Dies ist das letzte Aufgabenblatt, das Sie in den verbleibenden acht Präsenzstunden (heute und am 24. Juni) bearbeiten können. Die Aufgaben sind nicht anspruchsvoller als bisherige Aufgaben, aber sie sind spärlicher mit Lösungshinweisen versehen. Das gilt besonders für die Zusatzaufgabe 8, deren Bearbeitung freiwillig ist (und die gar keine Lösungshinweise enthält).

Von den Aufgaben 9 und 10 ist genau eine Aufgabe zu bearbeiten (siehe Erläuterung unten auf dieser Seite).

P7. Konfigurierbarkeit

Ändern Sie Ihr Programm so ab, dass es Kommandozeilenoptionen auswerten kann. Es sollen die folgenden Optionen möglich sein:

- `-p` *`Portnummer`*: ändert die Portnummer. Wenn diese Option nicht verwendet wird, nutzt der Server (wie gehabt) den Port 8080, anderenfalls den angegebenen. Wird über `-p` eine zu niedrige Port-Nummer angefordert, bricht das Programm mit einer Fehlermeldung ab. (Port-Nummern unter 1024 sind reserviert und benötigen beim `bind()`-Aufruf Root-Rechte.)
- `-r` *`Root-Verzeichnis`*: setzt ein alternatives Document-Root-Verzeichnis. Wenn diese Option nicht verwendet wird, wird das über `DOCROOT` festgelegte Document-Root-Verzeichnis verwendet. Wenn mit der Option ein nicht vorhandener Ordner oder ein Ordner mit unzureichenden Zugriffsrechten verwendet wird, bricht das Programm mit einer Fehlermeldung ab.
- Falls noch nicht geschehen, sorgen Sie außerdem dafür, dass der gebundene Port nach dem Programmende wieder verfügbar ist. (Dadurch können Sie den Server sofort erneut mit derselben Portnummer starten.) Das geht, indem Sie nach dem Befehl

  ```
  sd = socket (...);
  ```

  die folgenden Zeilen einfügen:

  ```
  int reuseaddr = 1;
  setsockopt (sd, SOL_SOCKET, SO_REUSEADDR, &reuseaddr, sizeof(reuseaddr));
  ```

Speichern Sie Ihre Lösung als `projekt07.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt07.log`.

P8. Keep-Alive (*Zusatzaufgabe*)

Die Bearbeitung dieser Aufgabe ist freiwillig. Sie können damit 10 % der insgesamt zu erreichenden Punkte zusätzlich erwerben (und damit Schwächen in anderen Aufgaben ausgleichen).

Recherchieren Sie, wie der Keep-Alive-Mechanismus (bei dem Verbindungen aufrecht erhalten werden) funktioniert, und implementieren Sie diesen.

Speichern Sie Ihre Lösung als `projekt08.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt08.log`.

P9 oder P10.

Bearbeiten Sie nur eine der beiden Aufgaben P9 und P10. Welche das ist, stellen Sie folgendermaßen fest:

Notieren Sie die Matrikelnummern aller Mitglieder Ihrer Gruppe. Bei der kleineren der beiden Nummern betrachten Sie die letzte Ziffer. Ist diese ungerade, bearbeiten Sie Aufgabe P9. Ist sie gerade, bearbeiten Sie Aufgabe P10. Sie können mit einer anderen Gruppe tauschen, dann müssen den Tausch beide Gruppen dokumentieren.

P9. Pfadüberprüfung und Intrusion Detection

Ihr Webserver lässt klassische Attacken zu, bei denen ein relativer Pfad angefordert wird, um Systemdateien auszulesen (z. B. der Reihe nach `../etc/passwd`, `../../etc/passwd`, `../../../etc/passwd` usw., in der Hoffnung, dass sich beim Zusammensetzen von Document-Root und angefordertem Pfad der Pfad zur Benutzerdatei `/etc/passwd` ergibt.

a) Passen Sie das Programm so an, dass relative Pfade verboten sind und der Server eine Fehlermeldung zurückgibt.

b) Bauen Sie den Schutzmechanismus zu einer *Intrusion Detection* aus: Bei jedem Versuch, einen relativen Pfad anzufordern, vermerken Sie die IP-Adresse des „Angreifers“ und die Zugriffszeit in einer Tabelle. Kommt von derselben Adresse innerhalb einer Minute ein weiterer Angriffsversuch, zählen Sie für diese IP-Adresse einen Counter hoch und aktualisieren die gespeicherte Zugriffszeit. Wenn dieser den Wert 3 erreicht, tragen Sie die IP-Adresse in eine programminterne Blacklist ein. Alle weiteren Anfragen von dieser IP-Adresse sollen dann ignoriert werden, d. h., die Verbindung wird ohne Fehlermeldung sofort abgebrochen. Wichtig: Nicht die Kombination IP-Adresse/Portnummer ist das Kriterium, sondern nur die IP-Adresse, denn Clients verwenden wechselnde Portnummern.

c) Entwickeln Sie ein kleines Client-Programm, das diesen Angriffsversuch unternimmt. Mit `wget` oder dem Browser funktioniert es nicht, weil diese Anwendungen führende `..`-Anteile im Pfad entfernen: Wenn Sie z. B. `wget http://localhost:8080/../../index.html` eingeben, wird `wget` einfach die Datei `/index.html` (und nicht `../../index.html`) anfordern.

Wie Sie Ihr Programm implementieren (in C oder einer anderen Sprache), spielt dabei keine Rolle; Sie können auch Ihnen bekannte Bibliotheken nutzen oder ein Shell-Skript mit `netcat` bzw. `nc` schreiben.

d) Testen Sie den Angriff (mit Ihrem Client-Programm) und stellen Sie sicher, dass der Server korrekt reagiert und den Client blacklistet. Dass Ihr Server danach noch Anfragen von anderen Rechnern verarbeitet, können Sie nur von einem anderen PC aus testen.

Speichern Sie Ihre Lösung als `projekt09.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt09.log`.

P10. Aktive Inhalte

Webserver können nicht nur statische HTML-Dokumente an Clients ausliefern, sondern Inhalte auch dynamisch erzeugen. In dieser Aufgabe bauen Sie ein entsprechendes Feature in Ihren Server ein. Der Server wird nach den Änderungen angeforderte Dateien mit den Endungen `.shtml` und `.sh` wie folgt verarbeiten:

- Bei einer Datei mit Endung `.sh` wird angenommen, dass es sich um ein Shell-Skript handelt. Es wird ein Kindprozess (`/bin/bash`) gestartet, der dieses Skript ausführt, und die Ausgabe wird an den Client geschickt.
- Bei einer Datei mit Endung `.shtml` wird angenommen, dass es sich um eine HTML-Datei mit aktiven Elementen handelt. Diese müssen Sie parsen und dabei nach Zeilen der Form

  ```
  <!--sh scriptname arg1 arg2 ... -->
  ```

  suchen. Wird eine solche Zeile gefunden, dann passiert folgendes: Ein Kindprozess führt das Kommando `scriptname arg1 arg2 ...` in einer Shell aus. Die Ausgabe wird vom Server-Thread abgefangen und anstelle des obigen HTML-Kommentars eingesetzt.

  Beispiel: Die HTML-Datei enthält nur die Zeilen

  ```
  <p>Ausgabe:
  <!--sh /echo2.sh Hallo -->
  (Ende)</p>
  ```

  und die Skript-Datei `echo2.sh` (die im Document-Root liegt) gibt mit dem Shell-Kommando `echo $1 $1` zweimal das erste Argument aus. Dann soll der Server die folgenden Daten übertragen:

```
<p>Ausgabe:
Hallo Hallo
(Ende)</p>
```

- Enthält eine SHTML-Datei keine aktiven Elemente, wird sie wie eine normale Datei behandelt.

a) Implementieren Sie das beschriebene Verhalten. Ihre Lösung für den Fall mit `.sh`-Dateien sollte so gestaltet sein, dass sie sich bei der komplexeren Bearbeitung der `.shtml`-Dateien recyceln lässt.

b) Erweitern Sie die Implementierung wie folgt: Berücksichtigen Sie die Möglichkeit, dass ein Shell-Skript potenziell unendlich viele Textzeilen produziert (z. B. durch eine einfache Endlosschleife). Die Daten sollen, wie sie anfallen, an den Client übertragen werden – der Server-Thread darf also nicht warten, bis alle Ausgaben des Kindprozesses erfolgt sind.

c) SHTML-Dateien sollen auch mehrere aktive Elemente enthalten können.

Speichern Sie Ihre Lösung als `projekt10.c` und dokumentieren Sie das Laufverhalten in einer Protokolldatei `projekt10.log`.

Hinweise zur Abgabe

- Falls Sie eine Aufgabe nicht vollständig lösen konnten, ergänzen Sie im Kopf der Lösungsdatei (oder in der Protokolldatei, die Sie mit abgeben) Hinweise dazu, welche Teile fehlen, und beschreiben Sie, welche Probleme Sie bei der Lösungsfindung hatten.
- Beachten Sie, dass der Code ausführlich dokumentiert sein soll.
- Erzeugen Sie ein Verzeichnis `sp-ss2014-`*`name`* (wobei *name* der Nachname eines der Gruppenmitglieder ist). In dieses Verzeichnis kopieren Sie alle gut dokumentierten Lösungsdateien. Die Dateien müssen im Kopf als Kommentar die Namen und Matrikelnummern aller Gruppenmitglieder enthalten:

  ```
  // Lösung zu Aufgabe NN
  // 123456 Martin Mustermann
  // 234567 Katja Musterfrau     (diese Zeile nur bei Gruppen)
  ```

- Erstellen Sie im Verzeichnis außerdem eine Datei `abstract-`*`name`*`.txt`, in der Sie kurz einen Vorschlag für das Thema Ihres Projektvortrags aufschreiben. (Wenn Sie eine Gruppe sind, erstellen Sie jeweils eine Datei `abstract-`*`name`*`.txt` für jedes Mitglied.)
- Wechseln Sie dann in den Ordner, der `sp-ss2014-`*`name`* enthält, und erstellen Sie mit

  `tar czf sp-ss2014-`*`name`*`.tgz sp-ss2014-`*`name`*`/`

  ein `tgz`-Archiv, das Sie mir per Mail zuschicken (`h.g.esser@gmx.de`). Ich schicke Ihnen eine Eingangsbestätigung. Falls Sie keine erhalten sollten, fragen Sie bitte nach.
- Die **Deadline** für das Verschicken dieser Mail ist **Samstag, 28.06.2014, 18:00 Uhr**.
- Bereiten Sie einen ca. zehn Minuten dauernden Vortrag vor, in dem Sie Ihren Lösungsansatz beschreiben. Wenn Sie mit einem Partner zusammen gearbeitet haben, werden Sie gemeinsam einen ca. 15 Minuten dauernden Vortrag halten, bei dem Sie sich abwechseln. Nach jedem Vortrag ist ca. fünf Minuten Zeit für Fragen der anderen Teilnehmer (und von mir).

Terminhinweis: Den Alternativtermin für den Vortragstermin 08.07.2014 legen wir am nächsten Dienstag fest. Die Vorträge finden am 01.07. und an dem gewählten Ausweichtermin statt. Ich werde über eine Doodle-Umfrage erfragen, wer an welchem der beiden Termine vortragen kann, und das dann passend aufteilen.