Network Programming Templates

This document contains templates for UDP and TCP client-server programs in C, along with explanatory notes.

UDP Client Template

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

#define MAX_BUFFER 1024
#define SERVER_PORT 8888

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[MAX_BUFFER];

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));

    // Configure server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  // Change to
server IP if needed

    while (1) {
        printf("Enter message: ");
        fgets(buffer, MAX_BUFFER, stdin);
        buffer[strcspn(buffer, "\n")] = 0;  // Remove newline

        // Send message to server
        sendto(sockfd, buffer, strlen(buffer), 0, (struct
sockaddr*)&server_addr, sizeof(server_addr));

        // Receive response from server
        int n = recvfrom(sockfd, buffer, MAX_BUFFER, 0, NULL, NULL);
        buffer[n] = '\0';
        printf("Server: %s\n", buffer);
    }
```

```
    close(sockfd);
    return 0;
}
```

Notes:

- UDP is connectionless, so no explicit connection is established.
- `sendto()` and `recvfrom()` are used for sending and receiving data.
- The server's address is specified for each `sendto()` call.
- No handshaking occurs between client and server.

UDP Server Template

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_BUFFER 1024
#define SERVER_PORT 8888

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    char buffer[MAX_BUFFER];
    socklen_t client_len = sizeof(client_addr);

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    memset(&client_addr, 0, sizeof(client_addr));

    // Configure server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(SERVER_PORT);

    // Bind socket to server address
    if (bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
```

```
    printf("UDP Server listening on port %d...\n", SERVER_PORT);

    while (1) {
        // Receive message from client
        int n = recvfrom(sockfd, buffer, MAX_BUFFER, 0, (struct
sockaddr*)&client_addr, &client_len);
        buffer[n] = '\0';
        printf("Client: %s\n", buffer);

        // Process the message (echo back in this example)
        sendto(sockfd, buffer, strlen(buffer), 0, (struct
sockaddr*)&client_addr, client_len);
    }

    close(sockfd);
    return 0;
}
```

Notes:

- The server binds to a specific port and listens for incoming datagrams.
- `recvfrom()` provides the client's address, which is used to send responses.
- The server can handle multiple clients without maintaining connection state.

TCP Client Template

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_BUFFER 1024
#define SERVER_PORT 8888

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[MAX_BUFFER];

    // Create TCP socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));

    // Configure server address
    server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  // Change to
server IP if needed

    // Connect to server
    if (connect(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    while (1) {
        printf("Enter message: ");
        fgets(buffer, MAX_BUFFER, stdin);
        buffer[strcspn(buffer, "\n")] = 0;  // Remove newline

        // Send message to server
        send(sockfd, buffer, strlen(buffer), 0);

        // Receive response from server
        int n = recv(sockfd, buffer, MAX_BUFFER, 0);
        if (n <= 0) {
            printf("Server disconnected\n");
            break;
        }
        buffer[n] = '\0';
        printf("Server: %s\n", buffer);
    }

    close(sockfd);
    return 0;
}
```

Notes:

- TCP is connection-oriented, so `connect()` is called to establish a connection.
- `send()` and `recv()` are used for sending and receiving data.
- The connection remains open for multiple exchanges until explicitly closed.

TCP Server Template

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_BUFFER 1024
#define SERVER_PORT 8888
```

```c
int main() {
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    char buffer[MAX_BUFFER];
    socklen_t client_len = sizeof(client_addr);

    // Create TCP socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    memset(&client_addr, 0, sizeof(client_addr));

    // Configure server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(SERVER_PORT);

    // Bind socket to server address
    if (bind(server_fd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("TCP Server listening on port %d...\n", SERVER_PORT);

    while (1) {
        // Accept client connection
        if ((client_fd = accept(server_fd, (struct
sockaddr*)&client_addr, &client_len)) < 0) {
            perror("Accept failed");
            continue;
        }

        printf("New client connected\n");

        while (1) {
            // Receive message from client
            int n = recv(client_fd, buffer, MAX_BUFFER, 0);
            if (n <= 0) {
                printf("Client disconnected\n");
                break;
            }
            buffer[n] = '\0';
```

```
            printf("Client: %s\n", buffer);

            // Process the message (echo back in this example)
            send(client_fd, buffer, strlen(buffer), 0);
        }

        close(client_fd);
    }

    close(server_fd);
    return 0;
}
```

Notes:

- The server uses `listen()` to wait for incoming connections.
- `accept()` is called to create a new socket for each client connection.
- The server can handle multiple clients by creating a new process or thread for each connection (not implemented in this basic example).
- The inner while loop handles communication with a single client until disconnection.

General Notes:

1. Error Handling: All templates include basic error handling using `perror()` and `exit()`.
2. Buffer Management: Always ensure proper null-termination of received data.
3. Port Numbers: The templates use port 8888. Change this as needed, but remember that ports below 1024 typically require root privileges.
4. IP Addresses: The client templates use "127.0.0.1" (localhost). Change this to the actual server IP for non-local connections.
5. Closing Sockets: Always close sockets when they're no longer needed to free up system resources.
6. Compilation: Compile these programs with gcc: `gcc -o program_name program_name.c`
7. These are basic templates and may need additional features like multi-threading for handling multiple clients simultaneously in a real-world scenario.