Wichtig: Lesen Sie auch den Teil "**Hinweise zur Aufgabe**" auf diesem Blatt; Spezifikationen in diesem Teil sind ebenfalls einzuhalten!

Aufgabe 2: sister (14.0 Punkte)

Implementieren Sie einen einfädigen Webserver `sister` (**Si**mple **S**ingle-**T**hreaded Serv**er**), der HTTP-GET-Anfragen auf einem wählbaren Port *p* entgegennimmt und Dateien innerhalb eines festen Verzeichnisbaums *dir* ausliefert. Das Programm wird wie folgt aufgerufen:

```
./sister --wwwpath=<dir> [--port=<p>]
```

Wird auf der Befehlszeile kein Port angegeben, soll der Server Verbindungen auf Port 1337 akzeptieren.
Das Programm ist modular aufgebaut und in mehrere Komponenten untergliedert, die separat zu implementieren sind. In einer späteren Aufgabe in diesem Semester werden Sie einzelne Module austauschen bzw. ihre Funktionalität erweitern.

a) Makefile: `Makefile`

Erstellen Sie ein zur Aufgabe passendes Makefile, welches keine ins **make(1)**-Programm eingebauten Makros und Regeln voraussetzt. Das Makefile sollte mit dem Aufruf `make -Rr` funktionieren und die Standardtargets `all` und `clean` mit der üblichen Funktionalität bereitstellen. Verwenden Sie – soweit möglich – Pattern-Regeln. Greifen Sie dabei stets auf Zwischenprodukte (z. B. `sister.o`) zurück. Nutzen Sie die auf der Webseite genannten Compilerflags.

b) Hauptprogramm: `sister.c`

Im Hauptprogramm muss zunächst die Initialisierungs-Funktion des Verbindungs-Moduls aufgerufen werden. Anschließend wird ein TCP/IPv6-Socket erzeugt, der auf dem angegebenen Port auf Verbindungsanfragen wartet (**socket(2)**, **bind(2)**, **listen(2)**).
Für jede eingehende Verbindung (**accept(2)**) wird die Funktion `handleConnection()` aufgerufen und anschließend die nächste Verbindung angenommen.

c) Verbindungs-Modul: `connection-fork.c`

Damit das Hauptprogramm ohne Verzögerung weitere Anfragen annehmen kann, soll für das Erledigen der eigentlichen Arbeit ein neuer Kindprozess erzeugt werden (**fork(2)**). Der Kindprozess ruft die Funktion `handleRequest()` auf, trennt die Verbindung und beendet sich anschließend.
Terminierte Kindprozesse sollen nicht in den Zombie-Zustand übergehen. Konfigurieren Sie dazu die Signalbehandlung des `SIGCHLD`-Signals entsprechend (**sigaction(2)**).
Bei seiner Initialisierung soll sich das Verbindungs-Modul darum kümmern, das Anfrage-Modul zu initialisieren.

d) Anfrage-Modul: `request-http.c`

Dieses Modul kümmert sich um die Kommunikation mit dem Client. Der Client sendet pro aufgebauter Verbindung genau eine Anfrage. Zunächst wird die HTTP-Anfragezeile (Maximallänge: 8192 Zeichen; terminiert mit `\r\n` oder mit `\n`) vom Socket gelesen, die wie im folgenden Beispiel aussieht:

```
GET /doc/index.html HTTP/1.0
```

Der Pfadname enthält hierbei keine Leerzeichen und verweist auf die gewünschte Datei, bei der es sich um eine reguläre Datei handeln muss. Der Dateipfad wird relativ zum auf der Befehlszeile angegebenen WWW-Verzeichnis interpretiert.
Anfragen müssen mit dem String *GET* beginnen und entweder mit *HTTP/1.0* oder *HTTP/1.1* enden, andernfalls soll der Server sie als ungültig zurückweisen.
Die Antwort des Servers besteht aus einer Kopfzeile, die den Bearbeitungsstatus der Anfrage beinhaltet (z. B. *200 OK* oder *404 Not Found*), und dem Inhalt der angeforderten Datei. Im Fehlerfall wird anstelle des Dateiinhalts eine Fehlerseite übertragen. Benutzen Sie zum Erzeugen der Kopfzeilen und Fehlerseiten die Hilfsfunktionen aus dem `i4httools`-Modul.
Beachten Sie, dass der Client **auf gar keinen Fall** Zugriff auf Dateien erhalten darf, die sich jenseits des WWW-Verzeichnisses befinden (→ *Directory-Traversal-Attacke*)! Verwenden Sie die Funktion `checkPath()` aus dem `i4httools`-Modul, um den Pfadnamen jeder Anfrage diesbezüglich zu prüfen.

Hinweise zur Aufgabe:

- Eine ausführliche Beschreibung des HTTP-Protokolls in der Version 1.0 finden Sie unter `http://www.w3.org/Protocols/HTTP/1.0/spec.html`.
- Im Verzeichnis `/proj/i4sp2/pub/aufgabe2` finden Sie Schnittstellenvorgaben für sämtliche Module – sowohl für die von Ihnen zu implementierenden als auch für die Hilfsmodule, die Sie zum Lösen der Aufgabe benutzen können. Die Schnittstellen sind verbindlich einzuhalten und die Header-Dateien dürfen nicht verändert werden. Auf der Übungswebseite finden Sie außerdem eine Doxygen-Dokumentation der APIs.
- Benutzen Sie die Funktionen aus dem `cmdline`-Modul zum Parsen und Auswerten der Befehlszeilenargumente.
- Testen Sie Ihren Webserver z. B. mit dem WWW-Pfad `/usr/share/doc/stl-manual/html` und rufen Sie dann in einem Browser (z. B. Firefox/Iceweasel) die folgende URL auf, woraufhin eine C++-Dokumentation erscheinen sollte:
  `http://localhost:<PORT>/index.html`
- Im Verzeichnis `/proj/i4sp2/pub/aufgabe2` befindet sich zu jedem Modul eine bereits vorkompilierte Objekt-Datei, so dass Sie alle Teilaufgaben getrennt voneinander bearbeiten können.

Erforderliche Dateien: `sister.c` (4 Punkte), `connection-fork.c` (4 Punkte), `request-http.c` (4 Punkte), `Makefile` (2 Punkte)
Bearbeitung: Einzeln

Bearbeitungszeit: 13 Werktage (ohne Wochenenden und Feiertage)
Abgabezeit: 17:30 Uhr