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.

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 14 aus Foliensatz 9) eine Client-Verbindung zum Ziel-Server (dort: Port 80) aufbauen.

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

Danach beginnt das Durchreichen der Server-Antwort: Sie empfangen in einer Schleife Teile der Nachricht vom Server und schicken sie an den Client. 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. In der Firefox-Version, die in der Linux-VM installiert ist, geht das über *Bearbeiten* / *Einstellungen* / Bereich *Erweitert* / Reiter *Netzwerk* / *Einstellungen* / *Manuelle Proxy-Konfiguration*. In die Felder *HTTP-Proxy* und *Port* tragen Sie dann `localhost` und `8080` ein, die übrigen Felder lassen Sie leer. Starten Sie Ihren Proxy-Server und rufen Sie probeweise die Seite http://ohm.hgesser.de/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.

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.

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.

Bonusaufgabe (Extra-Punkte, kann Punktabzug in früheren Aufgaben ausgleichen):

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`.