Section 4: Sockets

CS 162

February 19, 2021

Contents

1 Vocabulary

- **TCP** - Transmission Control Protocol (TCP) is a common L4 (transport layer) protocol that guarantees reliable in-order delivery. In-order delivery is accomplished through the use of sequence numbers attached to every data packet, and reliable delivery is accomplished through the use of ACKs (acknowledgements).
- **Socket** - Sockets are an abstraction of a bidirectional network I/O queue. It embodies one side of a communication channel, meaning that two must be required for a communication channel to form. The two ends of the communication channel may be local to the same machine, or they may span across different machines through the Internet. Most functions that operate on file descriptors like read() or write() work on sockets. but certain operations like lseek() do not.

2 Socket Programming

Note: sockets will be featured more prominently on midterm 3.

2.1 Multi-threaded Echo Server

Write a server implementation that uses multiple threads in a single process. Each connection is handled in its own thread, and threads should be allowed to handle connections concurrently. For simplicity assume `read()` and `write()` do not return short.

```
#define BUF_SIZE 1024

struct addrinfo *setup_address(char *port) {
    struct addrinfo *server;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int rv = getaddrinfo(NULL, port, &hints, &server);
    if (rv != 0) {
        printf("getaddrinfo failed: %s\n", gai_strerror(rv));
        return NULL;
    }
    return server;
}

void *serve_client(void *client_socket_arg) {
    int client_socket = (int)client_socket_arg;
    char buf[BUF_SIZE];
    ssize_t n;

    while ((n = read(client_socket, buf, BUF_SIZE)) > 0) {
        buf[n] = '\0';
        printf("Client Sent: %s\n", buf);

        if (write(client_socket, buf, n) == -1) {
            close(client_socket);
            pthread_exit(NULL);
        }
    }

    close(client_socket);
    pthread_exit(NULL);
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s <port>\n", argv[0]);
        return 1;
    }

    struct addrinfo *server = setup_address(argv[1]);
    if (server == NULL) {
        return 1;
    }
    int server_socket = socket(server->ai_family,
                            server->ai_socktype, server->ai_protocol);
    if (server_socket == -1) {
        return 1;
    }
    if (bind(server_socket, server->ai_addr,
         server->ai_addrlen) == -1) {
            return 1;
    }
    if (listen(server_socket, 1) == -1) {
        return 1;
    }

    while (1) {
        int connection_socket = accept(server_socket, NULL, NULL);
        if (connection_socket == -1) {
            perror("accept");
            pthread_exit(NULL);
        }

        pthread_t handler_thread;
        int err = pthread_create(&handler_thread, NULL,
                serve_client, (void *)connection_socket);
        if (err != 0) {
            printf("pthread_create: %s\n", strerror(err));
            pthread_exit(NULL);
        }
        pthread_detach(handler_thread);
    }
    pthread_exit(NULL);
}
```