Übungen zur Vorlesung
Betriebssysteme I
Studiengang Informatik
WS 2011/12

AG Betriebssysteme FB3
Prof. Dr. Jan Peleska, Florian Lapschies

Abgabe: 5.01.2012

Übungszettel 4

Fragmentierung von UDP-Paketen

In der Vorlesung wurde Euch die Kommunikation über Sockets vorgestellt. Das User Datagram Protocol (UDP) ist Teil der Internetprotokollfamilie und kann zur verbindungslosen Kommunikation mittels Datagrammen genutzt werden, wenn die falsche Reihenfolge oder der Verlust einzelner Datagramme akzeptabel ist. Bei der Übertragung von UDP-Paketen über ein Internet Protokoll Netzwerk werden UDP-Datagramme als Nutzlast von IP-Datagrammen übertragen. Da die Maximalgröße von übertragbaren IP-Datagrammen von dem jeweiligen Übertragungsmedium abhängt, ist es mitunter nötig, UDP-Datagramme auf einzelne IP-Datagramme zu verteilen. Die Zerlegung eines IP-Datagrammes zu mehreren kleineren IP-Datagrammen nennt man IP-Fragmentierung. Die muss nicht umbedingt direkt beim Versender geschehen, sondern kann auch auf einen der Zwischenstationen wie zum Beispiel einem Router auf dem Weg zum Empfänger erfolgen. Ist ein IP-Datagramm erst einmal fragmentiert, ist es im allgemeinen nicht sichergestellt, dass die einzelnen Fragmente den selben Weg zum Ziel nehmen, in der richtigen Reihenfolge ankommen oder überhaupt ankommen.

Dies erfordert natürlich, dass die IP-Datagrammfragmente auf der Empfängerseite wieder ordentlich zusammengesetzt werden, und genau das ist eure Aufgabe.

Aufgabe 1

Schreibt ein C-Programm `send`, welches über einen gewöhnlichen UDP/IP-Socket Datagramme lokal (also an `localhost`) an Port `41632` versendet. Ein weiteres von euch geschriebene C-Programm `receive` soll auf dem selben Rechner laufen und über einen speziellen Packet-Socket diese möglicherweise fragmentierten IP-Datagramme in Rohform auslesen und sie wieder manuell zum vollständigen UDP-Paket zusammensetzen.

Als Maximum Transmission Unit (MTU) bezeichnet man die maximale Paketgröße, die über ein Protokoll ohne Fragmentierung übertragen werden kann. Die einzelnen MTU-Werte können über den Befehl `ifconfig` für die verschiedenen Netzwerkinterfaces erfragt werden. Das Netzwerkinterface zur lokalen Kommunikation ist dabei das Loopbackdevice (`lo`). Damit es bei dieser Aufgabe auch wirklich zur Fragmentierung kommt müssen die UDP-Pakete größer als die MTU des Loopbackdevices sein. Standardmäßig werden beim Versenden von UDP Paketen über einen Socket alle Pakete, die größer als die MTU sind, mit einem Fehler zurückgewiesen (siehe `man 7 udp`). Dies kann man jedoch umgehen, indem man die sogenannte Path MTU Discovery abschaltet. Hierzu muss für den Socket mit `setsockopt()` die Option `IP_MTU_DISCOVER` auf `IP_PMTUDISC_DONT` gesetzt werden (siehe `man 7 ip`). Zudem ist die MTU für das Loopbackdevice standardmäßig recht groß, so dass Fragmentierung erst bei recht großen Paketen auftritt. Setzt daher die MTU auf 200 Bytes:

```
ethtool -K lo ufo off
ifconfig lo mtu 200
```

Der `receive` soll nun die IP-Datagramme im Rohform auslesen und die Fragmente zusammensetzen. Um die einzelnen noch nicht defragmentierten IP-Datagramme über einen Socket auszulesen, muss er wie folgt als Packet-Socket geöffnet werden (siehe `man 7 packet`):

```
sock = socket(AF_PACKET, SOCK_DGRAM, ntohs(ETH_P_IP))
```

Wird nun von diesem Socket gelesen, bekommt man jeweils ein IP-Datagramm, dass an ein beliebiges Netzwerkinterface und an einen beliebigen Port eures Rechners geschickt wurde, geliefert. Verständlicherweise kann der Socket also nur als `root` erstellt werden. Das bedeutet, dass bestimmte IP-Datagramme aussortiert werden müssen:

- Pakete, deren IP-Header Version nicht die erwartete ist.
- Pakete, deren Quell-IP nicht die erwartete ist.
- Pakete, die kein UDP Datagramm enthalten.
- Pakete, deren Zielport nicht der erwartete ist.
- Wenn ein neues Paket defragmentiert werden soll, alle Pakete, die nicht das ein erstes Fragment darstellen.
- Wenn bereits Fragmente gesammelt wurden, alle Pakete, deren Identifier nicht zu den bereits gesammelten passt.

Gibt für jedes aussortierte Paket eine Begründung auf `stderr` aus. Fertig zusammengesetzte Pakete werden auf `stdout` ausgegeben.

Am Anfang eines IP-Pakets steht stets der IP-Header. Die zugehörige C-Struktur ist `struct ip` in `netinet/ip.h`. Direkt danach folgt die Nutzlast, also das die Einzelteile des UDP-Paket. Am Anfang eines UDP-Pakets steht stets ein UDP-Header. Die zugehörige C-Struktur ist `struct udphdr` in `netinet/udp.h`. Direkt danach folgt die Nutzlast des UDP-Pakets, also die eigentlich Nachricht. In den beiden der Aufgabe beigefügten Dokumenten findet ihr die nötigen Informationen, um die einzelnen UDP-Pakete wieder zusammen zu setzen.

Da die Kommunikation nur lokal auf eurem Rechner abläuft, könnt ihr folgende vereinfachenden Annahmen machen:

- Jede versendete Nachricht kommt genau einmal an.
- Alle Nachrichten kommen in der richtigen Reihenfolge an.
- Es kommt keine Nachricht verfälscht an.

Sendet mit dem `send` Datagramme unterschiedlicher Größe in Form von Text, so dass es auch zur Fragmentierung kommt, und weist nach, dass diese vom `receive` korrekt empfangen werden. Weist anhand Eurer Ausgaben noch, dass Pakete korrekt aussortiert werden.

Hinweise

- Alle in dieser Aufgabe geforderten Programme sollen mit einem Makefile übersetzt werden können. In diesem Makefile sollen Abhängigkeiten des Build-Prozesses richtig erfasst sein. Des weiteren soll das Makefile auch eine Regel `clean` enthalten, die alle Komplilate wieder löscht.
- Achtet darauf, dass Ihr die Rückgaben aller Systemaufrufe auf mögliche Fehler überprüft. Im ungewünschten Fehlerfall soll eine erklärende Ausgabe auf `stderr` erfolgen und das Programm dann mit `exit(EXIT_FAILURE)` (siehe `man 3 exit`) beendet werden. Viele Funktionen setzen im Fehlerfall die Variable `errno` (siehe `man 3 errno`), über die der genaue Fehler identifiziert werden kann. Hilfreich für Fehlerausgaben ist die Funktion `strerror()` (siehe `man 3 strerror`) zur Formatierung von `errno`. Studiert auf jeden Fall die entsprechenden man-Pages, um zu erfahren welche Rückgabewerte Fehler kennzeichnen und welche Fehler auftreten können.
- Achtet darauf, dass Ihr allen Speicher, den Ihr mit `malloc()` alloziert habt, auch zum Programmende wieder mit `free()` freigegeben habt. Bei der Behandlung von Fehlerfällen, bei denen das Programm mit einem Fehlercode beendet wird, darf darauf verzichtet werden. Hilfreich zum Aufdecken von Speicherlecks und Speicherzugriffsfehlern ist das Tool `valgrind`.
- Die Abgabe erfolgt als Ausdruck am Ende der Übung und zusätzlich elektronisch über das Subversion Repository.
- Die Dokumentation der Aufgabenlösung ist in LaTeX anzufertigen.