13.6 daemon_inetd Function

Figure 13.11 shows a function named daemon_inetd that we can call from a server we know is invoked by inetd.

Figure 13.11 daemon_inetd function: daemonizes process run by inetd.

daemon_inetd.c

```
1 #include     "unp.h"
2 #include     <syslog.h>

3 extern int daemon_proc;         /* defined in error.c */

4 void
5 daemon_inetd(const char *pname, int facility)
6 {
7     daemon_proc = 1;            /* for our err_XXX() functions */
8     openlog(pname, LOG_PID, facility);
9 }
```

This function is trivial compared to daemon_init, because all of the daemonization steps are performed by inetd when it starts. All that we do is set the daemon_proc flag for our error functions (Figure D.3) and call openlog with the same arguments as the call in Figure 13.4.

Example: Daytime Server as a Daemon Invoked by inetd

Figure 13.12 is a modification of our daytime server from Figure 13.5 that can be invoked by inetd.

Figure 13.12 Protocol-independent daytime server that can be invoked by inetd.

inetd/daytimetcpsrv3.c

```
 1 #include     "unp.h"
 2 #include     <time.h>

 3 int
 4 main(int argc, char **argv)
 5 {
 6     socklen_t len;
 7     struct sockaddr *cliaddr;
 8     char    buff[MAXLINE];
 9     time_t  ticks;

10     daemon_inetd(argv[0], 0);

11     cliaddr = Malloc(sizeof(struct sockaddr_storage));
12     len = sizeof(struct sockaddr_storage);
13     Getpeername(0, cliaddr, &len);
14     err_msg("connection from %s", Sock_ntop(cliaddr, len));

15     ticks = time(NULL);
16     snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
17     Write(0, buff, strlen(buff));

18     Close(0);                   /* close TCP connection */
19     exit(0);
20 }
```

There are two major changes in this program. First, all the socket creation code is gone: the calls to tcp_listen and to accept. Those steps are done by inetd and we reference the TCP connection using descriptor 0 (standard input). Second, the infinite for loop is gone because we are invoked once per client connection. After servicing this client, we terminate.

Call getpeername

11–14 Since we do not call tcp_listen, we do not know the size of the socket address structure it returns, and since we do not call accept, we do not know the client's protocol address. Therefore, we allocate a buffer for the socket address structure using sizeof(struct sockaddr_storage) and call getpeername with descriptor 0 as the first argument.

To run this example on our Solaris system, we first assign the service a name and port, adding the following line to `/etc/services`:

```
mydaytime      9999/tcp
```

We then add the following line to `/etc/inetd.conf`:

```
mydaytime  stream  tcp  nowait  andy
      /home/andy/daytimetcpsrv3  daytimetcpsrv3
```

(We have wrapped the long line.) We place the executable in the specified location and send the `SIGHUP` signal to `inetd`, telling it to reread its configuration file. The next step is to execute `netstat` to verify that a listening socket has been created on TCP port 9999.

```
solaris % netstat -na | grep 9999
      *.9999               *.*               0      0  49152     0  LISTEN
```

We then invoke the server from another host.

```
linux % telnet solaris 9999
Trying 192.168.1.20...
Connected to solaris.
Escape character is '^]'.
Tue Jun 10 11:04:02 2003
Connection closed by foreign host.
```

The `/var/adm/messages` file (where we have directed the `LOG_USER` facility messages to be logged in our `/etc/syslog.conf` file) contains the following entry:

```
Jun 10 11:04:02 solaris daytimetcpsrv3[28724]: connection from 192.168.1.10.58145
```