Portieren von Anwendungen auf IPv6

Michael Stapelberg

2. IPv6-Kongress

20. Mai 2010

Über mich

- Student an der Uni Heidelberg
- seit über 10 Jahren im Internet
- Autor IPv6-fähiger Programme wie `mxallowd`, `i3status`
- IPv6-Patches für `radvd`, `mit-krb5`, `vsftpd`, ...
- IPv6-interessiert wegen ~~NAT~~, ~~DHCP~~ und statischen IPs mit globaler Erreichbarkeit

Vorwort zum Beispielcode

- nur die relevanten Ausschnitte (mit Variablendeklarationen)
- alle Beispiele und diese Folien finden Sie (vollständig) auf `http://michael.stapelberg.de/ipv6-beispiele.tar.bz2`
- aus Platzgründen wird z.B. `EXIT_FAILURE` durch 1 ersetzt
- Verwendung der `err()` und `errx()`-Funktionen (BSD-Erweiterungen) zwecks kürzerer Fehlerbehandlung:

```
if (ret == -1) {
    perror("function()");
    exit(EXIT_FAILURE);
}

/* selber Effekt, weniger code: */
if (ret == -1)
    err(1, "function()");
```

Inhalt

1. Problemkandidaten erkennen
2. IP-Adressen darstellen
3. Auflösen von Hostnames
4. Verbindungen aufbauen
5. Server-sockets
6. IPv6-Adressen parsen
7. IPv6 in Perl, Erlang, Ruby, Python

Erkennen von Problemkandidaten

- Aussagekräftige Fehlermeldung (oder auch irgendeine Fehlermeldung)
- Programmfluss in `strace` verfolgen
- Binaries, die `gethostbyname(3)` benutzen:
  ```
  $ nm ./example-old | grep gethostbyname
   U gethostbyname@@GLIBC_2.2.5
  ```
- Programme mit IPv6-only-Hostnames füttern
- Link-local-Adressen verwenden (Verbindungen/Serveraddresse)
  `fe80::21c:c0ff:fe7e:4776%eth0`

IP-Adressen darstellen

- `Connecting to streamsaver.sur5r.net[32.1.4.112]: 80...`

- Eigentlich gehört dieser Server nicht in 32.0.0.0/8 (AT&T Global Network Services, LLC)

Dezimal	32	01	04	112
Hex	20	01	04	70

- `streamsaver.sur5r.net has IPv6 address 2001:470:9e42:0:250:baff:fe21:c27f`

- . . . sondern in 2001:470::/32 (Hurricane Electric, Inc.)

IP-Adressen darstellen (2)

- `getnameinfo (3) - address-to-name translation in protocol-independent manner` (POSIX.1-2001)

```
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
```

- Hostname/Port jeweils in Textform (falls möglich), für IP/Portnummer `NI_NUMERICHOST` bzw. `NI_NUMERICSERV` angeben

- Internationalized Domain Name (IDN) möglich, flag `NI_IDN`

IP-Adressen darstellen (Beispiel)

`fd` beinhaltet einen gültigen Socket (z.B. eingehende Verbindung)

```
char buf[INET6_ADDRSTRLEN+1];
struct sockaddr_storage addr;
int ret;

socklen_t addrlen = sizeof(addr);
if (getsockname(fd, (struct sockaddr*)&addr, &addrlen) == -1)
    err(1, "getsockname()");

memset(buf, 0, INET6_ADDRSTRLEN+1);
if ((ret = getnameinfo((struct sockaddr*)&addr, addrlen,
                       buf, sizeof(buf),
                       NULL, 0,
                       NI_NUMERICHOST)) != 0)
    errx(1, "getnameinfo(): %s\n", gai_strerror(ret));

printf("New connection from %s\n", buf);
```

Auflösen von Hostnames

- Traditionell: `gethostbyname(3)` etc.
- `getaddrinfo(3)`
  - kann mehrere Records zurückgeben
  - Adressfamilien-Präferenz
  - beachtet `RFC 3484 (Default Address Selection for IPv6)`
  - unterstützt Internationalized Domain Names (IDN)
  - AI_ADDRCONFIG

Auflösen von Hostnames (Beispiel)

```
int ret;
struct addrinfo *res, *walk;
struct addrinfo hints;
memset(&hints, 0, sizeof(struct addrinfo));

hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
ret = getaddrinfo("www.google.de", "http", &hints, &res);
if (ret != 0)
    errx(1, "getaddrinfo(): %s\n", gai_strerror(ret));

for (walk = res; walk != NULL; walk = walk->ai_next) {
    char buf[INET6_ADDRSTRLEN+1];
    memset(buf, 0, INET6_ADDRSTRLEN+1);
    ret = getnameinfo((struct sockaddr*)walk->ai_addr,
                      walk->ai_addrlen, buf, sizeof(buf),
                      NULL, 0, NI_NUMERICHOST);
    if (ret != 0)
        errx(1, "getnameinfo(): %s\n", gai_strerror(ret));
    printf("resolved to %s\n", buf);
}
```

Verbindungen aufbauen

- zunächst via IPv6, dann IPv4 (RFC 3484)
- `getaddrinfo(3)` liefert die richtige Reihenfolge
- in einer Schleife alle Ergebnisse durchprobieren, bis die Verbindung aufgebaut werden kann

Verbindungen aufbauen (Beispiel)

```
int open_connection(struct addrinfo *res) {
    int fd;
    struct addrinfo *walk;
    for (walk = res; walk != NULL; walk = walk->ai_next) {
        fd = socket(walk->ai_family, walk->ai_socktype,
                    walk->ai_protocol);
        if (fd < 0) {
            perror("socket()");
            return -1;
        }
        if (connect(fd, walk->ai_addr, walk->ai_addrlen) < 0) {
            perror("could not connect()");
            close(fd);
            fd = -1;
            continue;
        }
        return fd;
    }
    return -1;
}
```

Server-sockets erstellen (1 Socket)

- Umstellen mancher Programme am einfachsten mit `IPV6_V6ONLY` (RFC 3542)
- IPv4-Verbindungen möglich (`::ffff:192.168.1.2`, RFC 2553)
- Standardwert in `/proc/sys/net/ipv6/bindv6only`
- Probleme bei dieser Lösung:
  - Notation `::ffff:192.168.1.2` ist ungewohnt und inkonsistent
  - Drittsoftware kann Logfiles evtl. nicht verarbeiten
  - Keine spezifischen Serveradressen möglich (nur Wildcard)

Server-sockets erstellen (Beispiel)

```
struct addrinfo *res, *walk;
struct addrinfo hints;
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = PF_INET6;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
if ((ret = getaddrinfo(NULL, "8080", &hints, &res)) != 0)
    errx(1, "getaddrinfo(): %s\n", gai_strerror(ret));
for (walk = res; walk != NULL; walk = walk->ai_next) {
    if ((fd = socket(walk->ai_family, walk->ai_socktype,
                walk->ai_protocol) < 0)
        err(1, "socket()");
    int off = 0;
    if (setsockopt(fd, IPPROTO_IPV6, IPV6_V6ONLY,
                &off, sizeof(off)) == -1)
        err(1, "setsockopt()");
    if (bind(fd, walk->ai_addr, walk->ai_addrlen) != 0)
        err(1, "bind()\n");
    listen(fd, 5);
    break;
}
```

Server-sockets (dual-stack)

- Mehrere Sockets erstellen, einen für jede konfigurierte Adresse
- `IPV6_V6ONLY` explizit anschalten (für IPv6-Sockets)
- `select(3)` o.ä. nutzen (oder z.B. `libev`)
- Fehlerbehandlung fehlt aus Platzgründen im Beispiel, siehe tar-Archiv!

Server-sockets (Beispiel, Teil 1)

```
int n = 0;

struct addrinfo *res, *walk, hints;
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
getaddrinfo(NULL, "9090", &hints, &res);
for (walk = res; walk != NULL; walk = walk->ai_next) {
    int fd = socket(walk->ai_family, walk->ai_socktype,
                    walk->ai_protocol);

    if (walk->ai_family == AF_INET6) {
      int on = 1;
      setsockopt(fd, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof(on));
    }

    bind(fd, walk->ai_addr, walk->ai_addrlen);
    listen(fd, 5);

    sockfds[n++] = fd;
}
```

Server-sockets (Beispiel, Teil 2)

```
fd_set fds;
int highest;
while (1) {
    highest = 0;
    FD_ZERO(&fds);
    for (int i = 0; i < n; i++) {
        FD_SET(sockfds[i], &fds);
        highest = (sockfds[i] > highest ? sockfds[i] : highest);
    }
    select(highest+1, &fds, NULL, NULL, NULL);
    for (int i = 0; i < n; i++) {
        if (!FD_ISSET(sockfds[i], &fds))
            continue;

        struct sockaddr_storage peer;
        socklen_t peersize = sizeof(struct sockaddr_storage);
        int client = accept(sockfds[i], (struct sockaddr*)&peer,
                            &peersize);

        /* ... */
    }
}
```

IPv6-Adressen parsen

- Üblich: Eckige Klammern (wie in URLs, siehe RFC 2732)
- Scope identifier (übernimmt `getaddrinfo(3)` für Sie)
- Mögliche Adressen (Vorsicht beim Validieren!):
  - `streamsaver.sur5r.net`
  - `[2001:470:9e42:0:250:baff:fe21:c27f]`
  - `[2001:470:9e42:0:250:baff:fe21:c27f]:http`
  - `[fe80::250:baff:fe21:c27f%eth0]:80`
  - `[::]:80`
  - `[::ffff:192.168.1.3]:80`

IPv6-Adressen parsen (Beispiel)

```
const char *address = "[2a02:2e0:3fe:100::6]:http";
char *host, *copy, *port;
if ((host = copy = strdup(address)) == NULL)
    err(1, "strdup()");
if (*host == '[') {
    host++;
    if ((port = strchr(host, ']')) != NULL) {
        *port = '\0';
        port++;
        if (*port == ':')
            port++;
        else port = "http";
    }
} else {
    if ((port = strchr(host, ':')) != NULL) {
        *port = '\0';
        port++;
    } else port = "http";
}
int ret = getaddrinfo(host, port, NULL, &addr);
free(copy);
/* addr verwenden... */
```

Perl 5

- IPv6 möglich, sofern man es explizit nutzt
- Module unterstützen oftmals transparent IPv6, z.B. `AnyEvent::Socket`
- ansonsten `Net::INET6Glue`, welches die Standard-Funktionen überschreibt (auch für Drittmodule)

Erlang, Ruby, Python

- Erlang: inet6-Option angeben bei `gen_tcp:connect`, beide Protokolle werden dann transparent unterstützt (auch in inetrc konfigurierbar)
- Ruby: benutzt IPv6 (`socket`, `TCPSocket`), allerdings nicht bevorzugt
- Python: Zugriff nur über Low-Level-API?

Das war's

Fragen?