

Lab 4:

Interprocess Connections



Image Generated by Dalle-E 2 from chat GPT

Prepared for

Rajani Phadtare

Prepared by Group 5

Valentine Jingwa, Jean-Pierre Nde-Forgwang

Operating System, Cohort D

School of Advance Digital Technology

SAIT

22 February 2024

Part A: Basic Mailbox Communication

Datagrams are simple messages that are sent to and read from a port (mailbox) on a local computer or on a computer on a network. Datagrams are basic building blocks used in making network data streams. The combination of an IP address and a port number is called a socket. A socket uniquely identifies a specific mailbox on a specific machine.

In this part, you'll investigate a Linux program that sends datagrams and a program that receives datagrams.

Before you start

- Ensure that you reboot your host computer at some point after you entered the lab. This ensures proper network connections.
- If you haven't already done so in a previous lab, add the Guest Additions.
- Download the file **Lab04-1.zip** from Brightspace and unzip it to your host machine.
- Open the folder and ensure you have all three files: **dgrecv.c**, **dgram.c** and **dgsend.c**.

You'll examine how to share a folder between your host machine and your Linux guest using a VirtualBox shared folder.

1. On your running Linux guest, go to the VirtualBox toolbar and open **Devices > Shared Folders**.
2. Click the **Add Folder** icon on the right, and then in the folder path input dropdown, select **Other**. In the browser window that opens, find and select the directory where you downloaded the program source code on your Windows host machine.
3. The Folder Name is automatically generated, but if there were any characters illegal for a Linux file name (e.g., spaces), you'll need to change the name. It is best to keep this name short and simple (e.g., WinFolder). Also, make the shared folder **Read Only** (to protect it), **not Auto-mounted** and **Permanent**, so it will re-attach the shared folder after the guest system is restarted.
4. Once you've chosen a legal folder name, the **OK** button is enabled and you can add the share.
5. Restart the Linux machine.
6. Because the shared folder was not auto-mounted (this is more bother than it's worth), you must manually mount it (i.e., connect to the Linux file system like any other newly added file device, like a USB memory stick).
 - a. Open a terminal window.

- b. Create an empty directory (e.g., “Shared”) as a “mount point” (a connection point for the shared files):

```
mkdir ~/Shared
```

- c. Mount the shared folder (assuming the suggested names above):

```
sudo mount -t vboxsf WinFolder ~/Shared
```

- d. The files must be copied from the shared folder to another location to be used, because it is read-only. You can save a copy to the home directory using `cp` in the terminal window, or open a file browser (**Places > Home Folder**) and copy/paste.

7. Open a terminal window, and then find and record the Linux hostname using the `hostname` shell command. Include all group members’ responses below. (1 mark)

Answer: Valentine-1-2 and| ubuntu

8. Find and record your IP address using the `ifconfig` command (not your loopback address). Include all group members’ responses. (1 mark)

Answer:

```
valentine@valentine-1-2:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe49:29a6 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:49:29:a6 txqueuelen 1000 (Ethernet)
    RX packets 553 bytes 339889 (339.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 686 bytes 118760 (118.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 385 bytes 112267 (112.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 385 bytes 112267 (112.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
ubuntu@ubuntu:~$ ifconfig
Command 'ifconfig' not found, but can be installed with:
sudo apt install net-tools
ubuntu@ubuntu:~$ sudo apt install net-tools
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  net-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 204 kB of archives.
After this operation, 815 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu mantic/main amd64 net-tools amd64 2.10-0.1ubuntu3 [204 kB]
Fetched 204 kB in 1s (296 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 217359 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu3_amd64.deb ...
Unpacking net-tools (2.10-0.1ubuntu3) ...
Setting up net-tools (2.10-0.1ubuntu3) ...
Processing triggers for man-db (2.11.2-3) ...
ubuntu@ubuntu:~$
```

9. Research the `gcc` (Gnu C compiler) command and briefly explain what it will do, given the parameters `gcc dgrecev.c dgram.c -o dgrecev`. (2 marks)

Answer: The `gcc` command is part of the GNU Compiler Collection, which includes the GNU C Compiler. The specific command `gcc dgrecev.c dgram.c -o dgrecev` is used to compile the source files `dgrecev.c` and `dgram.c` and link them together to create an executable file named `dgrecev`. This process involves compiling the individual C source code files into object files and then linking these object files to produce the final executable that can be run on the system.

10. Compile and run the program as follows:

```
gcc dgrecev.c dgram.c -o dgrecev
```

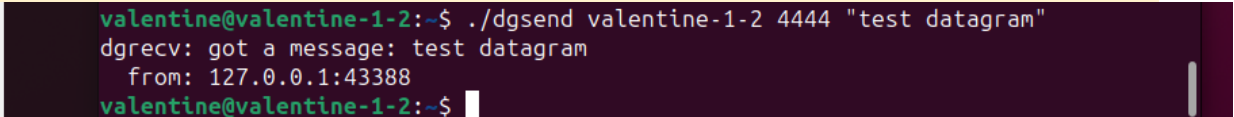
```
./dgreceive 4444 & (this server program runs in the background)
gcc dgsend.c dgram.c -o dgsend
./dgsend yourhostname 4444 "test datagram"
```

11. Why was `./` added before the executable file name? (1 mark)

Answer: The `./` before an executable file name in Unix-like systems, such as Linux, is used to indicate to the shell that it should execute a program located in the current working directory.

12. Record the results from the terminal. (1 mark)

Answer:



```
valentine@valentine-1-2:~$ ./dgsend valentine-1-2 4444 "test datagram"
dgreceive: got a message: test datagram
from: 127.0.0.1:43388
valentine@valentine-1-2:~$
```

13. Why is the port number recorded in the previous step not 4444? (1 mark)

Answer: This is part of the TCP/IP protocol behavior where the source port usually changes to a random high number port for the duration of the session to uniquely identify the connection.

14. Identify and explain the purpose and parameters of TWO network related system calls used in `dgram.c`. (2 marks)

Hint: If you see source code that defines the function (like a method), that isn't a system call. Look at the other calls and examine whether the name suggests a network operation.

socket() System Call: This system call creates a new socket, which is an endpoint for communication. In the `make_dgram_server_socket` and `make_dgram_client_socket` functions, this call is used to create a datagram socket for UDP communication.

- `PF_INET`: Specifies the protocol family. `PF_INET` is used for IPv4 Internet protocols.
- `SOCK_DGRAM`: Indicates the socket type, which is datagram in this case, meaning it's for UDP communication.
- `0`: Specifies the protocol. `0` is used to choose the default protocol for the given combination of family and type, which is UDP for `SOCK_DGRAM`.

Usage in `make_dgram_server_socket` function:

```
sock_id = socket(PF_INET, SOCK_DGRAM, 0);
```

bind() System Call: This system call binds a socket to an address, in this case, the address of the current host and port number. It is used in the `make_dgram_server_socket` function after creating a socket to specify the port on which the server will listen for incoming datagrams.

- `sock_id`: The file descriptor of the socket to be bound.
- `(struct sockaddr *)&saddr`: A pointer to a struct `sockaddr` that contains the address to bind to. This structure is cast from `sockaddr_in` which is specific to Internet operations.
- `sizeof(saddr)`: The length of the address structure.

Usage in `make_dgram_server_socket` function:

```
if (bind(sock_id, (struct sockaddr *)&saddr, sizeof(saddr)) != 0)
```

```
____return -1;
```

Part B: Simple Web Servers

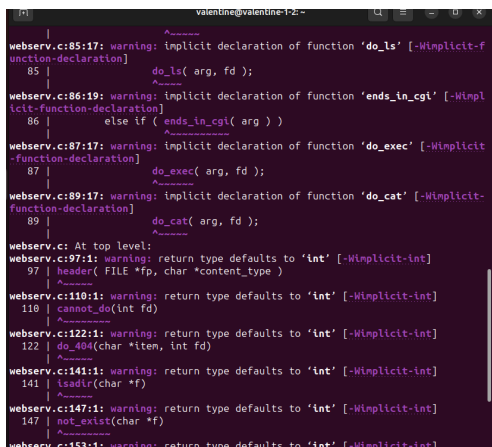
In this part of the lab, you'll investigate the operation of a server. A server is simply a program that waits for a request to arrive from some other process (possibly across a network), reads the request and then acts upon it.

Different servers handle different types of requests. For example, an HTTP request is usually a request for a file name (e.g., myfile.html). When the request arrives, an HTTP server reads the requested file from its storage, and then sends the file contents to the machine that sent the request.

The operating system continually scans the network for messages. When a message arrives on the network, the operating system determines from part of the message, to which port (mailbox) number the message was sent. That port number refers to a unique mailbox for each server running on that particular machine (e.g., an HTTP web server usually expects messages on port 80).

1. Download **La04-2.zip** from Brightspace to your host machine.
2. Unzip and ensure you have two files: **webserv.c** and **socklib.c**.
3. Compile the **webserv.c** program with the **socklib.c** library.

```
gcc webserv.c socklib.c -o webserv
```



```
webserv.c:85:17: warning: implicit declaration of function 'do_ls' [-Wimplicit-f
function-declaration]
85 |         do_ls( arg, fd );
    |         ^~~~~
webserv.c:86:19: warning: implicit declaration of function 'ends_in_cgi' [-Wimpl
ict-function-declaration]
86 |         else if ( ends_in_cgi( arg ) )
    |                   ^~~~~~
webserv.c:87:17: warning: implicit declaration of function 'do_exec' [-Wimplicit
-function-declaration]
87 |         do_exec( arg, fd );
    |         ^~~~~~
webserv.c:89:17: warning: implicit declaration of function 'do_cat' [-Wimplicit-
function-declaration]
89 |         do_cat( arg, fd );
    |         ^~~~~~
webserv.c: At top level:
webserv.c:97:1: warning: return type defaults to 'int' [-Wimplicit-int]
97 | header( FILE *fp, char *content_type )
    | ^~~~~~
webserv.c:110:1: warning: return type defaults to 'int' [-Wimplicit-int]
110 | cannot_do(int fd)
    | ^~~~~~
webserv.c:122:1: warning: return type defaults to 'int' [-Wimplicit-int]
122 | do_404(char *item, int fd)
    | ^~~~~~
webserv.c:141:1: warning: return type defaults to 'int' [-Wimplicit-int]
141 | loadtr(char *f)
    | ^~~~~~
webserv.c:147:1: warning: return type defaults to 'int' [-Wimplicit-int]
147 | not_exist(char *f)
    | ^~~~~~
webserv.c:153:1: warning: return type defaults to 'int' [-Wimplicit-int]
```

Before you can use the webserv program, you must install an HTTP daemon. This daemon handles the HTTP protocol requests that arrive at port 80 (default). If a port number is included in the request, the request will be redirected to another port. Thus, you can connect your webserv program to another port, and HTTP requests can be forwarded to that port.

4. Install the apache2 HTTP server application using the command:

```
sudo apt-get install apache2
```
5. This operation calculates the space needed to install the program. When asked if you wish to continue, choose **y**.
6. Use **ps** to ensure that the HTTP daemon (apache2) is running.

7. To test the installation of the HTTP daemon, start a web browser and navigate to **http://localhost**. By default, localhost is interpreted as the IP address of your own computer.
8. Open a terminal window and run the web server program, specifying a port number (e.g., 1234).

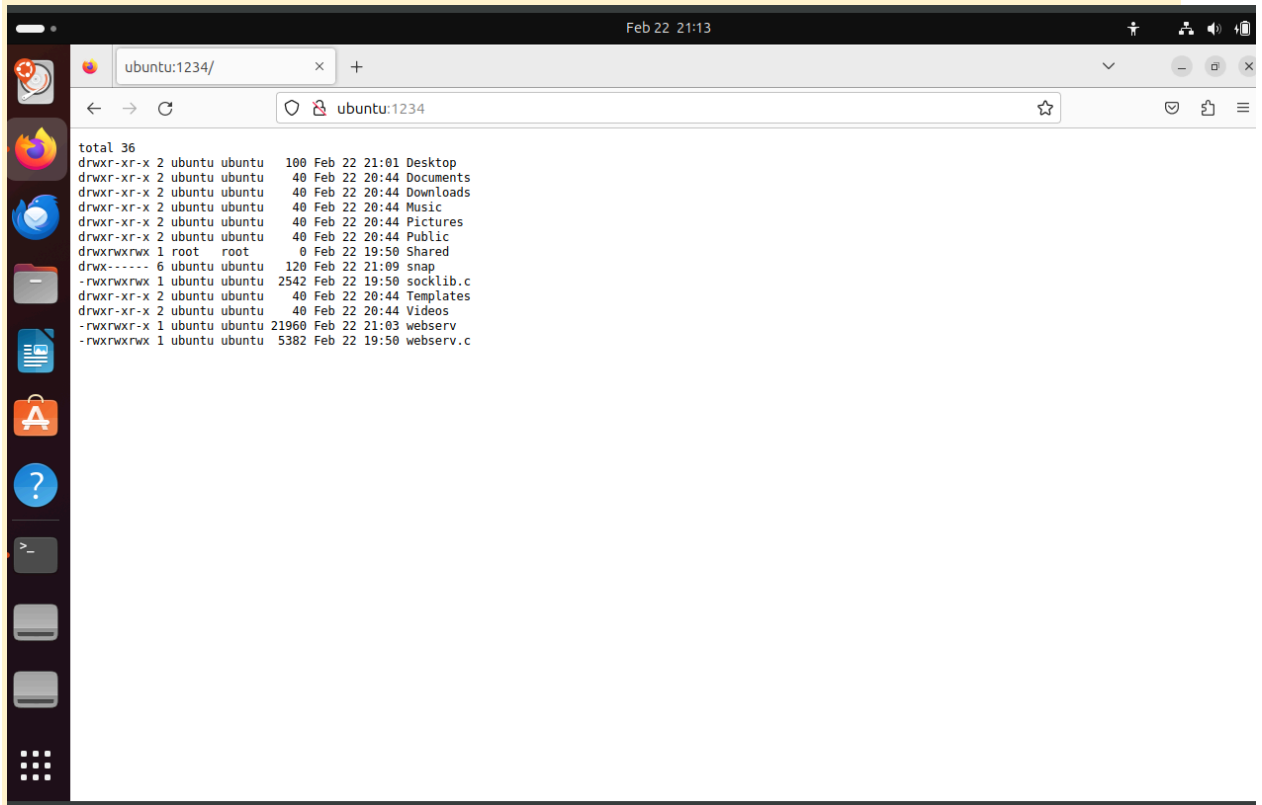
```
./webserv 1234
```

9. Start a web browser and navigate to **http://your_IP_address:1234**. Why doesn't it connect? (**Hint:** Check the file /etc/hosts) (1 mark)

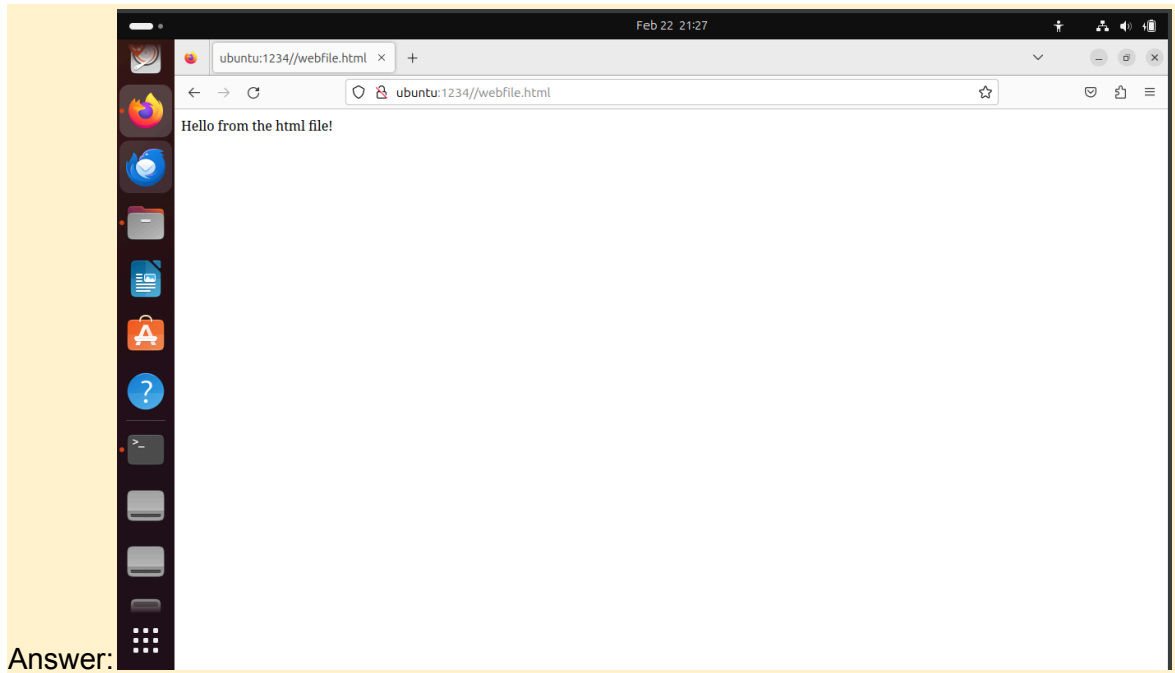
Answer: This file on the system maps hostnames to IP addresses. When you use an IP address directly, the system doesn't need to perform any name resolution which is the process of converting a hostname into an IP address. If the IP address I'm using doesn't match what the server is bound to, there will be no connection.

10. Navigate to: **http://yourhostname:1234**. How does this compare to the result of the previous step? Why does this connect? (1 mark)

Answer: My system can resolve my hostname to an IP address that the web server is listening on.



11. Add an html file (e.g. **webfile.html** with the line “Hello from the html file!” in it) to your web server root directory (e.g., /~), and then view the file using your web browser (e.g., **http://yourhostname:1234/webfile.html**)
12. Record the first line of output on the terminal that is running the web server. (1 mark)



13. Study the result from the previous step. What does this output indicate regarding the type and structure of the communication? (2 marks)

Answer: its an HTTP connection - The browser is the client requesting the HTML file, and the server is serving the requested file.
Structured Communication: Client-Server Model/Stateless/Connection Over TCP/IP

14. Identify and explain the purpose and parameters of TWO network related system calls used in **socklib.c**, excluding those you described in section 1. (2 marks)

Answer: The `socket()` and `bind()` calls are essential for setting up a server socket that listens for incoming connections. The `socket()` call is used both by the client to create a socket that can initiate a connection and by the server to create a socket that can accept connections. The `bind()` call is specific to the server, as it ties the newly created socket to a specific IP address and port so that clients know where to connect to.