

Aufgabe 1.1 Testen der API und des Echo-Programms:

Kopieren Sie den Quellcode der CNAI-API mit ihren Anwendungsbeispielen `chapter3-api.tar` von <http://www.netbook.cs.purdue.edu/cnaiapi/> oder vom Skripte-Server, Verzeichnis `skripte\bachelor\betriebssysteme_netze_1\ss06\materialien\cnai_api` in eines Ihrer Verzeichnisse und entpacken Sie das tar-Archiv.

Entpacken des Archivs:
`tar -zxvf chapter3-api.tar`

Übersetzen Sie die API sowie den mitgelieferten echo-client. Eine Anleitung dazu finden Sie in der mit der API mitgelieferten README-Datei (auch auf dem Skripte-Server zu finden).

Hinweis: Die README-Datei scheint nicht mehr ganz zur API zu passen. Statt `compile_...` heißen die dort genannten Verzeichnisse nun `make_...`

In den Ordner `make-linux` wechseln und dort eingeben:
`make apps`

Machen Sie sich anhand der Header-Dateien und Quellcodes mit der Funktionsweise der CNAI-API, der Verwendung der CNAI-API-Funktionen und des echo-clients vertraut.

In den Ordner `../apps` wechseln, dort kann man den Quellcode einsehen.

Testen Sie den echo-client. Verwenden Sie dabei als echo-server zunächst den standardmäßig auf einigen Linux-Rechnern, im FB I z.B. auf `wspool1serv` unter der "Applikationsnummer" 7 installierten echo-Dienst.

Zurück im Ordner `make-linux` kann man den Client oder Server öffnen.

Server: `./echoserver 7`
Client: `./echoclient localhost 7`

Man befindet sich nun mit einem Client auf dem Server.

Machen Sie sich anhand des Quellcodes mit dem echo-Server vertraut.

Siehe vorletzten Punkt

Übersetzen und binden Sie den echo-server. Verwenden Sie beim Start des Servers eine auf Ihrem Rechner noch nicht verwendete "Applikationsnummer", z.B. 20001, 20002, o.ä.

Siehe vorletzten Punkt, statt „7“ nimmt man jetzt „20001“ oder „20002“.

Testen Sie echo-client und server
- zunächst mit beiden Programmen auf Ihrem eigenen Rechner (Name: localhost).
- dann mit Client und Server auf verschiedenen Rechnern. Testen Sie, ob Ihr Client tatsächlich mit dem richtigen Server in Kontakt steht, indem Sie den Server während der Kommunikation mit dem Client beenden.

Siehe oben

Aufgabe 1.2 Erweiterung des Echo-Servers

Erweitern Sie den echo-server, so dass dieser eine Protokolldatei mit allen "Echo-Daten" schreibt,

Dazu fügt man ein:

- am Anfang der main-Methode:

```
FILE *datei;
```

- nach der if-Abfrage der Argumentenanzahl:

```
if ((datei=fopen("./protokoll.txt", "a"))==NULL) // Datei anlegen
{
    perror("Fehler bei der Dateierzeugung");
    exit(1);
}
```

- zwischen while((len = recv(conn, buff, BUFSIZE, 0)) > 0) und send_eof(conn);

```
{
    (void) send(conn, buff, len, 0);
    fprintf(datei, buff, sizeof(char), len); // Eingaben in Datei
    schreiben
    fflush(datei);
}
```

dieser sich bei Beendigung eines Clients nicht beendet, sondern auf die nächste Anfrage eines Clients wartet.

Dazu fügt man ein:

- nach dem oben eingefügten Dateiöffnungs-Dialog:

```
while(1) // Endlosschleife
{
```

- nach der Zeile conn = await_contact((appnum) atoi(argv[1]));

```
if (fork()==0) //Erzeugung eines Kindprozesses für die Ab-
{ //handlung der Eingaben eines Clients
```

- nach return 0;

```
}
}
```

Code:

```
/* echoserver.c */

#include <stdlib.h>
#include <stdio.h>
#include <cnaiaapi.h>

#define BUFFSIZE          256

/*-----
 *
 * Program: echoserver
 * Purpose: wait for a connection from an echoclient and echo data
 * Usage:  echoserver <appnum>
 *
 *-----
 */
int
main(int argc, char *argv[])
{
    connection    conn;
    int           len;
    char          buff[BUFFSIZE];
    FILE          *datei; // eingefÃ¼gt: FILE-Zeiger fÃ¼r Protokolldatei

    if (argc != 2) {
        (void) fprintf(stderr, "usage: %s <appnum>\n", argv[0]);
        exit(1);
    }

    /* wait for a connection from an echo client */

    if ((datei=fopen("./protokoll.txt", "a"))==NULL) // eingefÃ¼gt: Datei anlegen
    {
        perror("Fehler bei der Dateierzeugung");
        exit(1);
    }
    while(1) // eingefÃ¼gt: Endlosschleife
    {
        conn = await_contact((appnum) atoi(argv[1]));

        if (fork()==0) //eingefÃ¼gt: Erzeugung eines Kindprozesses fÃ¼r die Ab-
        {           //      handlung der Eingaben eines Clients
            if (conn < 0)
                exit(1);

            /* iterate, echoing all data received until end of file */

            while((len = recv(conn, buff, BUFFSIZE, 0)) > 0)
            {
                (void) send(conn, buff, len, 0);
                fprintf(datei, buff, sizeof(char), len); // eingefÃ¼gt:
                fflush(datei); // Eingaben in Datei schreiben
            }
            send_eof(conn);
            return 0;
        }
    }
}
```