In den Aufgaben P5 und P6 ergänzen Sie die Kern-Proxy-Funktionalität, nach der Fertigstellung ist Ihr Programm ein multi-threaded Web Proxy mit RAM-Caching. Die Aufgaben W7 bis W9 sind **Wahlaufgaben**. Entscheiden Sie sich für **eine** dieser Aufgaben. Beschreiben Sie in der Log-Datei (oder einer zusätzlichen Textdatei) zunächst Ihren Lösungsansatz und die Wahl neuer Typen und Datenstrukturen. Achten Sie auch auf Synchronisierung: Überlegen Sie sich jeweils, ob durch parallel laufende Threads auf gemeinsam genutzte Datenstrukturen zugegriffen wird und ob diese durch einen Mutex geschützt werden müssen.

Beachten Sie auch die Abgabe-Hinweise vor Aufgabe W7 (Seite 3).

P5. Proxy-Funktionalität

Im Wesentlichen soll jeder Thread die Proxy-Funktion wie folgt umsetzen:

- Über den schon geöffneten Socket (zum Client, also i. d. R. ein Browser) erhält er einen HTTP-Request, bei dem die ersten zwei Zeilen die Form
- `GET http://server/path`
  `Host: server`
- haben. In dieser Request-Nachricht muss der Server-Anteil (`http://server`) entfernt werden, so dass in der ersten Zeile nur noch `/path` übrig bleibt – ansonsten wird der Request unverändert übernommen und an den Webserver geschickt.
- Für die Kommunikation mit dem Webserver wird ein neuer Socket (als Client) benötigt.
- Die IP-Adresse zum Host muss über die Funktion `gethostbyname()` (siehe weiter unten) bestimmt werden.

Um die IP-Adresse zu einem Rechnernamen zu erhalten, können Sie die folgende Funktion verwenden (und müssen mit `#include <netdb.h>` eine Header-Datei einbinden, damit die Funktion `gethostbyname()` bekannt ist):

```
void host2ip (char *hostname, char *ip) {
  struct hostent *h = gethostbyname (hostname);
  unsigned char *adr = h->h_addr_list[0];
  sprintf (ip, "%d.%d.%d.%d", adr[0], adr[1], adr[2], adr[3]);
}
```

Haben Sie damit die IP-Adresse zum Servernamen herausgefunden, können Sie (wie in Folie G-14) eine Client-Verbindung zum Ziel-Server (dort: Port 80) aufbauen.

Wenn auch die Verbindung zum Webserver steht, können Sie den Request anpassen (den Server-Anteil entfernen: aus `GET http://server/path` die kürzere Version `GET /path` machen) und danach den vollständigen (angepassten) Request an den Server schicken.

Danach beginnt das Durchreichen der Server-Antwort: Sie empfangen in einer Schleife Teile der Nachricht vom Server und leiten sie an den Client weiter. Die Schleife läuft solange, bis Sie eine (Teil-) Nachricht der Länge 0 erhalten. Dann schließen Sie beide Ports (jeweils mit `shutdown`, `close`).

Zum Testen konfigurieren Sie Firefox so, dass es Ihren Server als Proxy-Server verwendet; als *HTTP-Proxy* und *Port* tragen Sie `localhost` und `8080` ein. Starten Sie Ihren Proxy-Server und rufen Sie probeweise die Seite http://swf.hgesser.de/bs-sp/test/ auf. Wenn die Darstellung der Seite gelingt, können Sie die dort angezeigten Links anklicken, um sich im für den Test vorbereiteten Bereich zu „bewegen“. Sie können alternativ auch komplexere Webseiten ausprobieren; es reicht aber aus, wenn der Zugriff auf die genannten Testseiten funktioniert. Statt Firefox können Sie auch `wget` und vor dem Einsatz die Variable `http_proxy` auf `http://localhost:8080` setzen.

Speichern Sie Ihr Programm als `projekt05.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt05.log`.

Hinweis: Die Aufgabe ist auch dann erfolgreich gelöst, wenn Ihr Proxy-Server nach der Bearbeitung mehrerer Anfragen abstürzt oder fehlerhaft arbeitet und neu gestartet werden muss – die Fehlersuche ist hier oft anspruchsvoll und im Rahmen dieser Programmierübung *nicht* nötig.

P6. RAM-Caching

In dieser Aufgabe geht es darum, im Hauptspeicher einen Cache anzulegen, der die Inhalte angeforderter Webseiten speichert, damit sie bei wiederholten Abrufen direkt aus dem Speicher ausgeliefert werden können.

a) Definieren Sie eine Struktur

```
struct cache_entry {
  char url[256];     // Web-Adresse
  char *location;    // Speicheradresse
  int counter;       // Anzahl Zugriffe
}
```

und ein Array solcher Einträge:

```
struct cache_entry cache[100] = { 0 };
```

Im Code, der eine Proxy-Anfrage bearbeitet, sollen Sie nun aus dem vom Client erhaltenen HTTP-Request die URL (`http://server/path`) isolieren und den Cache durchsuchen: Wenn Sie einen Index `i` finden, für den `cache[i].url` identisch mit der URL ist (Vergleich über `memcmp()`, nicht über ==), können Sie auf die Anfrage an den Server verzichten. Jeder Zugriff auf im Cache liegende Seiten erhöht den Counter-Eintrag.

Wie kommen nun überhaupt Einträge in den Cache? Bei jeder Anfrage, die nicht aus dem Cache bedient werden kann, holen Sie die Inhalte wie bisher vom Webserver, legen dann aber einen neuen Cache-Eintrag ein.

Ob ein Cache-Eintrag frei ist, erkennen Sie an der Bedingung `(location == NULL)`. Wenn es im Array keinen freien Eintrag gibt, löschen Sie einen – wählen Sie dabei einen Eintrag aus, der weniger oder gleich viele Zugriffe wie alle anderen Einträge hat.

Die Größe des nötigen Speichers (dessen Anfangsadresse in `location` gespeichert werden soll) kennen Sie erst, wenn Sie die Antwort des Servers vollständig empfangen haben – darum können Sie zunächst mit `malloc(BUFLEN)` Speicher in der Größe des Buffers reservieren und vor jedem weiteren `recv()`-Aufruf mit `realloc(location, ...)` die Größe erhöhen. Innerhalb der Schleife werden Sie Befehle der Form

```
  msglen = recv (sd, buf, BUFLEN, 0);      // nächsten Teil der Nachricht lesen
  realloc (location, offset+msglen);       // Speicher vergrößern
  memcpy (location+offset, buf, msglen);   // Teil in Cache kopieren
  offset += msglen;
```

benötigen, um die im jeweiligen `recv()`-Schritt gelesenen Daten an die richtige Stelle im Speicher zu kopieren; die Variable `offset` wird vor der Schleife auf 0 gesetzt.

b) Ergänzen Sie die Struktur um ein Timestamp-Feld `int timestamp`. Beim Anlegen eines neuen Eintrags speichern Sie darin `time(NULL)`, das ist der aktuelle Zeitpunkt (in „UNIX-Zeit“, d. h., die Anzahl der Sekunden, die seit 01.01.1970 0:00 Uhr UTC vergangen sind). Sie müssen dafür die Zeile

```
#include <time.h>
```

ergänzen. Finden Sie bei späteren Requests im Cache einen passenden Eintrag, verwenden Sie diesen nur, wenn die Differenz `time(NULL)-cache[i].timestamp` kleiner als 60 (entspricht einer Minute) ist. Ist die Differenz größer, holen Sie die Inhalte erneut vom Server und aktualisieren damit auch den Cache-Eintrag.

Speichern Sie Ihr Programm als `projekt06.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt06.log`.

Wahlaufgaben W7-W9

Bearbeiten Sie nur eine der drei folgenden Aufgaben.

Geben Sie am Ende ein *Archiv mit allen Code- und Protokolldateien* (auch von den vorangegangenen Aufgaben) ab:

```
zip abgabe.zip Code/*.c Code/*.log
```

Dabei muss mindestens die Code-Datei, die zur Wahlaufgabe gehört, *umfangreich dokumentiert* sein.

W7. Offline-Proxy

Erweitern Sie den Proxy-Server um die Möglichkeit, offline Webinhalte zu betrachten – das kann nur für URLs funktionieren, die vorab in den RAM-Cache eingelesen wurden.

- Wenn der Proxy auf dem Status-Port (8081) die Anfrage `/online` bzw. `/offline` erhält, schaltet er in den entsprechenden internen Status.
- Im Zustand „online“ arbeitet der Proxy wie gewohnt.
- Im Zustand „offline“ passiert bei einem Request (über Port 8080) folgendes:
  - Falls die Seite im RAM-Cache vorliegt, wird sie an den Client weiter geleitet.
  - Falls die Seite nicht im RAM-Cache vorliegt, wird eine Response generiert, die den Benutzer darüber informiert, dass der Proxy im Offline-Modus ist, die Seite noch nicht hat, aber bei der nächsten Gelegenheit für den Benutzer laden wird. Außerdem wird sie in eine Warteschlange eingetragen.
  - Die Warteschlange fürs Herunterladen von Seiten soll Platz für so viele URLs bieten, wie es Einträge im RAM-Cache gibt. Ist die Warteschlange voll, ersetzt ein neuer Eintrag den ältesten vorhandenen.
  - Beim Wechsel in den Zustand „Online“ arbeitet der Proxy als erstes die Warteschlange ab. Er lädt dazu die in der Warteschlange stehenden Seiten in den RAM-Cache. Verwenden Sie dafür denselben Mechanismus, der auch bei der regulären Nutzung Inhalte in den RAM-Cache schreibt.

Testen Sie die neue Offline-Funktion durch eine geeignete Sequenz von Zugriffen auf Webseiten im Offline- und Online-Modus.

Speichern Sie Ihr Programm als `projekt07.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt07.log`.

W8. Ausführliche Statusinformationen und Exit

Über den Aufruf der URL `/status` (am Status-Port 8081) soll der Proxy ausführliche Informationen über den Programmstatus ausgeben. Dazu gehören:

- Speicherplatzverbrauch (summierte Größe aller mit `malloc()` angeforderten Speicherbereiche)
- Liste der letzten 20 Anfragen mit Timestamps (wann die Anfrage kam)
- Liste aller im RAM-Cache vorgehaltenen Einträge (mit URL und Speicherbelegung)

Für das Logging von URLs und Timestamps müssen Sie u. a. `handle_connection()` anpassen, so dass es diese Zusatzinformationen speichert.

Bei der Ausgabe von Timestamps soll ein lesbares Format (Datum, Uhrzeit) und nicht die UNIX-Zeit (Sekunden seit 01.01.1970) verwendet werden. Die URLs sollen anklickbar sein, im HTML-Dokument muss also Code der Form

```
<ul>
<li>2024-01-24 14:15:55: <a href="http://fh-swf.de/">http://fh-swf.de/</a>
...
</ul>
```

stehen.

Außerdem soll das Programm auch eine Liste aller jemals (auch in früheren Läufen des Programms) erfolgten Requests speichern. Das erfordert das Ablegen der Liste in einer Datei, damit die Informationen auch nach Beenden des Programms verfügbar bleiben.

Über den Aufruf der URL `/allstats` (am Status-Port 8081) erhalten Sie diese Liste (jeweils in der Form *Timestamp: URL*).

Der Proxy-Server soll auch ein Löschen dieser Liste ermöglichen, wenn die URL `/deletelog` (am Status-Port) aufgerufen wird.

Zusätzlich soll die URL `/exit` erkannt werden; dann beendet sich der Proxy-Server.

Speichern Sie Ihr Programm als `projekt08.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt08.log`.

W9. Modifizierender Proxy

Die in dieser Aufgabe zu implementierende Funktion schafft die Grundlagen für Proxys, die z. B. Werbung oder andere unerwünschte Elemente aus Webseiten entfernen oder ersetzen.

Die Modifikationen sollen durch eine Konfigurationsdatei beschrieben werden, deren Pfad (z. B. `/home/user/proxy.conf`) Sie im Programm fest verdrahten, sie wird beim Programmstart eingelesen und soll aus mehreren mehrzeiligen Einträge einer der folgenden drei Formen bestehen:

Form 1	Form 2	Form 3

`html`
`remove`
`<b>`
	`html`
	`replace`
	`<script>`
	`<disabled>`
		`jpeg`
		`convert`
		`-resize "50%"`

Nach jedem Eintrag (auch nach dem letzten) folgt eine Leerzeile, an der Ihr Programm das Ende eines Eintrags erkennen kann.

- In der ersten Zeile steht, auf welche Dateitypen die Regel angewendet wird. Werten Sie in der HTTP-Response vom Webserver das Feld `Content-Type:` aus, darin steht z. B. `text/html; charset=...` oder `image/jpeg`.
- Die nächste Zeile gibt mit `remove` an, dass ein Muster aus der Datei entfernt werden soll (nur sinnvoll für HTML). `replace` gibt an, dass ein Muster ersetzt werden soll. `convert` gibt an, dass das externe Programm convert aufgerufen werden soll
- In der dritten und ggf. vierten Zeile folgen Optionen für die Verarbeitung. Bei `remove`- und `replace`-Regeln folgt hier der Suchbegriff. Bei `convert` folgen Optionen, die an das Programm `convert` weiter gegeben werden sollen (z. B. erzeugt `convert eingabe.jpg -resize "50%" ausgabe.jpg` eine neue Version des Bilds, die in Breite und Länge um 50% verkleinert ist). Wenn convert nicht verfügbar ist, installieren Sie ImageMagick nach (`sudo apt install imagemagick`)
- In der vierten Zeile steht bei `replace`-Regeln der Ersatztext.

Für die Verarbeitung der `convert`-Regel (und – wenn Sie wollen – auch der anderen beiden Regeln) reichen Sie die Inhalte über zwei Pipes an einen neuen Prozess weiter, in dem Sie `convert` oder andere Tools laufen lassen; von dort geht es mit der zweiten Pipe zurück in den Proxy-Server. Wenn Ihnen das Handling mit Pipes zu komplex ist, können Sie auch mit temporären Dateien arbeiten.

Der Proxy soll der Reihe nach alle Regeln abarbeiten, die in der Konfigurationsdatei gefunden wurden, und (bei `remove` und `replace`) sämtliche Treffer entfernen/ersetzen.

Speichern Sie Ihr Programm als `projekt09.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt09.log`.