1. AIMS

The aim of this project is to introduce the student to the socket API and to build a small client to test some services. As a side effect, they will learn about files in the Linux operating system pertaining to networking.

2. PRELIMINARY MATERIAL

In order to do this exercise the student will have to understand how certain aspects of network client building. We present those below:

2.1 Looking up a domain name

Clients must specify the address of a server using sockaddr_in shown below:

```c
struct sockaddr_in{
    u_char sin_len;
    u_short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8]
}
```

The problem is that the IP address in sockaddr_in is in 32 bit binary form, not dotted decimal or mnemonic form. Utility routines are provided by the OS to perform conversions:

1. inet_addr converts dotted decimal to a binary IP address
2. gethostbyname takes a domain name and returns the hostent structure that contains the IP address in binary.

To use gethostbyname we need a little understanding of the hostent structure in the include file netdb.h:

```c
struct hostent{
    /* official host name */
    /* other aliases */
    /* address type */
    /* address length */
    char **h_addr_list;
    /* list of addresses */
}
```

#define h_addr h_addr_list[0];

netdb.h defines h_addr to refer to the first location in the host address list. Therefore the client code can refer to it as though it were a field in the structure.

```c
struct hostent *hptr;
char *examp="my.cs.umbc.edu";
if (hptr=gethostbyname(examp)){
    /*IP addr no in hptr->h_addr*/
}
else{
    /* deal with error */
}
```

2.2 Looking up a well known port by name

The servent structure is defined in netdb.h:

```c
struct servent{
    /* official service name */
    /* other aliases */
    int s_port; /* port for this service */
    /* protocol to use */
}
```

The following code snippet shows how to get the port number for sockaddr.

```c
struct servent *sptr;
if (sptr=getservbyname("smtp", "tcp")){
    /*port # is now in sptr->s_port */
}
else{
    /* deal with error */
}
```

2.3 Looking up a known protocol by name

Use getprotobyname to return the address of a structure of type protoent declared in netdb.h. As an exercise, go examine this data structure on gl.umbc.edu.

```c
struct protoent *pptr;
if (pptr=getprotobyname("udp")){
```
/*proto # is now in pptr->p_proto */
}
else{
    /* deal with error */
}

2.4 Connecting a TCP socket to a server

suc = connect(s, remaddr, remaddrlen);

We leave it to the student to determine the meaning of the arguments to connect. Note that connect returns 0 on success and –1 on failure. It does several things:

Checks to see if a specified socket is valid and that it is currently unconnected.
It fills in the remote endpoint address
Chooses (automatically) a local IP address and port # if the socket doesn’t have one.
Initiates TCP connection and returns a value indicating success or failure.

3. WHAT YOU WILL DO

The student will write at least two routines. The first one is described below:

Note: Your code must include the following <sys/types.h>, <sys/socket.h>, <netinet/in.h>, <arpa/inet.h>, <netdb.h>, <string.h>, <stdlib.h>.

Also use the following code snippet after your includes:

 ifndef INADDR_NONE
 #define INADDR_NONE 0xffffffff
 endif

Here is what your connectsock routine will look like:

int connectsock(const char *host, const char *service, const char *transport)
./* Fill in xxx, yyy, zzz with suitable and illustrative names */
struct hostent  xxx;
struct protoent yyy;
struct sockaddr_in zzz;
int   s, type;

./* Need to select address family AF_INET */
./* Map service name to port # */
./* Map hostname to IP address, allowing For dotted decimal and mnemonic */
./* Map transport protocol name to protocol # */
./* Use protocol to choose a socket type */
./* Allocate a socket */
./* Connect */
return s;
}

The 2nd piece of code you will write is a main program called testconnect where you will call your connectsock routine. In pseudo-code it will look like:

main{
/* Do input processing here */
for (each element in services list){
    connectsock();
    /* Table whether service is active or not and the transport protocol used */
    /* close socket */
} /* end for */
} /* end main */

The only thing remaining to be defined is the "services list" in the for loop of main. This list is defined below:
1. daytime 
2. ftp 
3. ssh 
4. telnet 
5. smtp

3.1 Your interface

A user will run your program in the following manner:

```%
% testconnect <host-name> <protocol> <services-list>
```

Where:

- **host-name** is the address of a machine on which you will test the availability of services. Use “merlin.cs.purdue.edu” and “localhost”. Run your client on “gl.umbc.edu”.
- **protocol** “udp” and “tcp”.
- **services-list** This should be a comma-separated list. The list is defined above (i.e., echo, discard etc.).

3.2 Your output

On issuing the command to run your program as defined above the screen should show:

<table>
<thead>
<tr>
<th>Service</th>
<th>Protocol</th>
<th>Active (y/n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo</td>
<td>TCP</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>UDP</td>
<td>N</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

3.3 What to submit, when to submit etc.

You will submit a complete program to the grader using the "submit" program. This program will consist of all the separate files used to write your program. Also, the first line in the main program will contain (in a comment) the command used to compile your program. The grader will use this to compile your program.

Additionally you will submit a separate sheet (in ASCII) showing tables of the sort shown in the preceding paragraph. There will be two tables, one for “merlin.cs.purdue.edu” and the other for “localhost”.

The project will be due no later than midnight of 24th October 2002. Late submissions will not be accepted.

3.4 Grading policy

1. You will work alone.
2. Grades will be assigned for:
   a. Correctness of operation of program: 80%.
   b. Comments at appropriate places in the code (don’t go over board, but no comments in your code will be penalized): 10%.
   c. Appropriate error handling capabilities: 10%.

3.5 Academic honesty

Academic misconduct may include but is not limited to the following:

1. Cheating: knowingly using or attempting to use unauthorized material, information, or study aids in any academic exercise.
2. Fabrication: Intentional and unauthorized falsification or invention of any information or citation in an academic exercise.
3. Facilitating Academic Dishonesty: Intentionally or knowingly helping or attempting to help another commit an act of academic dishonesty.
4. Plagiarism: Knowingly representing the words or ideas of another as one's own in any academic exercise, including works of art and computer-generated information/images.