Session NM057

Library Routines

Joel Snyder
Opus1

Course Roadmap

- NM055 (11:00-12:00) Important Terms and Concepts
  - TCP/IP and Client/Server Model
  - Sockets and TLI
  - Client/Server in TCP/IP
- NM056 (1:00-2:00) Socket Routines
- NM057 (2:00-3:00) Library Routines
- NM058 (3:00-4:00) Sample Client/Server
- NM059 (4:00-5:00) VMS specifics (QIOs)
- NM067 (6:00-7:00) Clinic - Q&A

Library Routines Roadmap

- The byteorder routines
- The inet routines
- The /etc/* database routines

Byte order routines give machine independence

- Remember that network byte order is not necessarily the same as host byte order
- Always use byteorder routines when you want to look inside of protocol headers or call protocol routines
- What you do in your own code for byte order is up to you
  - If you are moving strange things (not ASCII) over the network, you should be using RPC/XDR

Byte Order converts 16 and 32 bit quantities

```
#include <sys/params.h>


u_long  htonl(u_long hostlong);      /* converts 32-bit host to 32-bit network*/
u_short  htons(u_short hostshort);   /* converts 16-bit host to 16-bit network */
u_long  ntohl(u_long netlong);       /* converts 32-bit network to 32-bit host */
u_short  ntohs(u_short netshort);    /* converts 16-bit network to 16-bit host */
```

Inet routines manipulate Internet addresses

- Internet addresses are expressed often as character strings (e.g., “192.245.12.2”)
- Internet addresses internally are all numbers (e.g., 0xABADF00D)
- Inet routines convert between the two (and do other things)

2 most important ones: inet_addr and inet_ntoa

Calling Sequence:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

u_long  inet_addr(char *cp);         /* convert string 192.245.12.2 to number */
char *  inet_ntoa(struct in_addr in);         /* convert number to string */
```

Example:

```
/* We have the IP address in the character array pointed to by host */
address = inet_addr(host);
if (address == INADDR_NONE) {
     printf("Your host name %s is not well formed.\n", host);
     return FALSE;
}
```

Converting names is not the same as converting strings

- inet_addr() and inet_ntoa() convert “192.245.12.2” and 0x0BADF00D back and forth.
  - This is purely a mechanical operation
- What about converting “Tennis.Opus1.COM” to a number?
  - Now we have to go to the network databases

In early TCP/IP world, network databases are files

What are we keeping track of?	Where are we storing it?
List of all hosts in the world	/etc/hosts
List of all networks in the Internet	/etc/networks
List of all protocols on this system	/etc/protocols
List of all services on this system	/etc/services

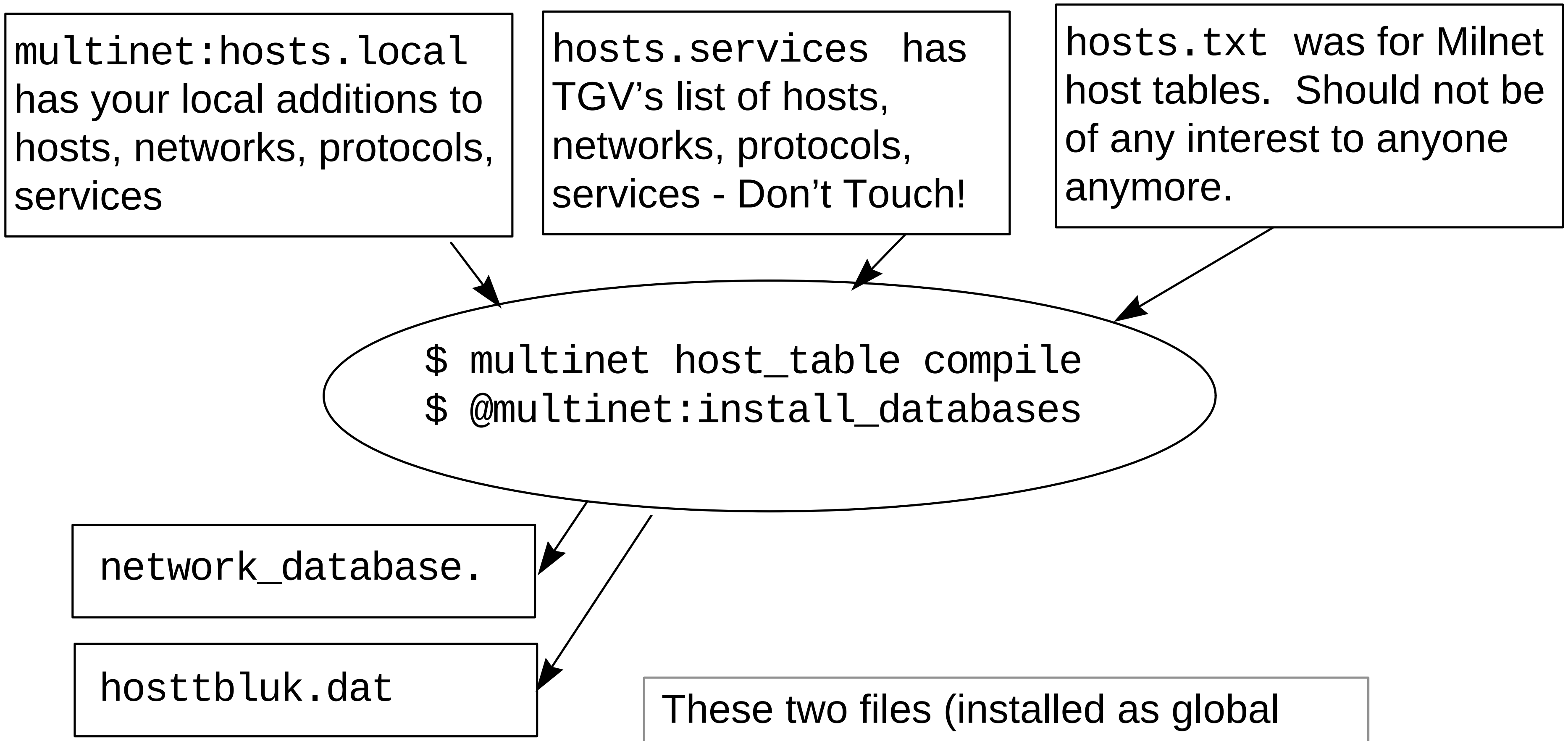
The Domain Name System helped a lot!

What are we keeping track of?	Where are we storing it?
List of all hosts in the world	on the network!
List of all networks in the Internet	<not used any more>
List of all protocols on this system	/etc/protocols
List of all services on this system	/etc/services

Ob: TCPware-specific info

- Local information - can edit these files:
  - TCPWARE:HOSTS. - local host definitions; initialized during configuration
  - TCPWARE:SERVICES. - initial defaults
  - TCPWARE:PROTOCOLS. - initial defaults
  - TCPWARE:NETWORKS. - initial defaults
- DNS client process provides this information to applications
- DNS client automatically updated when files are edited

Ob: Multinet-specific info



`multinet:hosts.local` has your local additions to hosts, networks, protocols, services

`hosts.services` has TGV's list of hosts, networks, protocols, services - Don't Touch!

`hosts.txt` was for Milnet host tables. Should not be of any interest to anyone anymore.

```
$ multinet host_table compile
$ @multinet:install_databases
```

`network_database.`

`hosttbluk.dat`

These two files (installed as global sections) plus the DNS (if used on your system) are used to answer queries from local applications.

Translating names to addresses

- Goal is to have IP address in a 32-bit quantity
- We can start with either a domain name, e.g. Hearts.ACES.COM, or an IP address, e.g. 198.102.68.2
- We can also take service, protocol, and network names and convert them to the relevant structures, e.g. TELNET, FTP, etc.

Retrieve information from databases with routines

What are we keeping track of?	Where are we storing it?	How do we see it?
List of all hosts in the world	on the network!	`gethostbyname()`, `gethostbyaddr()`
List of all networks in the Internet	<not used any more>	`getnetbyname()`, `getnetbyaddr()`
List of all protocols on this system	/etc/protocols	`getprotobyname()`, `getprotobynumber()`
List of all services on this system	/etc/services	`getservbyname()`, `getservbyport()`

There is, however, a little complication

- The `get`yyyby`xxx()` routines return information in structures

- `gethostby xxx()` returns hostent
- `getservby xxx`() returns servent
- `getprotoby xxx`() returns protoent
- `getnetby xxx`() returns netent

Host Routines: Overview

- Used to return host addresses from databases or name servers
- Can query by host name
  - DNS gurus: A records
- Can query by address
  - DNS gurus: PTR records
- Returns everything in hostent structures

Host Routines: Synopsis

```
#include <netdb.h>

struct hostent * gethostbyname(char *name);
    /* returns pointer to an object describing an Internet
    host referenced by name */

struct hostent * gethostbyaddr(char *addr, int len, int
type);
    /* returns pointer to an object describing an Internet
    host referenced by address */
    /* len = length of address */
    /* type = type of address, should be AF_INET */
```

hostent structure

- A hostent structure (defined in <netdb.h>) which is:

```
struct hostent {
   char  *h_name:           /* official name of host */
   char  **h_aliases;       /* alias list */
   int   h_addrtype;        /* host address type */
   int   h_length;          /* length of each address */
   char  **h_addr_list; /* list of addresses */
};
```

Service Routines: Overview

- Used to convert service names (SMTP, POP, FTP, TELNET) to numbers
- Also returns the preferred protocol for use with that service
- Returns everything in servent structures

Service Routines: Synopsis

```
#include <netdb.h>

struct servent * getservbyname(char *name, char *proto);
    /* returns pointer to an object describing a service
    on the local machine referenced by name and protocol
    to be used */

struct servent * getservbyport(int port, char * proto);
    /* returns pointer to an object describing a service
    on the local machine referenced by port and protocol
    number */
    /* “proto” may be left null */
```

servent structure

- A servent structure is:

```
struct servent {
   char  **s_name;       /* official service name */
   char  **s_aliases;    /* alias list */
   int    s_port;        /* port number, network byte order */
   char *s_proto;        /* protocol to use */
};
```

Protocol Routines: Overview

- Used to convert protocol names (IP, TCP, UDP) to numbers
- Returns everything in protoent structures
- Only ever used to be absolutely correct when calling socket() routines

Protocol Routines: Synopsis

```
#include <netdb.h>

struct protoent * getprotobyname (char *name);
    /* returns pointer to an object describing a protocol
    which may or may not be offered on the local machine,
    by name */

struct protoent * getprotobynumber (int proto);
    /* returns pointer to an object describing a protocol
    which may or may not be offered on the local machine,
    by number */
```

servent structure

- A protoent structure is:

```
struct protoent {
   char  *s_name;        /* official name of protocol */
   char  **p_aliases;    /* alias list */
   int   p_proto;        /* protocol number */
};
```

Network Routines: Overview

- Used to convert network names to numbers
- Returns everything in netent structures
- Never used any more

Library Routines
Key Concepts

- Use byteorder routines (htonl, ntohl, etc.) to keep protocol headers straight
- Use inet routines (inet_addr, inet_ntoa) to translate simple numeric strings to numbers
- Use netdb routines (gethostbyname, etc.) to look up information in the network databases

OpenVMS Specific Issues

Using Sockets on VMS

- Any application code can use sockets
- Any language can use sockets (e.g. Tony McCraken’s Bare Bones Telnet BBTN program in MACRO-32)
- Data structures and library routines pre-defined for the C language

C Run-Time Integration

- DEC has integrated support for sockets into VAX C and DEC C run-time libraries
- Must use DEC TCP/IP Services (UCX) or emulation
- Should use header files that come with VAX C/DEC C with this interface
- Can use read() and write()

TCPware’s Socket Library

- DISCOURAGED! Be Vendor Neutral!
- Pre-dates VAX C/DEC C API (and UCX)
- Socket library provided as shareable image and object library

MultiNet's Socket Library

- DISCOURAGED! Be Vendor Neutral!
- Pre-dates VAX C/DEC C API (and UCX)
- Socket library provided as shareable image
- Not integrated with C RTL I/O subsystem
  - Must use socket_read(), socket_write() and socket_close() on MultiNet sockets
  - Separate socket_errno value and socket_perror() routine

TCPware Header Files

- Don’t use. Modern versions of DEC C include all the routines you need
- Header files under TCPWARE_INCLUDE:

MultiNet Header Files

- Don’t use. Modern versions of DEC C include all the routines you need
- Header files under MULTINET_ROOT:[MULTINET.INCLUDE]
  - <sys/types.h> becomes MULTINET_ROOT:[MULTINET.INCLUDE.SYS]TYPES.H
  - <resolv.h> becomes MULTINET_ROOT:[MULTINET.INCLUDE]RESOLV.H
- There are bugs in the Multinet header files
  - We warned you!

Multinet - Easing the transition

- You can ease the transition by defining logical names
  - `$ define inc multinet_root:[multinet.include.]/translation=concealed`
  - `$ define sys inc:[sys],sys$share:`
  - `$ define net inc:[net],sys$share:`
  - `$ define netinet inc:[netinet],sys$share:`
- These are not needed for modern versions of DEC C; the compiler has it all figured out

OpenVMS Specific
Key Concepts

- Don’t use Multinet or TCPware specific routines unless you have to
- Portable sockets are usually the way to go
- Session NM059 (4:00) will discuss when and how to use QIOs for VMS