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

Aufgabe 5: mother (14.0 Punkte)

Implementieren Sie ausgehend von Ihrer `sister`-Implementierung einen mehrfädigen Webserver `mother` (**Mo**dular **Th**readed Serv**er**) mit erweiterter Funktionalität, der HTTP-Anfragen auf einem wählbaren Port *p* (falls nicht in der Befehlszeile angegeben: 1337) entgegennimmt und Dateien innerhalb eines festen Verzeichnisbaums *dir* ausliefert. Das Programm wird wie folgt aufgerufen:

```
./mother --wwwpath=<dir> [--port=<p>] [--bufsize=<b>] [--threads=<t>]
```

Das Programm ist modular aufgebaut und in mehrere Komponenten untergliedert, die separat zu implementieren sind. Sie können Ihr (fehlerbereinigtes) Hauptmodul `sister.c` als `mother.c` übernehmen oder alternativ das Programm gegen das `mother.o`-Modul in der zur Verfügung gestellten Bibliothek `libmother.a` binden. Analog können Sie entweder Ihre eigene Semaphor- und Ringpuffer-Implementierung oder die Musterlösung aus der Bibliothek `libjbuffer.a` benutzen.

a) Verbindungs-Abarbeitung durch einen Thread-Pool: `connection-mt.c`

Ersetzen Sie das Verbindungs-Modul durch ein neues, in dem mehrere Arbeiter-Threads (siehe *piper*) die Anfragebearbeitung übernehmen und der Haupt-Thread nur noch für die Verbindungsannahme zuständig ist. Bei der Initialisierung des Moduls werden *t* Arbeiter-Threads (falls nicht in der Befehlszeile angegeben: 4) erzeugt, die dann für die Laufdauer des Programms bestehen bleiben. Verwenden Sie zum Austausch gemeinsamer Daten zwischen den Arbeiter-Threads und dem Haupt-Thread einen Ringpuffer (siehe *jbuffer*) mit *b* Einträgen (Standard: 8). Nach Annahme einer Verbindung trägt der Haupt-Thread den zugehörigen Socket-Deskriptor in den Ringpuffer ein. Die Arbeiter-Threads entnehmen in einer Endlosschleife jeweils einen Deskriptor aus dem Ringpuffer und führen die Bearbeitung der zugehörigen Verbindung durch.
Falls eine Socket-Verbindung getrennt wird, während ein Datenaustausch im Gange ist, stellt das Betriebssystem dem sendenden Prozess ein `SIGPIPE`-Signal zu. Dies würde standardmäßig dazu führen, dass der Webserver sich unerwartet beendet – und wäre ein Einfallstor für *Denial-of-Service*-Angriffe. Sorgen Sie deshalb bei der Initialisierung des Verbindungs-Moduls dafür, dass das `SIGPIPE`-Signal ignoriert wird!

b) Automatische Anzeige von Verzeichnissen: `request-httpx.c`

Erweitern Sie das Anfrage-Modul so, dass nicht nur Dateien ausgeliefert, sondern auch Verzeichnisse aufgelistet werden können. Bezieht sich die übergebene URL auf ein Verzeichnis, so überprüfen Sie zunächst, ob die URL mit einem Schrägstrich (`/`) endet. Sollte dies nicht der Fall sein, würde der Webbrowser kaputte Hyperlinks anzeigen – weisen Sie den Client dann mit Hilfe einer *Moved-Permanently*-Antwort freundlich auf die korrekte URL (mit Schrägstrich am Ende) hin und trennen Sie die Verbindung.
Falls sich in dem angeforderten Verzeichnis eine Datei namens `index.html` befindet, senden Sie deren Inhalt an den Client. Andernfalls geben Sie in alphabetischer Reihenfolge (siehe *wsort*) die Namen aller Verzeichniseinträge aus, die nicht mit einem Punkt (`.`) beginnen (siehe *creeper*). Benutzen Sie zur Formatierung der Ausgabe die Funktionen im Modul `dirlisting`.
Beachten Sie, dass Sie in den Arbeiter-Threads keine Funktionen verwenden dürfen, die als nicht-*reentrant* gekennzeichnet sind (z. B. **strtok(3)**)! Benutzen Sie stattdessen deren *reentrant*-Varianten (siehe entsprechende Man-Pages).

c) Ausführen von Perl-Skripten: `request-httpx.c`

Ermöglichen Sie nun noch die Ausführung von Perl-Skripten, deren Standardausgabe an den Client umgeleitet wird (siehe *clash* und *rush*, **fileno(3)**). Für das Senden einer HTTP-Statuszeile ist das Skript selbst verantwortlich.
Aus Sicherheitsgründen sollen nur Dateien, die auf `.pl` enden und deren Eigentümer das Ausführen-Recht hat, ausgeführt werden. Jedes Perl-Skript soll in einem eigenen Prozess gestartet werden. Achten Sie darauf, die entstehenden Zombies sofort einzusammeln!

Hinweise zur Aufgabe:

- Im Verzeichnis `/proj/i4sp2/pub/aufgabe5` 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 Headerdateien dürfen nicht verändert werden. Auf der Übungswebseite finden Sie außerdem eine Doxygen-Dokumentation der APIs.
- Testen Sie Ihren Webserver z. B. mit dem WWW-Pfad `/proj/i4sp2/pub/aufgabe5/wwwdir`. In diesem Verzeichnis finden Sie einige ausführbare Perl-Skripte sowie die Schnittstellendokumentation der Module.
- Stellen Sie im Verbindungs-Modul sicher, dass die offenen Sockets des Servers nicht an aufgerufene Perl-Skripte vererbt werden (**fcntl(2)**). Falls Sie das `mother`-Hauptmodul aus der vorgegebenen Bibliothek verwenden, können Sie davon ausgehen, dass dort für den Verbindungsannahme-Socket das *Close-on-exec*-Flag bereits gesetzt wird.

Hinweise zur Abgabe:

Erforderliche Dateien: `connection-mt.c` (7 Punkte), `request-httpx.c` (7 Punkte)
Bearbeitung: Zweiergruppen
Abgabe bis spätestens Freitag 24.07.2020, 17:30 Uhr