the name of our host:

```
8$ ls -l /etc/hostname.*[0-9] | more
-rw-r--r--   1 root     root          11 Jan  9  2004 hostname.ge0

9$ cat /etc/hostname.ge0              "Gigabit Ethernet"
i5.nyu.edu
```

/etc/init.d/network also looks without success for the following files:

```
10$ ls -l /etc/hostname6.*[0-9] | more
hostname6.*[0-9]: No such file or directory

11$ ls -l /etc/dhcp.*[0-9] | more
dhcp.*[0-9]: No such file or directory
```

Here's a simplified version of what **/etc/init.d/network** does after configuring the loopback interface. The first line puts the string **hostname.ge0** into the variable **$interface_name**.

```
1 interface_name=‘echo /etc/hostname.*[0-9]‘
2 IFS=$IFS.                              Add dot to the shell input field separators.
3 set $interface_name                    Copy hostname into $1, ge0 into $2.
4 shift                                  Copy ge0 into $1.
5 /sbin/ifconfig $1 plumb
6 ifcmds=’‘cat /etc/hostname.ge0‘ netmask + broadcast + up’
7 /sbin/ifconfig $1 inet $ifcmds 2>&1 > /dev/null
```

The **ifconfig** command in the above line 7 is therefore

```
/sbin/ifconfig ge0 inet i5.nyu.edu netmask + broadcast + up 2>&1 > /dev/null
```

It looks up the IP address of **i5** in the file **/etc/hosts**. See p. 143.

```
12$ awk ’$3 == "i5"’ /etc/hosts
128.122.253.152  i5.nyu.edu  i5 loghost
```

Now that we know that the IP address of our host is **128.122.253.152**, the **netmask +** and **broadcast +** tells **ifconfig** to get the netmask and broadcast address from the file **/etc/netmasks**. It looks for the network address in column 1 that matches the most network bits of **128.122.253.152**. In fact, column 1 has an address that matches all the network bits of **128.122.253.152**. See pp. 143−145.

```
13$ awk ’$1 == "128.122.253.128"’ /etc/netmasks
128.122.253.128 255.255.255.192
```

We saw that the address of our network is **128.122.253.128** in Handout 1, p. 21.

The environment variables

```
_INET_NET_IF
_INET_NET_STRATEGY
```

used in **/etc/init.d/network** (a.k.a **/etc/rcS.d/S30network.sh**) were created by the shellscript **/sbin/rcS** listed in the file **/etc/inittab**; see Handout 4, p. 1. The (non-environment) variable

```
TRACK_INTERFACES_ONLY_WITH_GROUPS
```

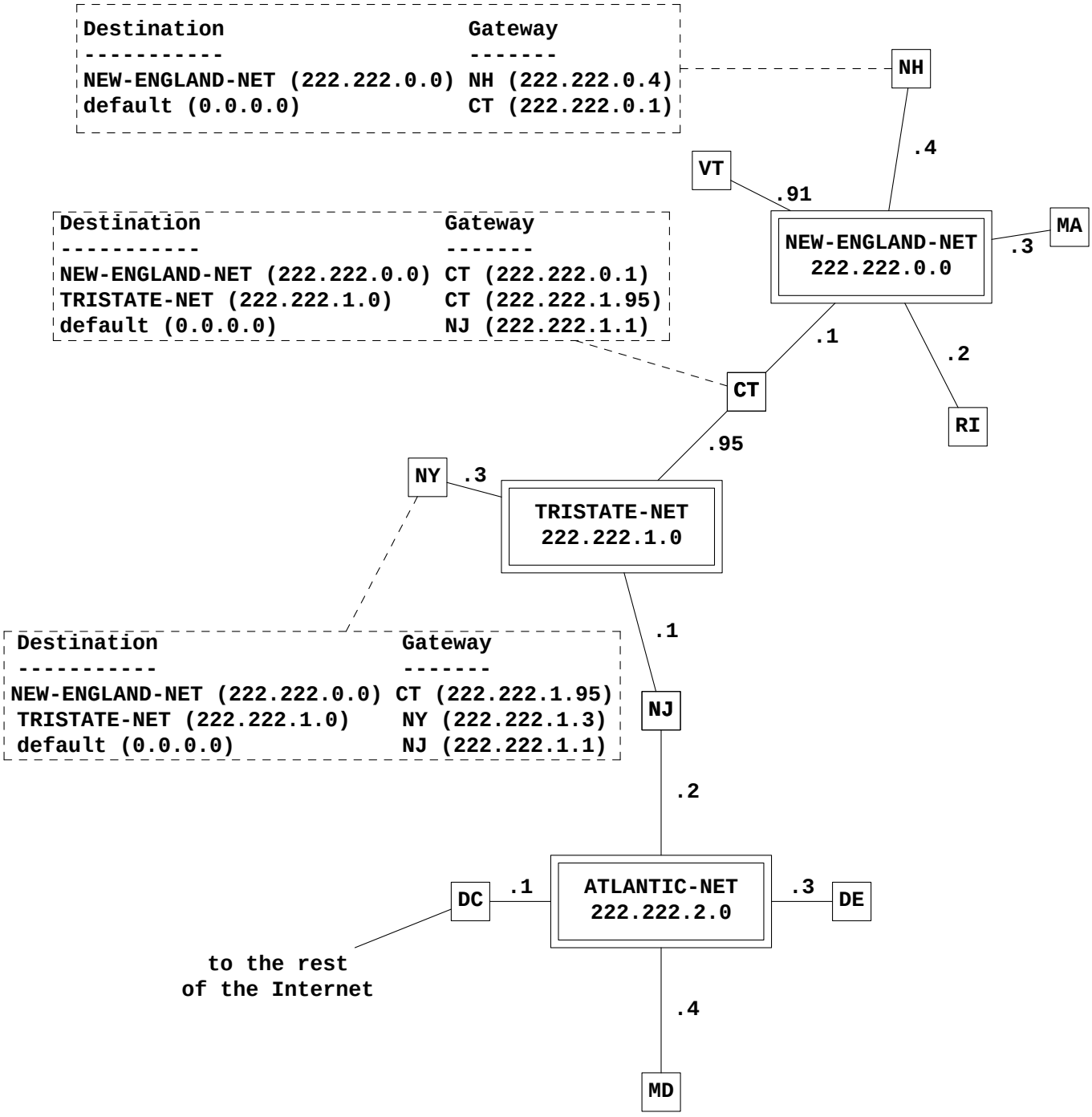
used in **/etc/init.d/network** was created when that shellscript ran **/etc/default/mpathd**.

Gateways between networks

All of the following IP addresses are class C addresses. In the absence of an explicit netmask, we will therefore assume that the first 24 bits of each IP address identifies the network, and the last 8 bits identifies the host. For example, on the New England Network the IP address of Massachusetts is **222.222.0.3**, and the IP addresses of Connecticut is **222.222.0.1**. But Connecticut is the gateway to New England. It therefore has an additional IP address, **222.222.1.95**, because from the Tristate area you take Route I-95 to New England.

Note that New York’s routing table is bigger than New hampshire’s, even though neither of them are gateways. This is because the New England Network is a *stub* network: a cul de sac.

To avoid cluttering the diagram, we do not show the loopback address (**127.0.0.1**) or the multicast address (**240.0.0.0**) in the routing tables.



See the routing table:
pp. 35–43, 170–204

The **-n** option (for ‘‘numeric’’) will display the IP address instead of the fully qualified domain names:


```
1$ netstat -nr | more
Routing Table: IPv4
  Destination           Gateway           Flags  Ref   Use   Interface
-------------------- -------------------- ----- ----- ------ ---------
128.122.253.128      128.122.253.152      U         1   7498  ge0
224.0.0.0            128.122.253.152      U         1      0  ge0
default              128.122.253.129      UG        1 251695
127.0.0.1            127.0.0.1            UH        33252997  lo0
```

If your DNS server is up and running, you can ask to see the fully qualified domain names:

```
2$ netstat -r | more
Routing Table: IPv4
Destination           Gateway                   Flags  Ref   Use   Interface
-------------------   --------------------      ----- ----- ------ ---------
NYU-FDDI253-128-NET   i5                        U         1   7498  ge0
BASE-ADDRESS.MCAST.NET i5                       U         1      0  ge0
default               WWITSGW-VLAN-13.NET.NYU.EDUUG       1 251695
localhost             localhost                 UH        33252997  lo0
```

Create the routing table

The first and fourth entries in the above routing table were created by the **`ifconfig`** commands in **`/etc/init.d/network`** (Handout 4, p. 11). Another shellscript executed at startup is **`/etc/init.d/inetinit`**:

```
1$ ls -li /etc/init.d/inetinit /etc/rc2.d/S69inet
5830 -rwxr--r--   5 root     sys        12655 Jan 13  2004 /etc/init.d/inetinit
5830 -rwxr--r--   5 root     sys        12655 Jan 13  2004 /etc/rc2.d/S69inet

2$ more /etc/defaultrouter
128.122.253.129                    six hostbits are 000001
```

After an unusual amount of agony, **`/etc/init.d/inetinit`** does the following. The **`-n`** option of **`/usr/sbin/route`** tells it not to attempt to use DNS, which might not be installed at this early time.

```
 1 #Simplified excerpt from /etc/init.d/inetinit
 2
 3 if [ -f /etc/defaultrouter ]; then
 4     defrouters=‘grep -v ’^#’ /etc/defaultrouter | awk ’{print $1}’‘
 5
 6     if [ -n "$defrouters" ]; then   #if the string is not empty
 7         for router in $defrouters; do
 8             /usr/sbin/route -n add default $router
 9         done
10     fi
11 fi
```

traceroute: pp. 451−454

I’m ssh’ing to i5.nyu.edu from a Mac at the Education Building:

```
1$ who | more
mm64       pts/5        Oct  7 09:46    (ftcg5faculty2.edlab.its.nyu.edu)
```

```
2$ /usr/sbin/nslookup ftcg5faculty2.edlab.its.nyu.edu | awk 'NR == 5'
Address:  192.168.20.196
```

Let's verify this on the Mac:

```
Apple menu
System Preferences...
Network
TCP/IP
IP 192.168.20.196
Router 192.168.20.1
```

traceroute tries to send a UDP packet to port 33434 of the specified host. (See the **-p** option of **traceroute**). But there is no program listening to port 33434 on that host: all we'll get is an error message of type **ICMP_UNREACH**, carried back to us via ICMP.

Furthermore, most of the UDP packets will never even reach the specified host: each of the first three UDP packets are sent out in an IP packet whose TTL (time to live, Handout 2, p. 6, line 58) is 1, so they never get more than one hop away from the starting host. ICMP error messages of type **ICMP_TIMXCEED** will be sent back, and **traceroute** will display the name of the sender. The next three UDP packets are sent out with a TTL of 2, so a host that is two hops away will send back the error messages.

```
3$ traceroute ftcg5faculty2.edlab.its.nyu.edu | more
traceroute to ftcg5faculty2.edlab.its.nyu.edu (192.168.20.196), 30 hops max, 40 byt
 1  WWITSGW-VLAN-13.NET.NYU.EDU (128.122.253.129)  0.448 ms  0.307 ms  0.301 ms
 2  COREGWD-WP-GEC-4.NET.NYU.EDU (128.122.1.34)  0.440 ms  0.356 ms  0.353 ms
 3  GODDARDGW-FE-6-0-1.NET.NYU.EDU (128.122.1.104)  0.834 ms  0.496 ms  0.462 ms
 4  FTCG5FACULTY2.EDLAB.ITS.NYU.EDU (192.168.20.196)  0.543 ms  0.516 ms  0.435 ms
```

On **ftcg5faculty2.edlab.its.nyu.edu**,

```
% traceroute i5.nyu.edu
traceroute to i5.nyu.edu (128.122.253.152), 30 hops max, 40 byte packets
 1  goddardgw-ether-1-2.nyu.net (192.168.20.1)  3.156 ms  0.452 ms  0.335 ms
 2  coregwd-wp-vlan904.net.nyu.edu (128.122.1.97)  0.459 ms  0.455 ms  0.436 ms
 3  wwitsgw-ge-3-1.net.nyu.edu (128.122.1.7)  0.511 ms  0.485 ms  0.426 ms
 4  i5.nyu.edu (128.122.253.152)  0.483 ms  0.563 ms  0.466 ms
```

Therefore **128.122.253.129** and **128.122.1.7** are two interfaces on the same gateway. **128.122.253.129** is the interface that's closest to us; **128.122.1.7** is the interface that's closest to **ftcg5faculty2.edlab.its.nyu.edu**.

On i5.nyu.edu ,

```
4$ traceroute 128.122.1.7
traceroute to 128.122.1.7 (128.122.1.7), 30 hops max, 40 byte packets
 1  WWITSGW-VLAN-13.NET.NYU.EDU (128.122.253.129)  2.275 ms *  0.434 ms
```

▼ Homework 4.3: traceroute (not to be handed in)

Trace the route from **i5.nyu.edu** (**128.122.253.152**) to a few other hosts, including a far-away host like **www.urz.uni-heidelberg.de** (The University of Heidelberg, **129.206.218.89**). Does the route always begin with **WWITSGW-VLAN-13.NET.NYU.EDU** (**128.122.253.129**)?

On another host, run **traceroute** to trace the route back to **i5.nyu.edu**. Does the route always end by going through **WWITSGW-VLAN-13.NET.NYU.EDU** (**128.122.253.129**)? How could it not?
▲

Turn on routed

`/etc/init.d/inetinit` (a.k.a **`/etc/rc2.d/S69inet`**) will run routed only if there is no default router, two or more network interfaces, and no file **`/etc/notrouter`**. The **`grep`** is needed because each interface listed by **`ifconfig`** occupies two lines.

```
 1 if [ -z "$defrouters" ]; then       #if the variable $defrouters is empty
 2     inetifaddrs="‘/usr/sbin/ifconfig -a4u | /usr/bin/grep inet‘"
 3     numifs=‘echo "$inetifaddrs" | /usr/bin/wc -l‘
 4
 5     if [ ! -f /etc/notrouter -a $numifs -gt 2 ]; then
 6         /usr/sbin/in.routed
 7     else
 8         /usr/sbin/in.routed -q
 9     fi
10
11 fi
```

Tow ways to create an empty file,

```
1$ cd /etc
2$ cat /dev/null > notrouter
3$ touch notrouter

4$ ls -l notrouter
-rw-------   1 mm64     users          0 Dec  1 12:51 notrouter
```

□