COMPUTER SYSTEMS AND ORGANIZATION

Sockets

Daniel G. Graham Ph.D

1. ”Everything is a file” Kinda
2. Network communication as file operations.
3. Server socket
4. Building a Simple Webserver
5. Serving Files Webserver
6. Demo of Webserver in Browser
7. Next time Client Server Model and more

LECTURE PROJECT

Throughout the lecture, we'll build up to building a web server. The goal is to work up to a demo where we'll pull up our browsers type in a URL and get served a webpage.

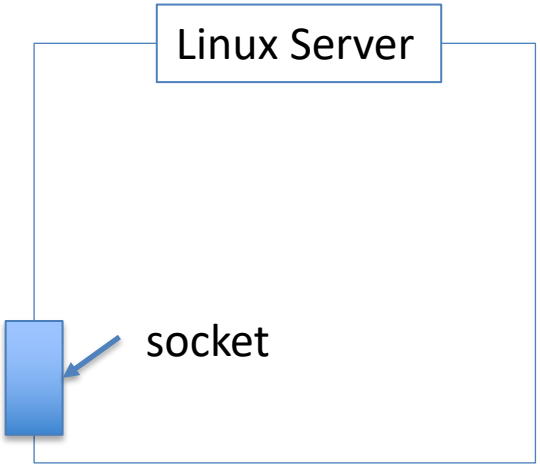
EVERYTHING IS A FILE (LINUX FILE INTERFACE)

```
 GNU nano 6.3                    everythingisafile.c
#include <stdio.h>
#include <unistd.h>

int main() {
    char buffer[256];
    ssize_t bytesRead;

    // Reading from standard input (file descriptor 0)
    bytesRead = read(0, buffer, sizeof(buffer) - 1);

    if (bytesRead < 0) {
        perror("Error reading from stdin");
        return -1;
    }

    // Null-terminate the string
    buffer[bytesRead] = '\0';

    // Writing to standard output (file descriptor 1)
    if (write(1, buffer, bytesRead) != bytesRead) {
        perror("Error writing to stdout");
        return -1;
    }

    return 0;
}
```

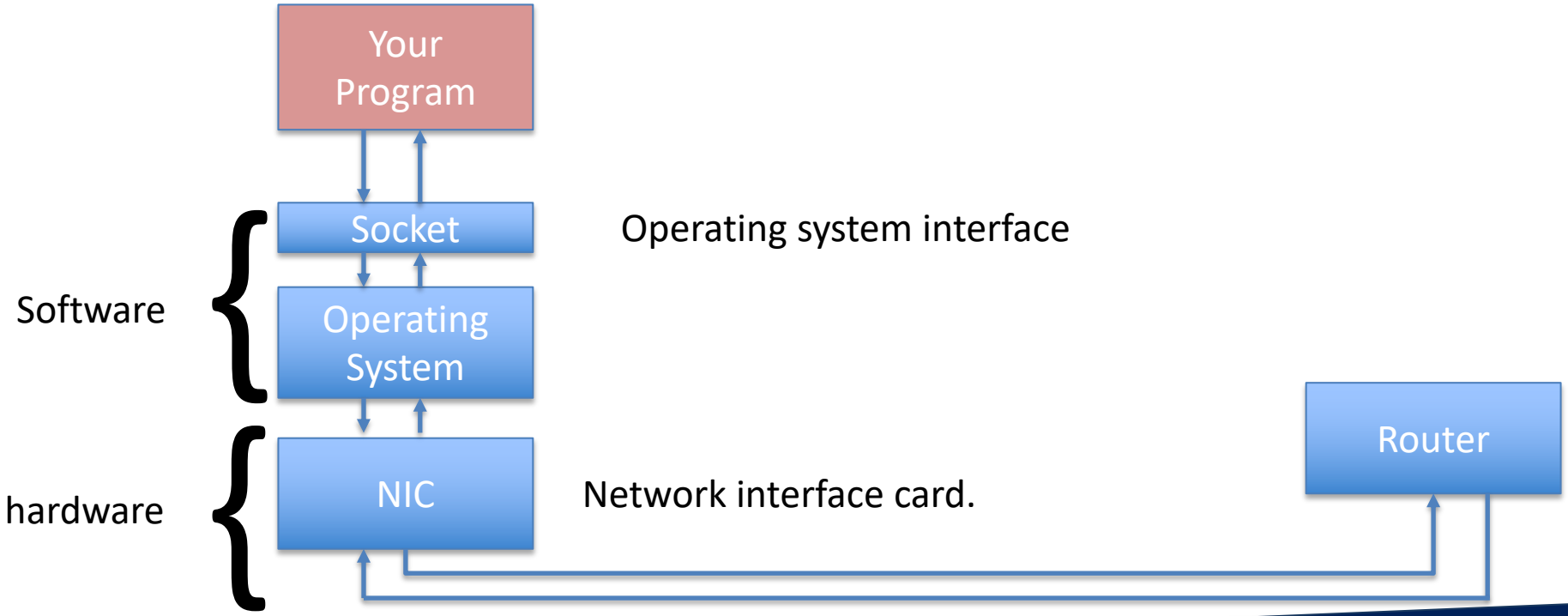
```
Home directory usage for /u/dgg6b: 1%
You have used 1.51G of your 100G quota

dgg6b@portal06:~/Lecture-Code/lecture-36$ clang everythingisafile.c
dgg6b@portal06:~/Lecture-Code/lecture-36$ ./a.out
Hello this is a test
Hello this is a test
dgg6b@portal06:~/Lecture-Code/lecture-36$
```

NETWORK COMMUNICATION

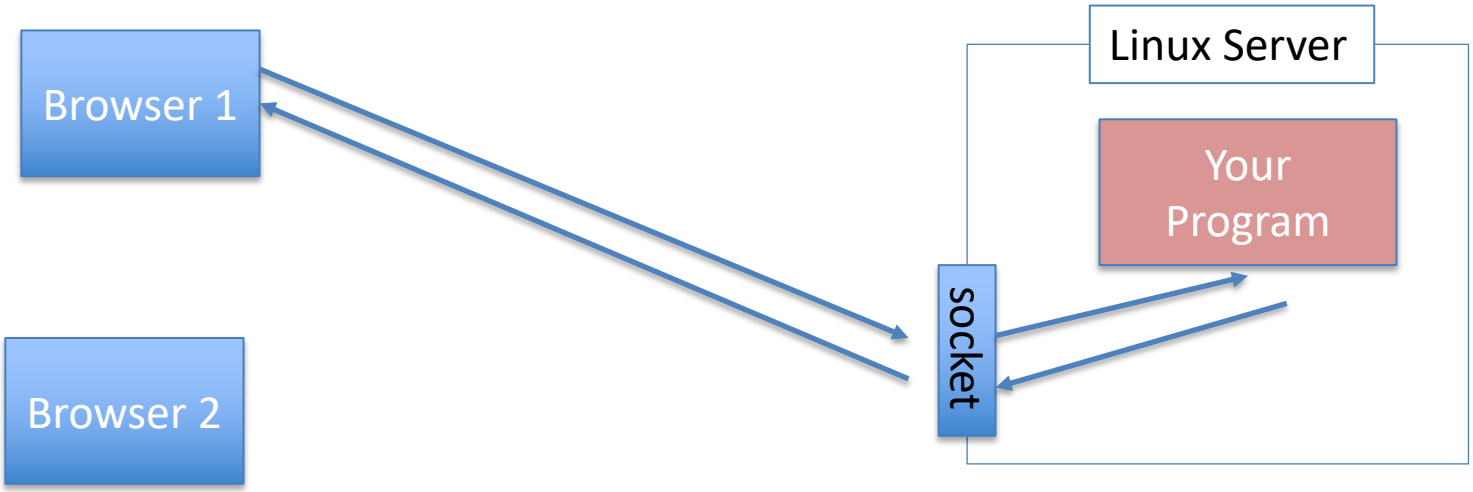
The socket provides a filesystem interface to the machine's network components.



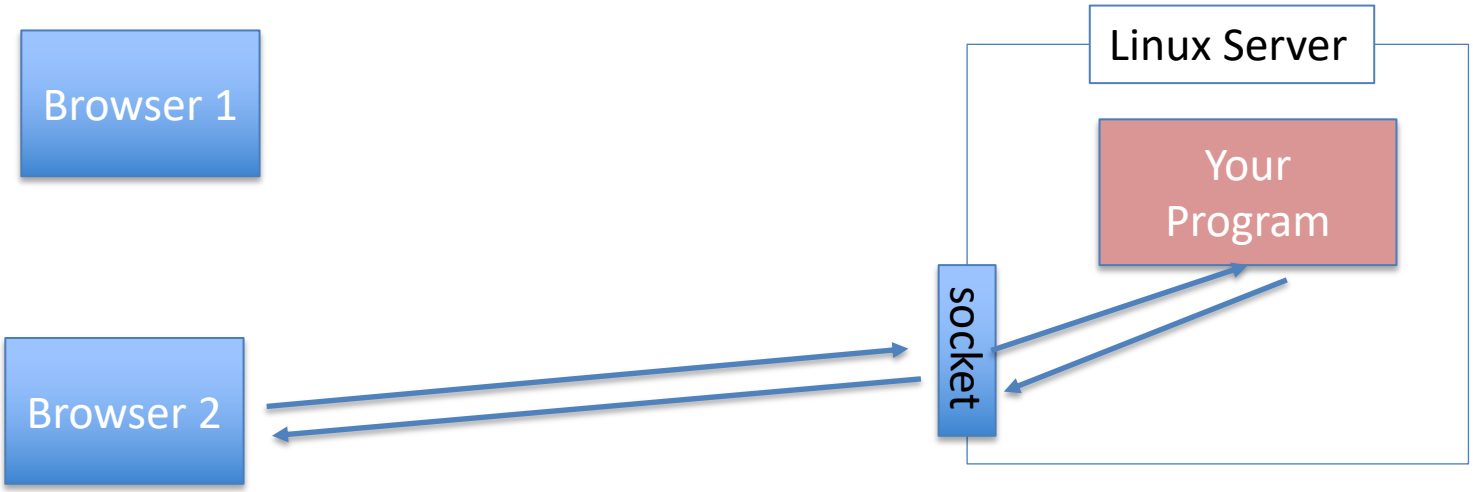
SOCKET



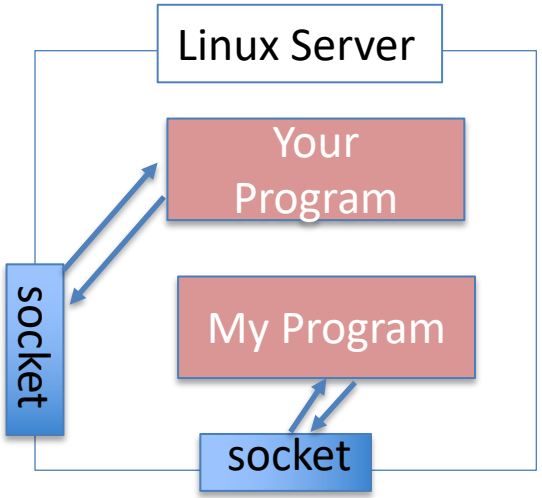
NETWORK COMMUNICATION



NETWORK COMMUNICATION

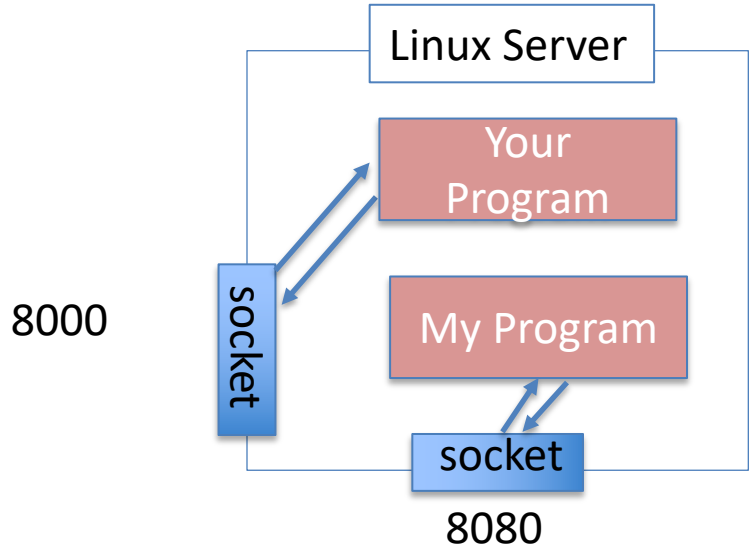


A COMPUTER CAN HAVE MULTIPLE PROGRAMS AND MULTIPLE SOCKETS



The solution: we assign a number between 1-65,535
The unique number is called a port number.

PORT NUMBERS

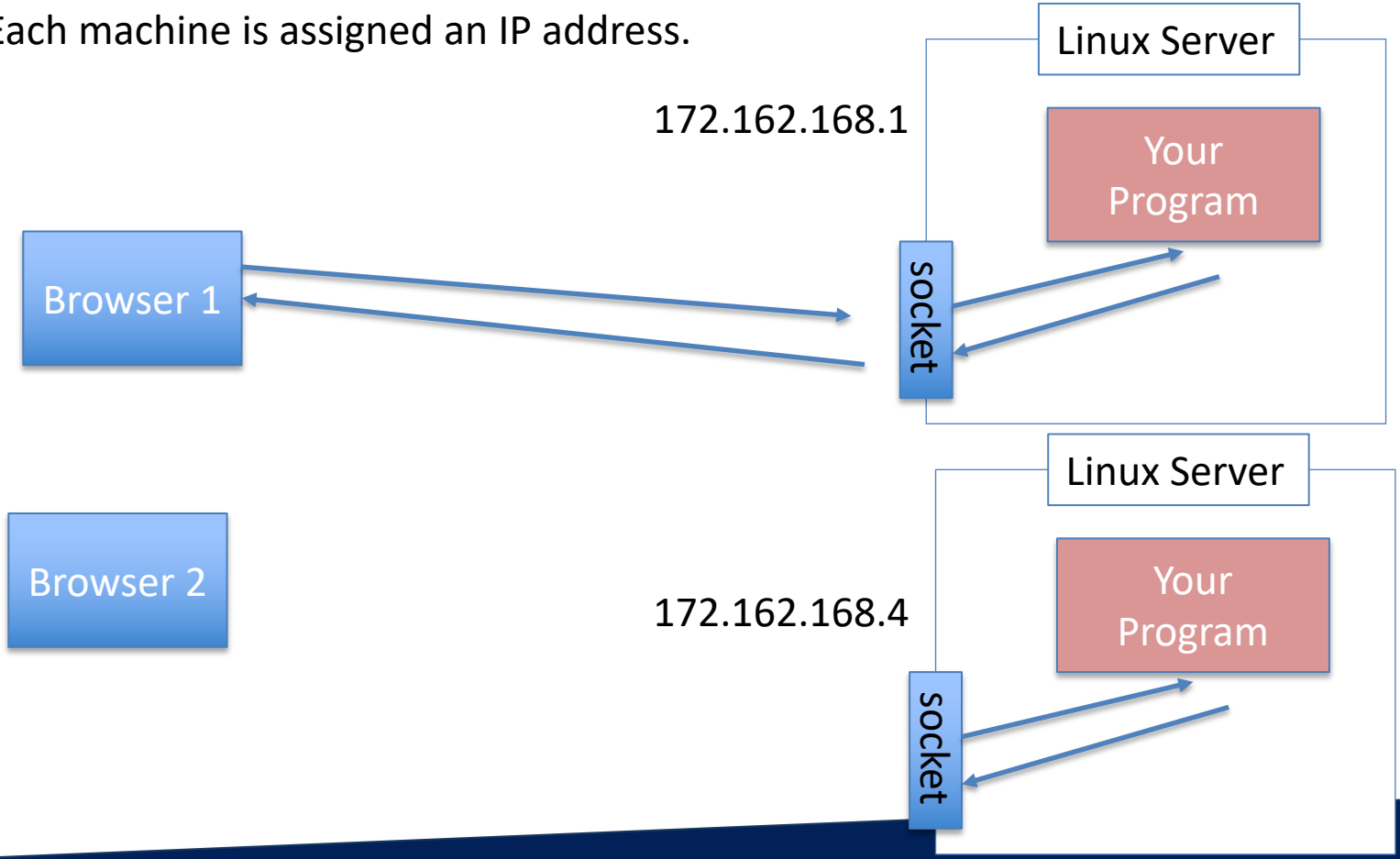


The solution: we assign a number between 1-65,535
The unique number is called a port number.

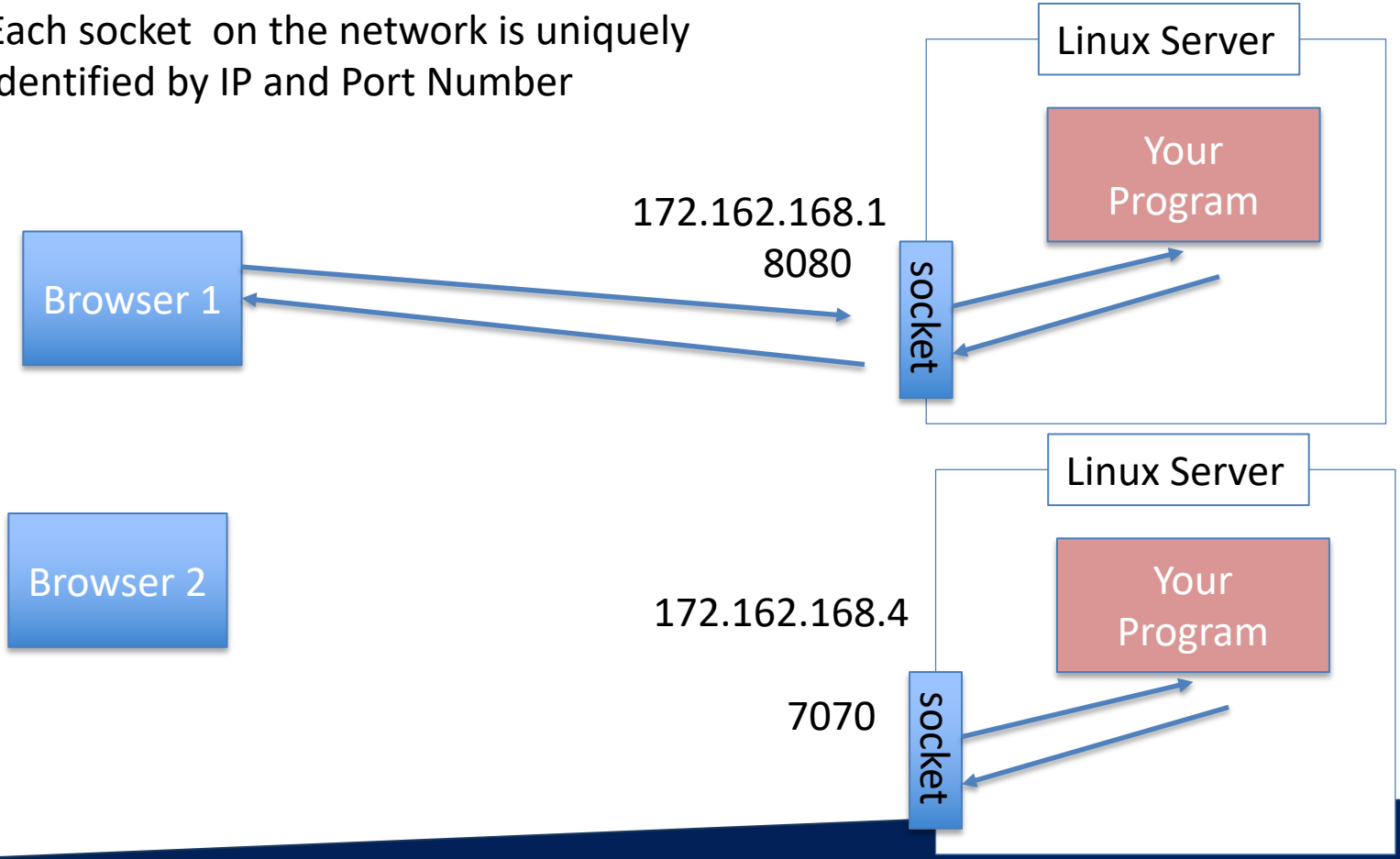
OS maintains the mapping between sockets (ports #) and processes.

THERE ARE SEVERAL MACHINES ON THE NETWORK HOW DO WE KNOW WHICH ONE TO CONNECT TO?

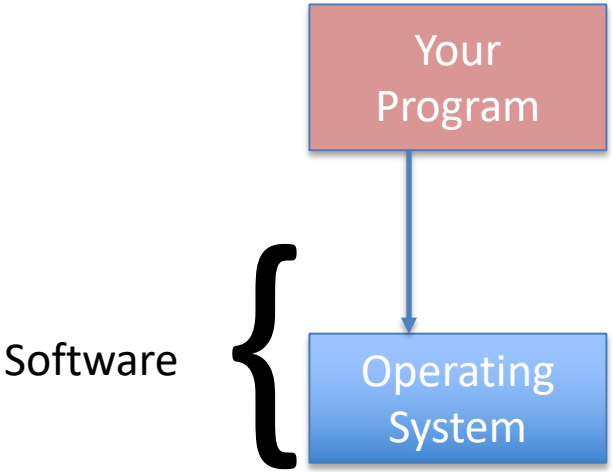
Each machine is assigned an IP address.



Each socket on the network is uniquely identified by IP and Port Number



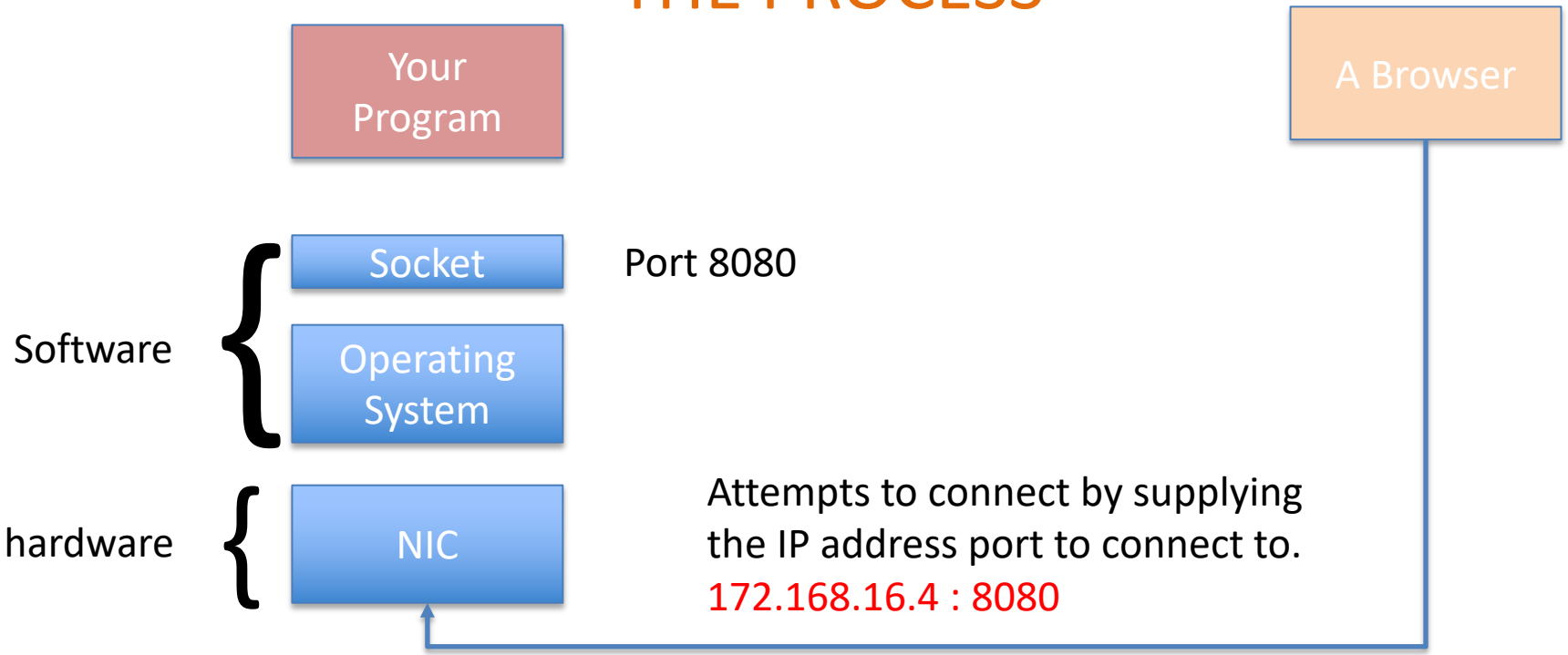
THE PROCESS



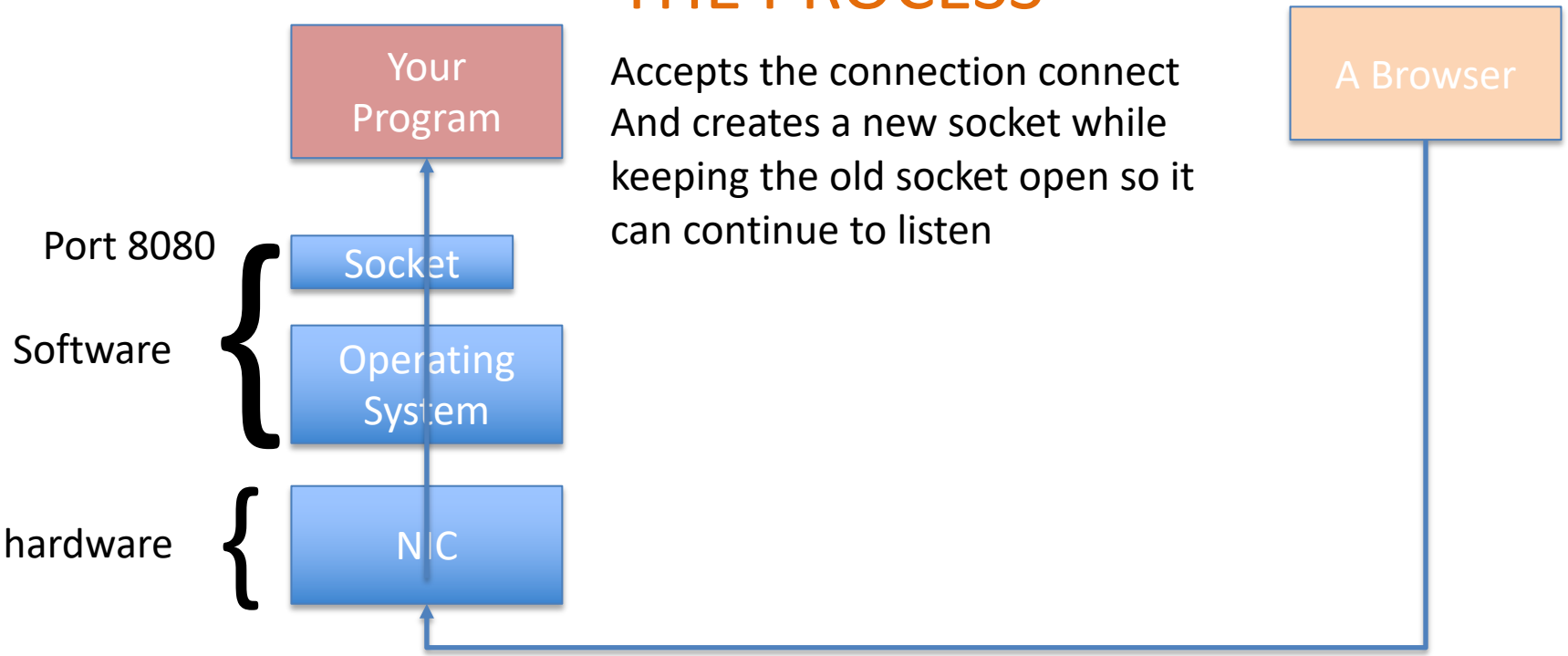
Tell the operating system to create a server socket interface.

Since this dynamically allocated socket pick a port number in the range 49152 to 65535.

THE PROCESS

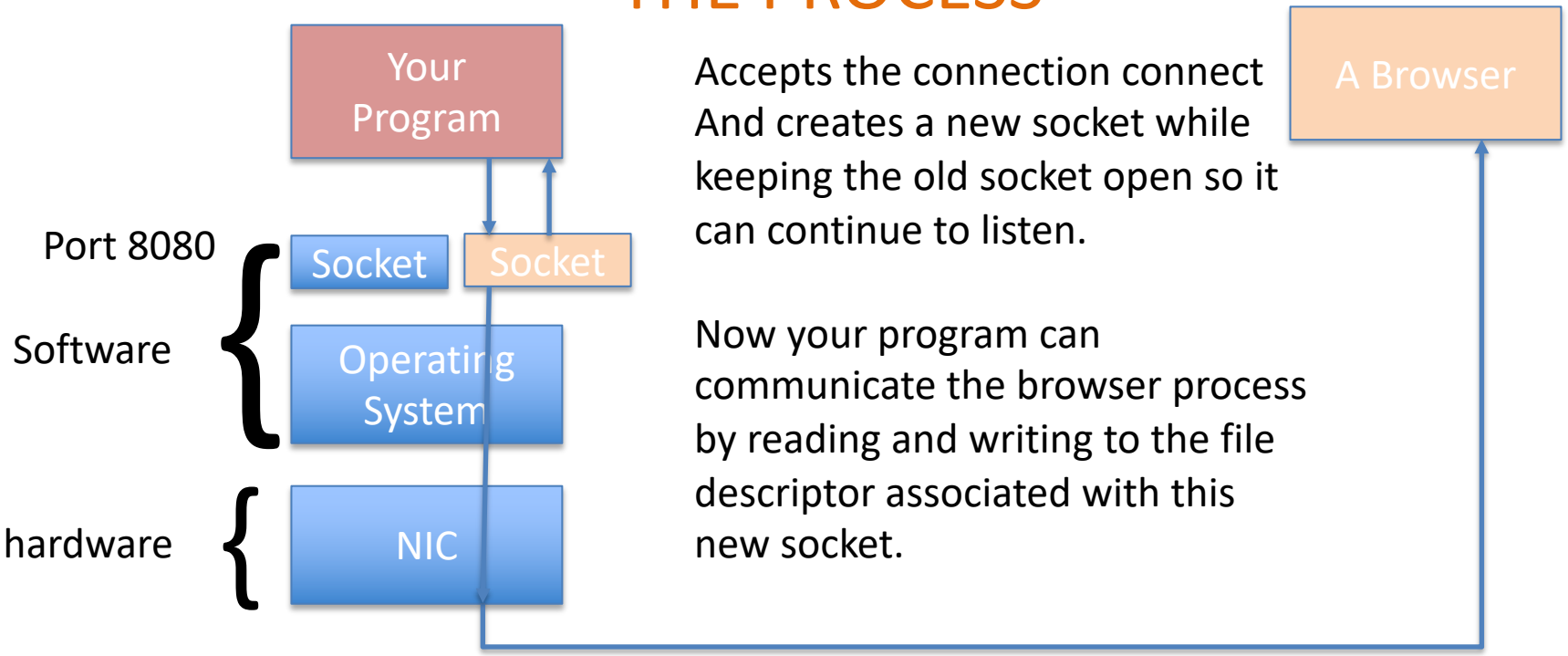


THE PROCESS



Accepts the connection connect And creates a new socket while keeping the old socket open so it can continue to listen

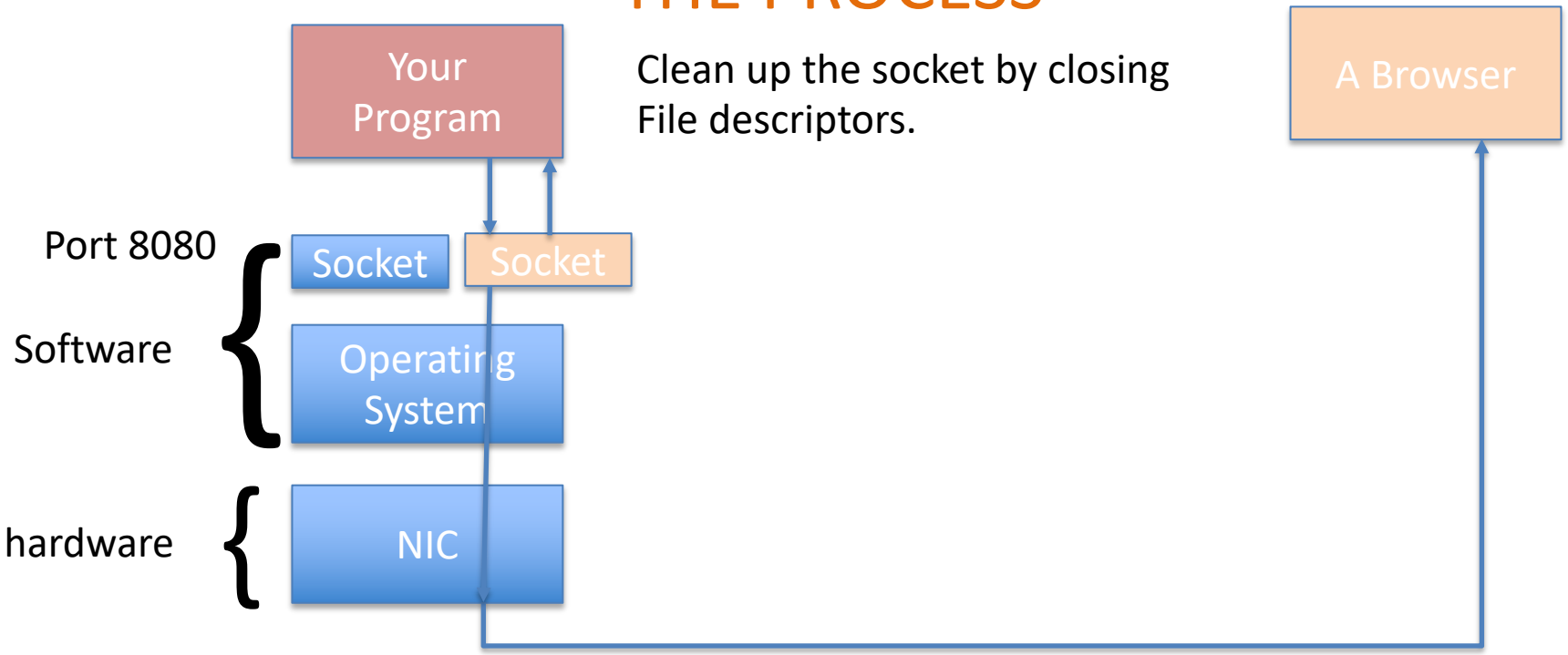
THE PROCESS



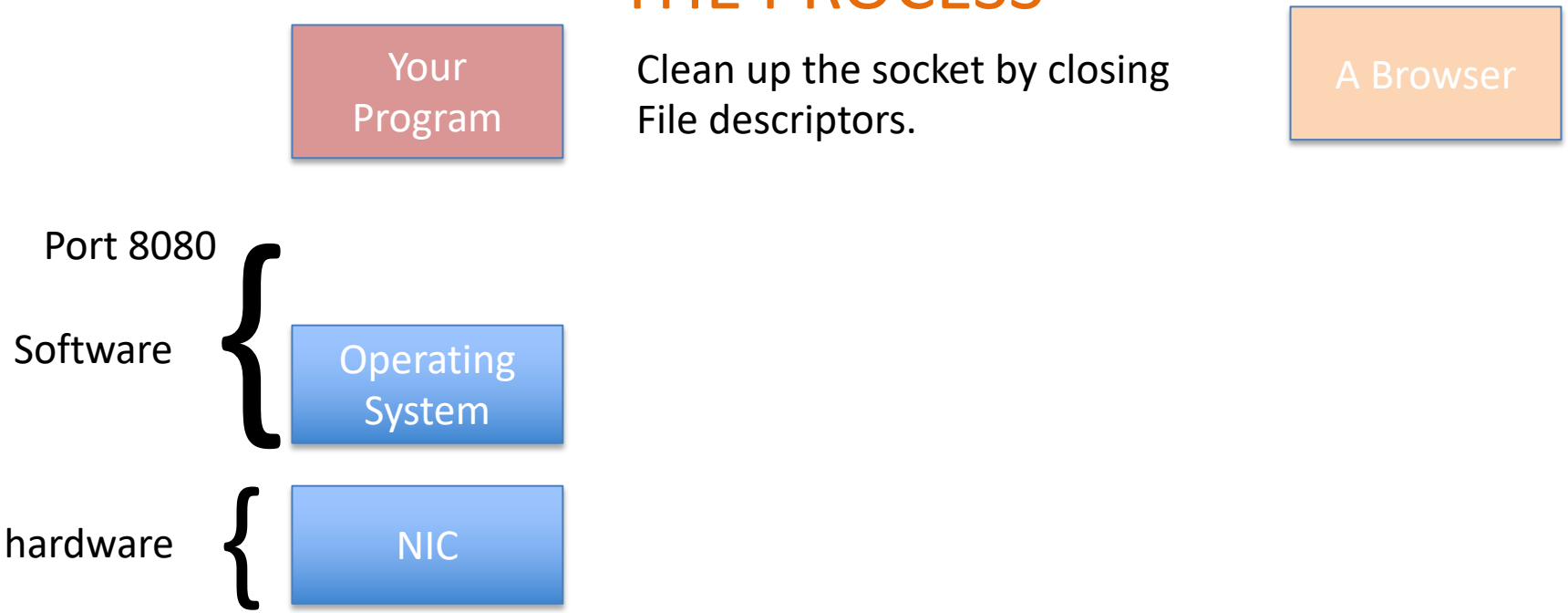
Accepts the connection connect And creates a new socket while keeping the old socket open so it can continue to listen.

Now your program can communicate the browser process by reading and writing to the file descriptor associated with this new socket.

THE PROCESS



THE PROCESS


Your Program
Clean up the socket by closing File descriptors.
A Browser
Port 8080
Software
Operating System
hardware
NIC

LET’S LOOK AT THE CODE FOR EACH STEP

Let’s start with the imports

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORT 8080
```

The unistd.h is a header file in C that provides access to the POSIX operating system API. It stands for "Unix standard" and includes a variety of functions and macros that are used to make system calls to the operating system. These calls include process control, file manipulation, and I/O operations, among others. This header is essential for programming in UNIX and UNIX-like environments, allowing for direct interaction with the kernel and system resources.

Let’s start with the imports

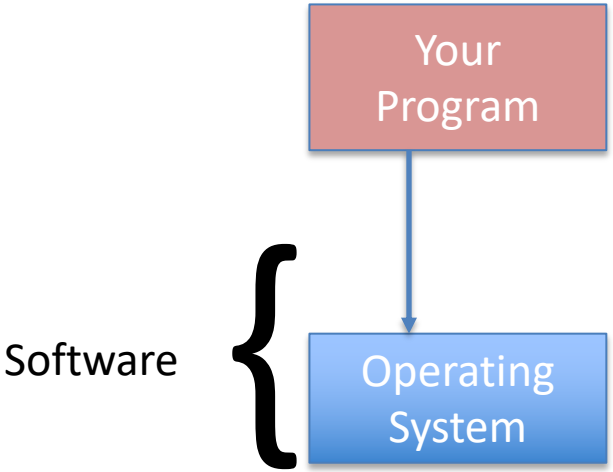
LET’S LOOK AT THE CODE FOR EACH STEP

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORT 8080
```

The <netinet/in.h> header file in C is used for Internet Protocol family socket definitions, providing the necessary structures and constants for network addresses. The / is simply a directory separator, indicating that in.h is inside the netinet directory.

We also defined a constant that will represent the port.

THE PROCESS



Initialize the variable that will hold the file descriptors.

```
int main() {
    int server_fd, client_fd;
    struct sockaddr_in address;
```

NEXT CREATE THE SOCKET

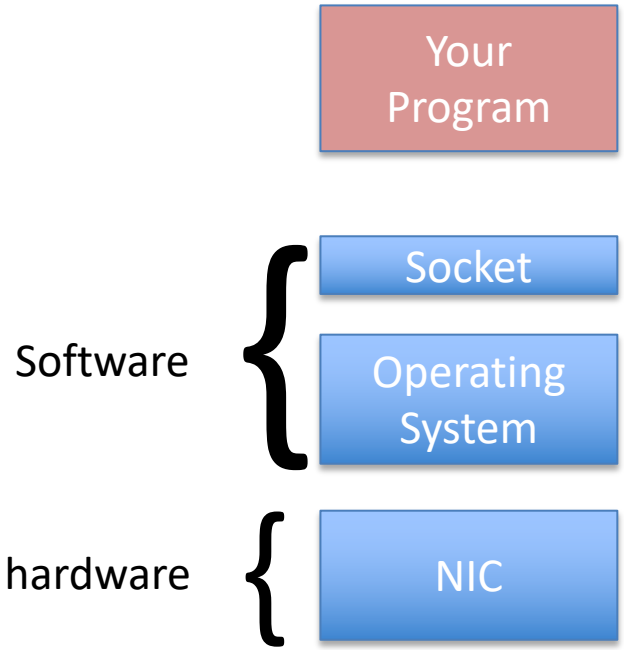
```
int socket(int domain, int type, int protocol);
```

domain: **AF_INET** IP v4 address 192.168.1.1, **AF_INET6** IP v6 address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 vs **AF_BLUETOOTH** bluetooth

type: TCP **SOCK_STREAM** reliable vs UDP **SOCK_DGRAM** unreliable

protocol: If the socket type supports multiple protocols, this parameter selects among them. However, most socket types only support a single protocol so this parameter is normally set to 0

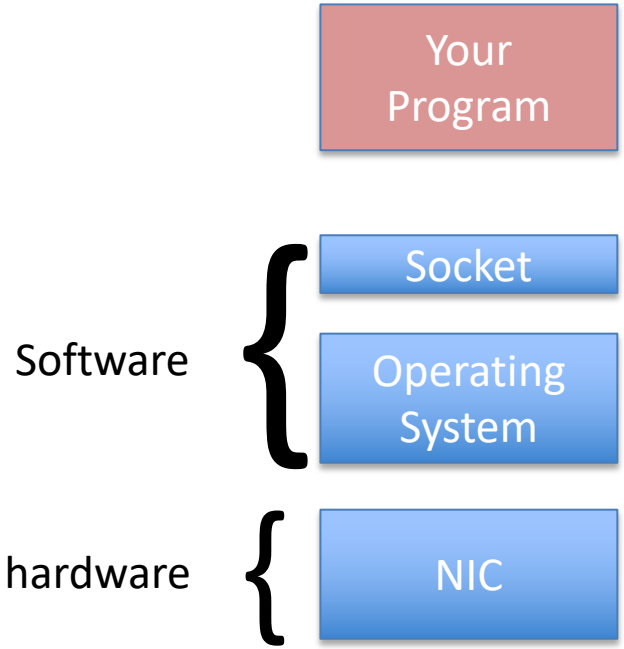
THE PROCESS




```
int main() {
    int server_fd, client_fd;
    struct sockaddr_in address;

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
```

Notice that the Port and IP have not yet been associated with the socket.

THE PROCESS




```
server_fd = socket(AF_INET, SOCK_STREAM, 0);
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);
```

Let's set up the struct that holds the address

`AF_INET:` Specifies address type IPv4

`INADDR_ANY:` Specifies that the socket assumes the IP address of the machine the program is running on.

`htons(PORT);` Host to Network short

HTONS

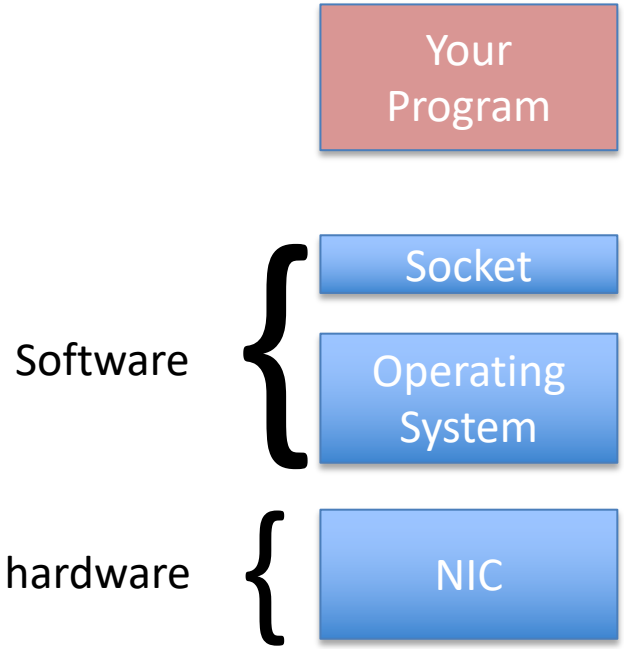
```c
#include <stdio.h>
#include <arpa/inet.h>

int main() {
    uint16_t hostshort = 0x1234; // Suppose 0x1234 is our host byte order short integer
    uint16_t netshort = htons(hostshort); // Convert to network byte order

    printf("Host ordered short: %hx\n", hostshort);
    printf("Network ordered short: %hx\n", netshort);

    return 0;
}
```

Prints

```
Host ordered short: 1234
Network ordered short: 3412
```

Network byte order is defined to be big-endian, which is the most significant byte first. Since different computer architectures use different byte orders, this function is used to ensure that data is represented consistently across different hosts.

THE PROCESS




```
server_fd = socket(AF_INET, SOCK_STREAM, 0);
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

bind(server_fd, (struct sockaddr *)&address
                                , sizeof(address));
```

The bind fuction final associates the IP address and port with the socket.

We need to pass in sizeof the address struct because we are passing in a pointer to the struct.

PLACE THE SOCKET IN LISTEN MODE

int listen(int sockfd**, int** backlog**);**

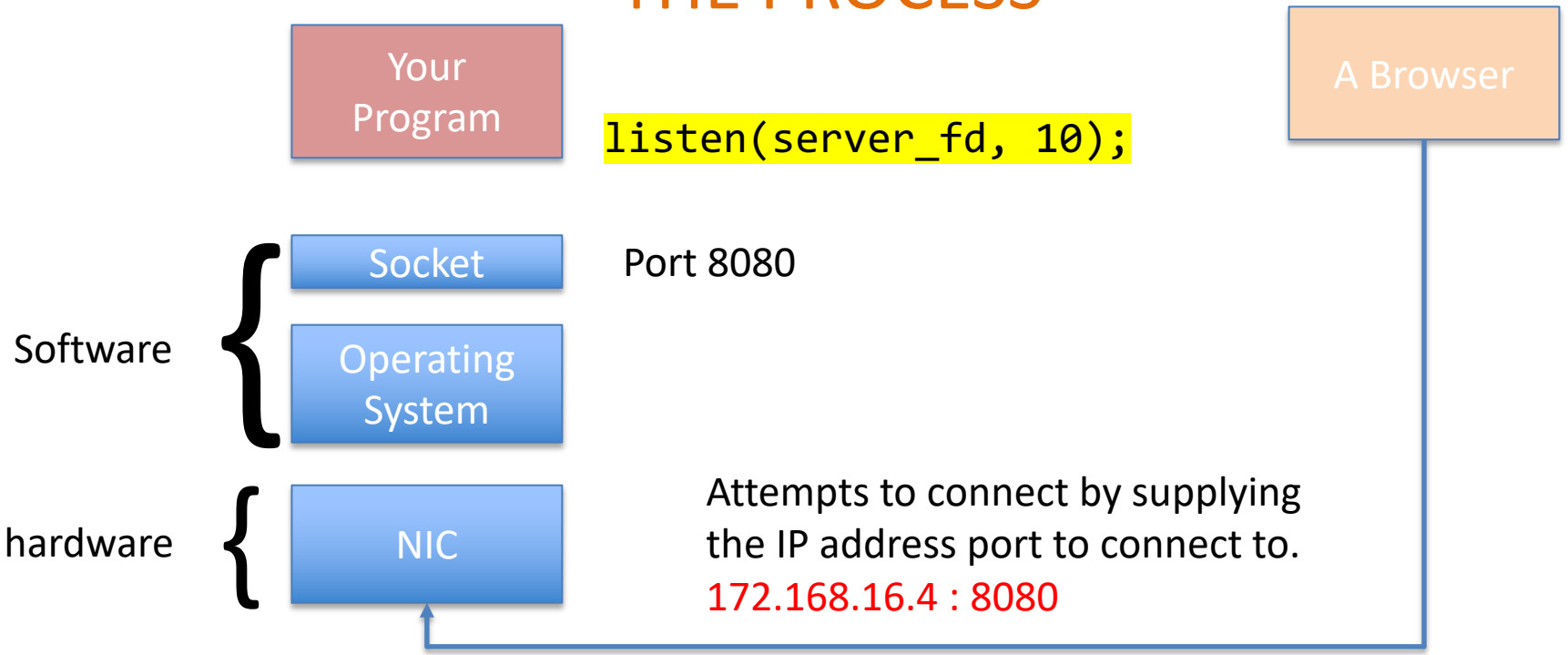
DESCRIPTION

listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using **accept**(2).

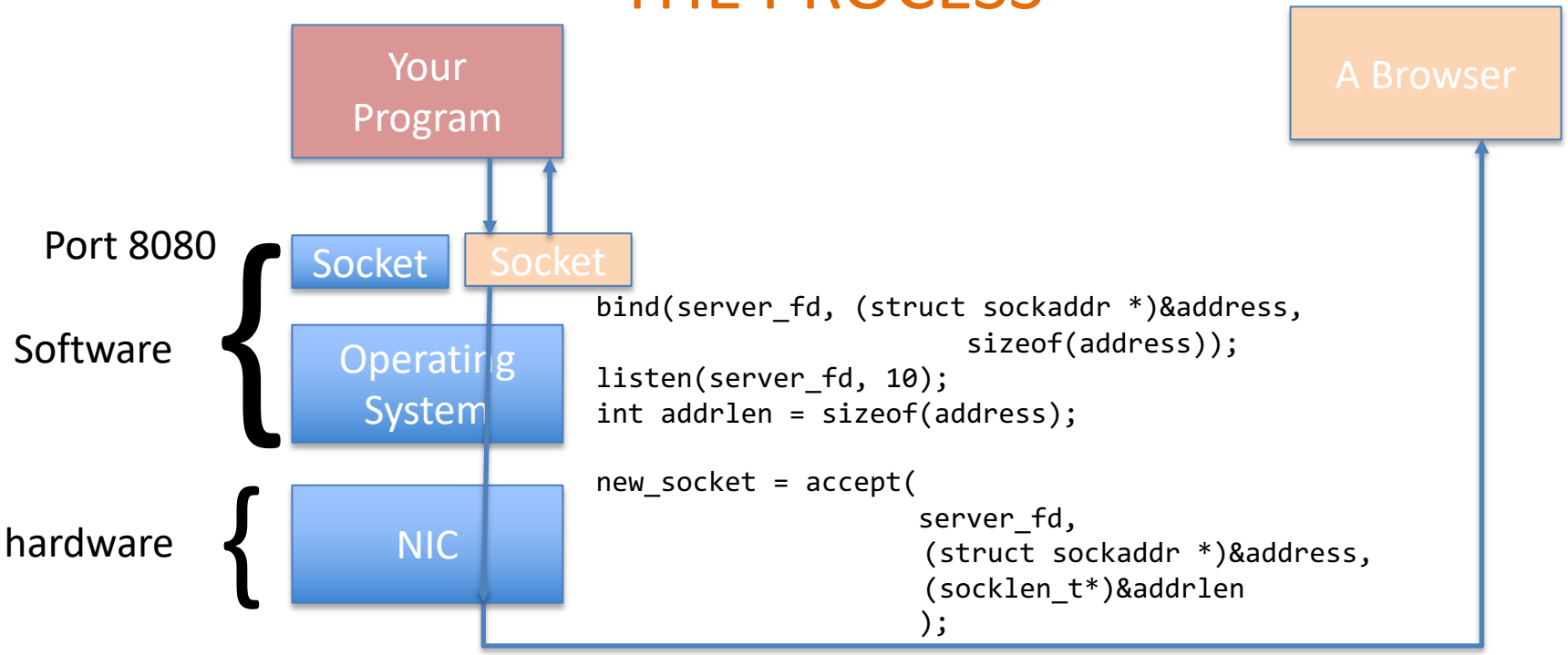
The sockfd argument is a file descriptor that refers to a socket of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of **ECONNREFUSED** or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds.

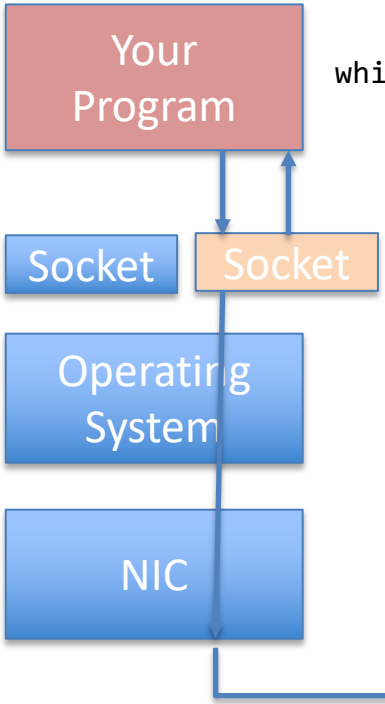
THE PROCESS



THE PROCESS

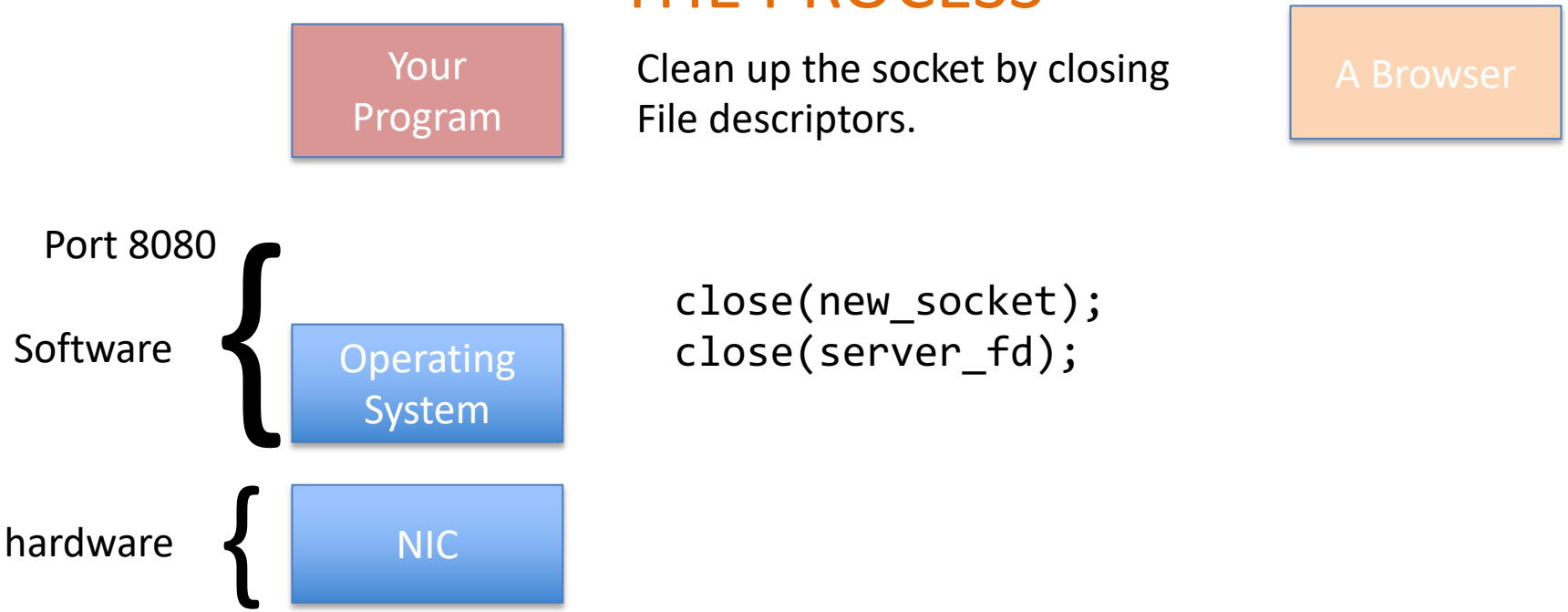

Your Program
A Browser
Port 8080
Socket
Socket
Software
Operating System
hardware
NIC
bind(server_fd, (struct sockaddr *)&address,
sizeof(address));
listen(server_fd, 10);
int addrlen = sizeof(address);
new_socket = accept(
server_fd,
(struct sockaddr *)&address,
(socklen_t*)&addrlen
);

SERVE THE PAGE




```
while (1) {
        new_socket = accept(
                             server_fd,
                             (struct sockaddr *)&address,
                              (socklen_t*)&addrlen
                              );
        write(new_socket, "HTTP/1.1 200 OK\n", 16);
        write(new_socket, "Content-Type: text/html\n\n", 25);
        write(new_socket, "<html><body><h1>Hello, World!</h1></body></html>", 44);
```

THE PROCESS




```
close(new_socket);
close(server_fd);
```

LET’S LOOK AT THE CODE FOR EACH STEP

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORT 8080

int main() {
    int server_fd, client_fd;
    struct sockaddr_in address;

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    bind(server_fd, (struct sockaddr *)&address, sizeof(address));
    listen(server_fd, 10);
    int addrlen = sizeof(address);

    while (1) {
        new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
        write(new_socket, "HTTP/1.1 200 OK\n", 16);
        write(new_socket, "Content-Type: text/html\n\n", 25);
        write(new_socket, "<html><body><h1>Hello, World!</h1></body></html>", 44);
        close(new_socket);
    }
    close(server_fd);
    return 0;
}
```

DEMO START SERVER ON PORTAL