Systemprogrammierung

Foliensatz G
– Projekt: Web- und FTP-Proxy
– TCP/IP-Grundlagen

Prof. Dr. Hans-Georg Eßer

Wintersemester 2023/24
v1.0 – 17.01.2024

Programmier-Projekt

- Implementierung eines Web-Proxys mit folgenden Features
  - HTTP Proxy (GET, HEAD, POST)
  - Keep-Alive (persistente Verbindung)
  - Caching im RAM plus HEAD-Anfrage
  - Frei wählbare Ports für Proxy-Dienst und Steuerung / Konfiguration
  - beliebig viele parallele Verbindungen, über Threads realisiert
  - Server beendet sich/gibt Statusinfo, wenn das Dokument /exit bzw. /status angefordert wird
  - Zugriff auf FTP-Server

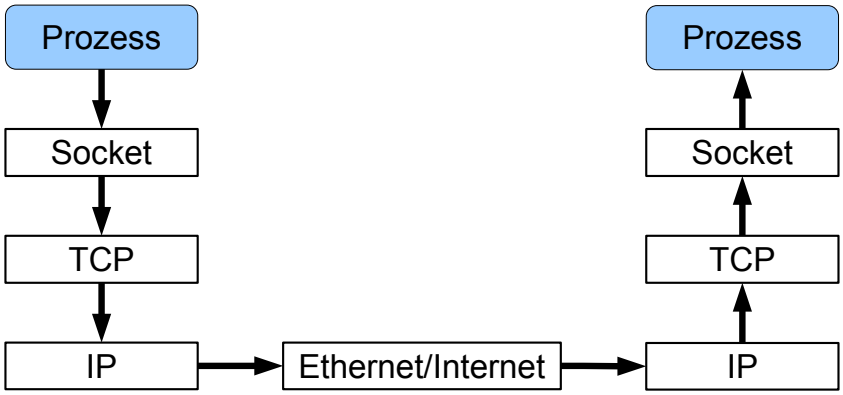
Projekt: Grundlagen

Übersicht

- Datenstrukturen für Sockets
- `socket()`, `bind()`, `listen()`, `accept()`
- `htons()`, `ntohs()`, `inet_ntoa()`, `inet_aton()`
- Socket-Deskriptoren als File-Deskriptoren
- Basics zu HTTP

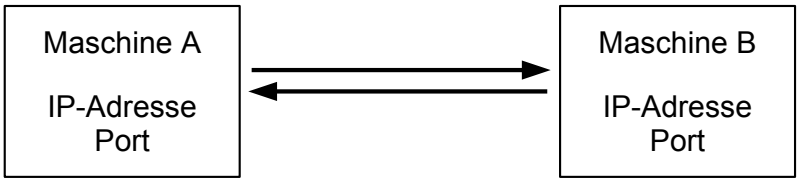
TCP, IP & Co. (1)

- Hier keine Theorie zu Netzwerken, Layers etc.
- Sockets erlauben Kommunikation über TCP/IP



TCP, IP & Co. (2)

- TCP (Transmission Control Protocol):
  - verbindungsorientiert, zuverlässig
  - nutzt IP (Internet Protocol)
- TCP-Verbindung: (IP_1, $Port_1$, IP_2, $Port_2$)



(Port nötig, da mehrere Verbindungen möglich)

TCP, IP & Co. (3)

- `netstat` zeigt (u. a.) offene TCP-Verbindungen an, bekannte Ports erscheinen mit Namen

```
[esser@macbookpro:C-Projekte]$ netstat
Active Internet connections (w/o servers)
Proto Rec-Q Snd-Q  Local Address    Foreign Address         State
tcp4       0      0  macbookpro.59433 faui1s219.inform.https ESTABLISHED
tcp4       0      0  macbookpro.59432 channelproxy-shv.https ESTABLISHED
tcp4       0      0  macbookpro.59428 edge-star-shv-07.https ESTABLISHED
tcp4       0      0  macbookpro.59348 my.ohmportal.de.imaps  ESTABLISHED
tcp4       0      0  macbookpro.59347 imap.1und1.de.imap2    ESTABLISHED
tcp4       0      0  macbookpro.59241 173.194.116.149.https  ESTABLISHED
[...]
[esser@macbookpro:~]$ egrep "^imap|^https" /etc/services | grep tcp
imap2          143/tcp         # Internet Message Access Protocol
https          443/tcp         # http protocol over TLS/SSL
imaps          993/tcp         # imap4 protocol over TLS/SSL
```

TCP, IP & Co. (4)

- `netstat -a` zeigt zusätzlich Server an, d. h., Sockets, die auf Verbindungsanfragen warten:

```
[esser@macbookpro:C-Projekte]$ netstat -a
Active Internet connections (servers and established)
Proto Rec-Q Snd-Q  Local Address    Foreign Address        State
tcp4       0      0  macbookpro.59433 faui1s219.inform.https ESTABLISHED
tcp4       0      0  *.http-alt       *.*                    LISTEN
tcp4       0      0  macbookpro.59432 channelproxy-shv.https ESTABLISHED
[...]
```

- Ubuntu: `netstat` in Paket `net-tools` enthalten;

  `sudo apt install net-tools`

Datenstrukturen

- Socket-Deskriptoren
  - einfache Integers, wie File-Deskriptoren
  - `socket()` & `accept()` geben Socket-Deskriptoren zurück

```
struct sockaddr_in {
    short            sin_family;   // Typ,  z. B. AF_INET
    unsigned short   sin_port;     // Port, z. B. htons (8080)
    struct in_addr   sin_addr;     // kodierte Adresse
    char             sin_zero[8];  // freier Platz
};
```

- IP-Adressen

```
struct in_addr {
    unsigned int s_addr;           // mit inet_aton() füllen
                                   // oder auf INADDR_ANY setzen
};                                 // 32 Bit für IP-Adresse
```


Sockets erzeugen

Zwei Arten von Sockets (für unsere Zwecke)

- generischer Socket, der mit `socket()` erzeugt wird

  → kann verwendet werden,

  - um einen TCP-Port zu binden; bleibt dann dauerhaft „in Betrieb“ (Server)
  - um eine Verbindung zu einem Server aufzubauen (Client)

- Verbindungs-Socket, der mit `accept()` erzeugt wird

  → ist nur für eine konkrete Verbindung (mit einem Client) zuständig

Sockets erzeugen

```
int
socket(int domain, int type, int protocol);

int
connect(int socket, const struct sockaddr *address,
    socklen_t address_len);

int
bind(int socket, const struct sockaddr *address,
    socklen_t address_len);

int
listen(int socket, int backlog);

int
accept(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

Client

Server

Sockets erzeugen

- Server
  - `sd = socket ()`
  - Server-Adresse konfigurieren
  - `bind (sd, Server-Adresse)`
  - `listen (sd)`
  - `conn = accept (sd, &Client-Adresse)`
    (zweiter Socket!)
- Client
  - `sd = socket ()`
  - Ziel-Adresse konfigurieren
  - `connect (sd, &Ziel-Adresse)`

TCP-Server erzeugen (1)

```
int sd;   // socket descriptor for server
struct sockaddr_in server;

sd = socket (PF_INET, SOCK_STREAM, 0);
// PF_INET: Protocol Family, Internet,  SOCK_STREAM: TCP
server.sin_port        = htons (PORT);
server.sin_addr.s_addr = INADDR_ANY;
server.sin_family      = AF_INET;     // Address Family, Internet

int conn;  // socket descriptor for connection
#define SOCKADDR_SIZE sizeof(struct sockaddr_in)
int clilen = SOCKADDR_SIZE;
struct sockaddr_in client;

bind (sd, (struct sockaddr *)&server, SOCKADDR_SIZE);
listen (sd, 0);
```

TCP-Server erzeugen (2)

```
conn = accept (sd, (struct sockaddr *) &client, &clilen);

char buf[bufsize];

// lesen: wie aus Datei
readbytes = read (conn, &buf, bufsize);

// schreiben: wie in Datei
write (conn, &buf, n);

// diese Verbindung schließen
shutdown (conn, SHUT_RDWR);
close (conn);
```


TCP-Client erzeugen

```
int sd;   // socket descriptor for server
struct sockaddr_in client;
sd = socket (PF_INET, SOCK_STREAM, 0);

client.sin_port        = htons(PORT);
client.sin_addr.s_addr = inet_addr("192.1.2.3");
client.sin_family      = AF_INET;   // Address Family, Internet

int res = connect (sd, (struct sockaddr *)&client, sizeof(client));
write (sd, request, strlen(request));
int n = read (sd, response, sizeof(response));
```


`send()`, `recv()`

- Statt `write` und `read` besser `send` und `recv` verwenden → speziell für Sockets gedacht
- `write (sd, buffer, length);`
  →
  `send (sd, buffer, length, flags);`
- `read (sd, buffer, length);`
  →
  `recv (sd, buffer, length, flags);`
- jeweils mit `flags = 0` (siehe Manpages, führt zu `send` = `write` bzw. `recv` = `read`)

htons(), ntohs()

- TCP/IP gibt Standard-Byte-Order vor (Network Byte Order, Big-Endian, most-significant byte first)
- Linux-PC verwendet Little-Endian (least-significant byte first)
- `htons()` und `ntohs()` konvertieren Portnummern: **h**ost **to** **n**etwork bzw. **n**etwork **to** **h**ost
- darum:

```
server.sin_port = htons(PORT);
```


inet_ntoa()

- Aus Adress-Eintrag IP-Adresse des Clients mit `inet_ntoa()` auslesen:

```
struct sockaddr_in client;
accept (sd, (struct sockaddr *)&client, &clilen);
printf ("Client-Adresse: %s \n",
        inet_ntoa (client.sin_addr));
```

- wertet `client.sin_addr.s_addr` aus

```
struct sockaddr_in {
  sa_family_t     sin_family;   /* AF_INET */
  in_port_t       sin_port;     /* Port number */
  struct in_addr  sin_addr;     /* IPv4 address */
};
```

```
struct in_addr {
  in_addr_t s_addr;
};

typedef uint32_t in_addr_t;
```


Demo: Echo-Server

- TCP Echo Server
- Quelle:

`https://github.com/mafintosh/echo-servers.c/blob/master/tcp-echo-server.c`

HTTP

- TCP-Verbindung über Sockets, ASCII-Protokoll
- Client → Server:

```
GET /index.html HTTP/1.1
Host: domainname.top
User-Agent: Mozilla/5.0 (...)
```

- Server → Client:

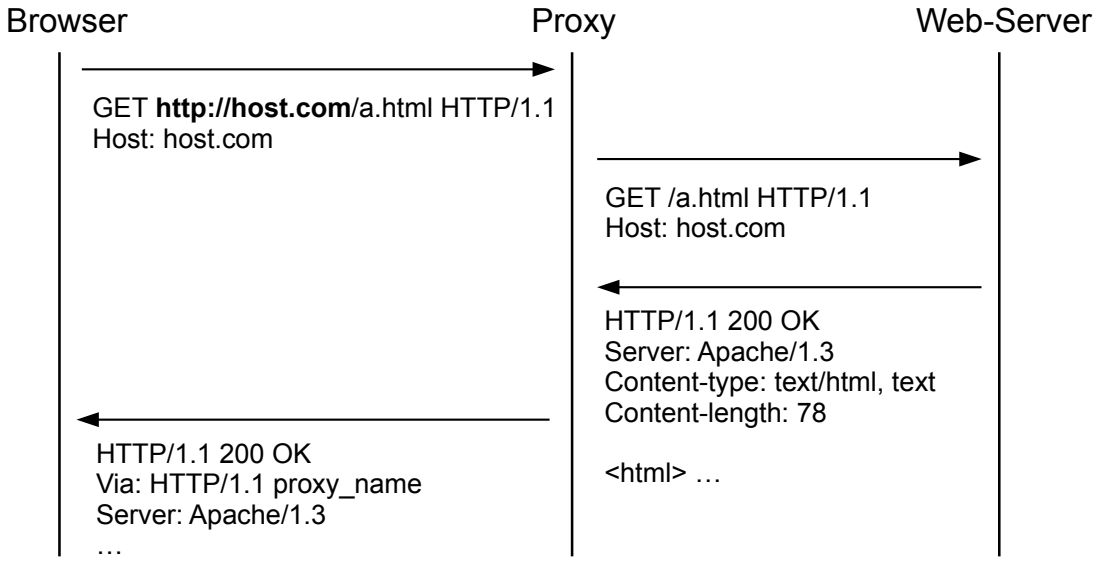
```
HTTP/1.1 201 OK
Content-Type: text/html

<html>
...
```

```
HTTP/1.1 404 Not found
Content-Type: text/html

<html>
...
```


Proxy-Grundlagen



Proxy-Grundlagen

- Browser schickt an Proxy-Server eine Anfrage der folgenden Form:

```
GET http://google.de/ HTTP/1.1
Host: google.de
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:40.0)
  Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: SID=DQAAAF [...]
Connection: keep-alive
```

- Anders als bei normaler HTTP-Anfrage: enthält Domain (`http://google.de/` statt `/`)

Schneller Test mit Squid in Docker-Container

- Container mit Squid:

```
$ docker run -d --name squid-container -e TZ=UTC -p 3128:3128 \
  ubuntu/squid:5.2-22.04_beta
```

- zweifacher Test-Download mit wget:

```
$ export http_proxy=localhost:3128
$ wget swf.hgesser.de/index.html
$ wget swf.hgesser.de/index.html
```

- Log im Container:

```
$ docker exec -it squid-container bash
# tail /var/log/squid/access.log
1705472535.805     31 172.17.0.1 TCP_MISS/200 390
  GET http://swf.hgesser.de/test.txt -
  HIER_DIRECT/217.160.135.96 text/plain
1705472541.534     14 172.17.0.1 TCP_REFRESH_UNMODIFIED/200 396
  GET http://swf.hgesser.de/test.txt -
  HIER_DIRECT/217.160.135.96 text/plain
```