CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Worksheet 21 Socket Programming - Client

Name (Print):_____ RHIT Username:_____

1. What is a *socket*?

   A **socket** is a [ ] [ ] that enables communication between [ ], which can be on [ ].

2. Which of the following statements about *server* and *client* are **true**?

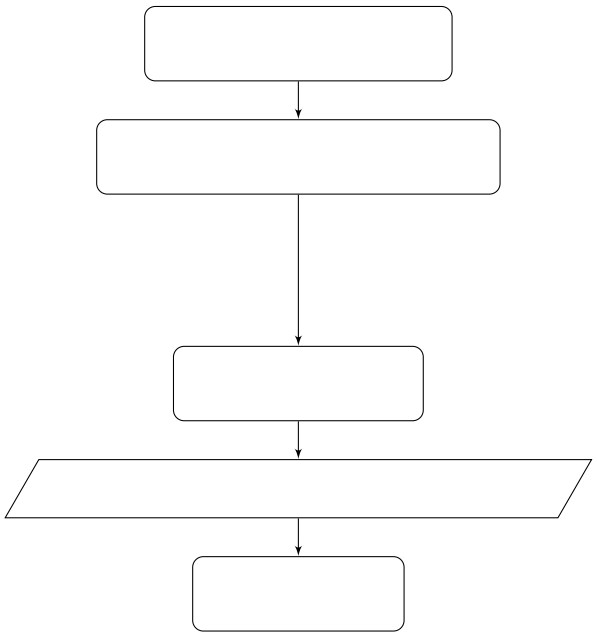
   (a) A server and its client(s) must be on different computers.

   (b) A server can only send information to client(s). And client(s) can only receive information.

   (c) A server usually needs to be launched first before a client can connect to the server.

   (d) One computer can play the roles of both server and client for different applications.

3. Complete the workflow diagram for a `socket` at the **client** side.



4. Follow the demo and start working on `client.c` in `socket-client-practice`.

5. Which file in the practice folder do you edit, and which provided program must be running first?

6. We're writing a client socket to communicate with a sender. Assume the socket has been created (as the variable `sock`) and successfully connected to the server, and the server just sent a 7-byte message `"csse132"` to our client. Check out the different code at the client side and write down the corresponding output. For each version, assume `recv()` returns 7 and writes exactly those seven bytes, without a null terminator. Each example uses the practice's receive call: `sizeof(buffer) - 1` reserves one byte for the terminator. Only the termination step differs. The practice checks errors and EOF before indexing; these snippets assume a successful read. In real TCP communication, a read may return fewer bytes.

   (a) Version 1:

```c
char buffer[20];
ssize_t count = recv(sock, buffer, sizeof(buffer) - 1, 0);
printf("%s", buffer);
```

   Output:

   (b) Version 2:

```c
char buffer[20];
ssize_t count = recv(sock, buffer, sizeof(buffer) - 1, 0);
buffer[count - 1] = '\0';
printf("%s", buffer);
```

   Output:

   (c) Version 3:

```c
char buffer[20];
ssize_t count = recv(sock, buffer, sizeof(buffer) - 1, 0);
buffer[count] = '\0';
printf("%s", buffer);
```

   Output:

7. In the practice, `recv()` requests at most `sizeof(buffer) - 1` bytes. Why reserve one byte? What should the client do when `recv()` returns -1 or 0? Why does the supplied client use a receive loop?

8. **Reflection:** What is one thing from this topic that you found interesting, OR what is one question you are now curious to learn more about?