

Übungen zu Systemprogrammierung 2 (SP2)

Ü1 – Interprozesskommunikation mit Sockets

Christoph Erhardt, Jens Schedel, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 2015 – 20. bis 24. April 2015

https://www4.cs.fau.de/Lehre/SS15/V_SP2

Miniklausur-Termin

- Mögliche Termine für die Miniklausur
 - Dienstag 28.4, 16:00 s.t.
 - Dienstag 28.4, 18:00 s.t.
 - Mittwoch 29.4, 10:00 s.t.
 - Mittwoch 29.4, 18:00 s.t.

Agenda

- 1.1 Organisatorisches
- 1.2 IPC-Grundlagen
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden

21-ClientSockets_handout



21-ClientSockets_handout



© ce, js, jk SP2 (Ü1 | SS 2015) 1 Client-Sockets

1-1

Organisatorisches

Übungsbetrieb

- Rechnerübungen von SP1 und SP2 finden gemeinsam statt
 - Termine siehe https://www4.cs.fau.de/Lehre/SS15/V_SP2/Uebung/univis.ushtml
 - Als *unbetreut* eingetragene Rechnerübungen sind mit Raumreservierung, aber ohne Tutor

SVN-Repository

- URL: <https://www4.cs.fau.de/i4sp/ss15/sp2/<login>>
 - Passwort setzen: /proj/i4sp2/bin/change-password
- Projektverzeichnisse in diesem Semester unter /proj/i4sp2
- SP1-Repository aus dem Vorsemester weiterhin zugreifbar

21-ClientSockets_handout



© ce, js, jk SP2 (Ü1 | SS 2015) 1 Client-Sockets | 1.1 Organisatorisches

1-2



© ce, js, jk SP2 (Ü1 | SS 2015) 1 Client-Sockets | 1.1 Organisatorisches

1-3

- Durchgängige Verwendung von 64-Bit
- Programmiersprache ISO C99
- Betriebssystemstandard SUSv4 (= POSIX.1-2008)

Kompilieren & Linken

- Standard-Flags in SP: `-std=c99 -pedantic -Wall -Werror -D_XOPEN_SOURCE=700`
 - Zum Debuggen zusätzlich `-g`
- In SP2 konsequente Benutzung von Makefiles

Ach, übrigens...

Fehlerabfragen nicht vergessen!

Wir wollen in SP *robuste* Programme schreiben.

Client-Server-Modell

IPC-Grundlagen

Ein **Server** ist ein Programm, das einen **Dienst** (*Service*) anbietet, der über einen Kommunikationsmechanismus erreichbar ist

Server

- **Akzeptiert Anforderungen**, die von außen kommen
- **Führt** einen angebotenen **Dienst aus**
- **Schickt das Ergebnis zurück** zum Sender der Anforderung
- In der Regel als normaler Benutzerprozess realisiert

Client

- Schickt eine **Anforderung an einen Server**
- Wartet auf eine Antwort

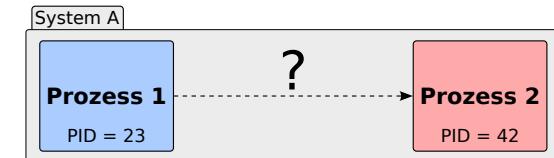
- 1.1 Organisatorisches
- 1.2 IPC-Grundlagen
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden

Kommunikation innerhalb eines Systems

IPC-Grundlagen

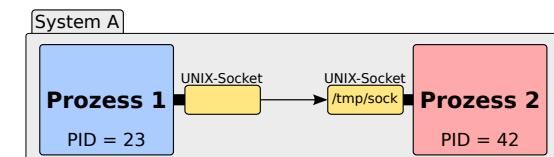
? Wie findet ein Client den gewünschten Dienstanbieter?

- Intuitiv: über dessen Prozess-ID



- **Problem:** Prozesse werden dynamisch erzeugt/beendet; PID ändert sich

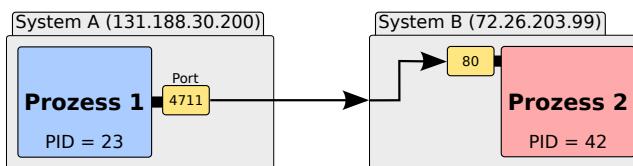
- **Lösung:** Verwendung eines abstrakten „Namens“ für den Dienst



- Prozess 2 ist über den Dienstnamen (hier: einen Dateinamen) erreichbar

? Wie findet ein Client nun den gewünschten Dienstanbieter?

- Über einen zweistufig aufgebauten „Namen“:
 1. Identifikation des Systems innerhalb des Netzwerks
 2. Identifikation des Prozesses innerhalb des Systems
- Beispiel TCP/IP: eindeutige Kombination aus
 1. IP-Adresse (identifiziert Rechner im Internet)
 2. Port-Nummer (identifiziert Dienst auf dem Rechner)



21-ClientSockets_handout

Adressierung von Rechnern im Internet

Internet Protocol, Version 6 (IPv6)

- Nachfolgeprotokoll von IPv4
 - 1998 als Standard verabschiedet, Einführung aktuell im Gange
 - *Dual Stack*: IPv4 und IPv6 parallel betreibbar
- 128-Bit-Adressraum ($\approx 3,4 \cdot 10^{38}$ Adressen)
 - Sollte fürs Erste ausreichen :-)
- Notation: 8 mit : getrennte 2-Byte-Werte in Hexadezimaldarstellung
 - z. B. 2001:638:a00:1e:219:99ff:fe33:8e75
- In der Adresse kann einmalig :: als Kurzschreibweise einer Nullfolge verwendet werden
 - z. B. *localhost*-Adresse: 0:0:0:0:0:0:1 = ::1

21-ClientSockets_handout

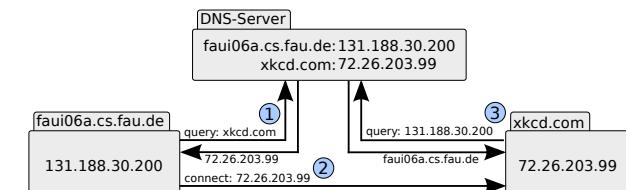
Internet Protocol, Version 4 (IPv4)

- Protokoll zur Bildung eines weltweiten virtuellen Netzwerks auf der Basis mehrerer physischer Netze (durch Routing)
 - 1981 als Standard verabschiedet
- 32-Bit-Adressraum (≈ 4 Milliarden Adressen)
- Notation: 4 mit . getrennte Byte-Werte in Dezimaldarstellung
 - z. B. 131.188.30.200
- Nicht zukunftsfähig wegen des viel zu kleinen Adressraums
 - Alle Adressblöcke in Asien/Pazifikraum (2011), Europa/Nahost (2012) und Lateinamerika/Karibik (2014) bereits vergeben!

21-ClientSockets_handout

Domain Name System (DNS)

- IP-Adressen sind nicht leicht zu merken
- ... und ändern sich, wenn man einen Rechner in ein anderes Rechenzentrum umzieht
- **Lösung:** zusätzliche Abstraktion durchs DNS-Protokoll



21-ClientSockets_handout

1. *Forward lookup*: Rechnername → IP-Adresse
2. Kommunikationsaufbau
3. *Reverse lookup* (im Beispiel optional): IP-Adresse → Rechnername

21-ClientSockets_handout

- Numerische Identifikation eines Dienstes innerhalb eines Systems
- Port-Nummer: 16-Bit-Zahl, d. h. kleiner als 65536
- Port-Nummern < 1024: *well-known ports*
 - Können nur von Prozessen gebunden werden, die mit speziellen Privilegien gestartet wurden (Ausführung als *root*)
 - z. B. ssh = 22, smtp = 25, http = 80, https = 443



Übertragung von Daten

- Beim Austausch von binären Datenwörtern zwischen verschiedenen Rechnern muss Einigkeit über die verwendete *Byteorder* herrschen!
- Beispiel: Kommunikation zwischen x86-Rechner (*little endian*) und SPARC-Rechner (*big endian*)

| Wert | Repräsentation |
|------------|----------------|
| 0xcafebabe | big endian |
| | little endian |

0 1 2 3

| | | | |
|----|----|----|----|
| ca | fe | ba | be |
| be | ba | fe | ca |

- **Definierter Standard:** Netzwerk-Byteorder = *big endian*
 - Konvertierung vor dem Senden und nach dem Empfangen nötig
 - Ausnahme: bei Übertragung von Einzelbytes bzw. Byte-Strings



Verbindungsorientiert (*Datenstrom/Stream*)

- Gesichert gegen Verlust und Duplizierung von Daten
- Reihenfolge der gesendeten Daten bleibt erhalten
- Vergleichbar mit einer UNIX-Pipe – allerdings bidirektional
- Implementierung: Transmission Control Protocol (TCP)

Paketorientiert (*Datagramm*)

- Schutz vor Bitfehlern – nicht vor Paketverlust oder -duplicierung
- Datenpakete können eventuell in falscher Reihenfolge ankommen
- Grenzen von Datenpaketen bleiben erhalten
- Implementierung: User Datagram Protocol (UDP)

Agenda

- 1.1 Organisatorisches
- 1.2 IPC-Grundlagen
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden



- Generischer Mechanismus zur Interprozesskommunikation
- Implementierung ist abhängig von der jeweiligen Kommunikations-Domäne
 - Innerhalb des selben Systems: z. B. UNIX-Socket
 - Adressierung über Dateinamen, Kommunikation über gemeinsamen Speicher, keine Sicherungsmechanismen notwendig
 - Über Rechnergrenzen hinweg: z. B. TCP/UDP-Socket
 - Adressierung über IP-Adresse + Port, nachrichtenbasierte Kommunikation, Sicherungsmechanismen bei TCP
- Beim Verbindungsaufbau sind die entsprechenden Parameter zu wählen
- Anschließende Verwendung im Programm ist transparent
 - ... egal, ob der Kommunikationspartner ein Prozess auf dem selben Rechner oder am anderen Ende der Welt ist

21-ClientSockets_handout



Verbindlungsaufbau durch den Client

IPC-Schnittstelle

- `connect(2)` meldet Verbindungswunsch an Server:


```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

 - `sockfd`: Socket, über den die Kommunikation erfolgen soll
 - `addr`: Beinhaltet „Namen“ (bei uns: IP-Adresse und Port) des Servers
 - `addrlen`: Länge der konkret übergebenen `addr`-Struktur
- `connect()` blockiert solange, bis der Server die Verbindung annimmt oder zurückweist
 - Mehr zum Server in der nächsten Übung
- Socket ist anschließend bereit zur Kommunikation mit dem Server

21-ClientSockets_handout

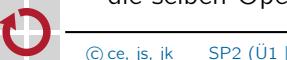


- Sockets werden mit dem Systemaufruf `socket(2)` angelegt:


```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

 - `domain`, z. B.:
 - `PF_UNIX`: UNIX-Domäne
 - `PF_INET`: IPv4-Domäne
 - `PF_INET6`: IPv6-Domäne (*kompatibel zu IPv4, wenn vom OS unterstützt*)
 - `type` innerhalb der gewählten Domäne:
 - `SOCK_STREAM`: Stream-Socket
 - `SOCK_DGRAM`: Datagramm-Socket
 - `protocol`:
 - 0: Standard-Protokoll für gewählte Kombination (z. B. TCP/IP bei `PF_INET(6)` + `SOCK_STREAM`)
- Ergebnis ist ein numerischer Socket-Deskriptor
 - Entspricht einem Datei-Deskriptor und unterstützt (bei Stream-Sockets) die selben Operationen: `read(2)`, `write(2)`, `close(2)`, ...

21-ClientSockets_handout



DNS-Anfragen

Betriebssystemschnittstelle zur IPC

- Zum Ermitteln der Werte für die `sockaddr`-Struktur kann das DNS-Protokoll verwendet werden
- `getaddrinfo(3)` liefert die nötigen Werte:


```
int getaddrinfo(const char *node,
                  const char *service,
                  const struct addrinfo *hints,
                  struct addrinfo **res);
```

 - `node` gibt den DNS-Namen des Hosts an (oder die IP-Adresse als String)
 - `service` gibt entweder den numerischen Port als String (z. B. "25" oder den Dienstnamen (z. B. "smtp", `getservbyname(3)`) an
 - Mit `hints` kann die Adressauswahl eingeschränkt werden (z. B. auf IPv4-Sockets). Nicht verwendete Felder auf 0 bzw. NULL setzen.
 - Ergebnis ist eine verkettete Liste von Socket-Namen; ein Zeiger auf das Kopfelement wird in `*res` gespeichert
- Freigabe der Ergebnisliste nach Verwendung mit `freeaddrinfo(3)`

21-ClientSockets_handout



```
struct addrinfo {
    int           ai_flags;      // Flags zur Auswahl (hints)
    int           ai_family;     // z. B. AF_INET6
    int           ai_socktype;   // z. B. SOCK_STREAM
    int           ai_protocol;   // Protokollnummer
    socklen_t     ai_addrlen;   // Größe von ai_addr
    struct sockaddr *ai_addr;   // Adresse fuer connect()
    char          *ai_canonname; // Offizieller Hostname (FQDN)
    struct addrinfo *ai_next;   // Nächstes Listenelement oder NULL
};
```

- `ai_flags` relevant zur Anfrage von Auswahlkriterien (`hints`)
 - `AI_ADDRCONFIG`: Auswahl von Adresstypen, für die auch ein lokales Interface existiert (z. B. werden keine IPv6-Adressen geliefert, wenn der aktuelle Rechner gar keine IPv6-Adresse hat)
- `ai_family`, `ai_socktype`, `ai_protocol` für `socket(2)` verwendbar
- `ai_addr`, `ai_addrlen` für `connect(2)` verwendbar



Agenda

- 1.1 Organisatorisches
- 1.2 IPC-Grundlagen
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden

```
struct addrinfo hints = {
    .ai_socktype = SOCK_STREAM, // Nur TCP-Sockets
    .ai_family = AF_UNSPEC, // Beliebige Adressfamilie
    .ai_flags = AI_ADDRCONFIG, // Nur lokal verfügbare Adresstypen
}; // C99: alle anderen Elemente der Struktur werden implizit genutzt

struct addrinfo *head;
getaddrinfo("xkcd.com", "80", &hints, &head);
// Fehlerbehandlung! Rueckgabewert mit gai_strerror(3) auswerten, um
// Fehlerbeschreibung zu erhalten

// Liste der Adressen durchtesten
int sock;
struct addrinfo *curr;
for (curr = head; curr != NULL; curr = curr->ai_next) {
    sock = socket(curr->ai_family, curr->ai_socktype, curr->ai_protocol);
    // Fehlerbehandlung!
    if (connect(sock, curr->ai_addr, curr->ai_addrlen) == 0)
        break;
    close(sock);
}
if (curr == NULL) {
    // Keine brauchbare Adresse gefunden :-(

}
// sock kann jetzt fuer die Kommunikation mit dem Server benutzt werden
```



Ein-/Ausgabemechanismen

- Nach dem Verbindungsaubau lässt sich ein Stream-Socket nach dem selben Schema benutzen wie eine geöffnete Datei
- Für Ein- und Ausgabe stehen verschiedene Funktionen zur Verfügung:
 - Ebene 2: POSIX-Systemaufrufe
 - arbeiten mit Dateideskriptoren (int)
 - Ebene 3: Bibliotheksfunktionen
 - greifen intern auf die Systemaufrufe zurück
 - wesentlich flexibler einsetzbar
 - arbeiten mit File-Pointern (FILE *)

| Ebene | Variante | Ein-/Ausgabedaten | Funktionen |
|-------|------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| 2 | blockorientiert | Puffer, Länge | read(), write() |
| 3 | blockorientiert zeichenorientiert zeilenorientiert formatiert | Array, Elementgröße, -anzahl Einzelbyte '\0'-terminierter String Formatstring, beliebige Variablen | fread(), fwrite() getc(), putc() fgets(), fputs() fscanf(), fprintf() |



Ein-/Ausgabemechanismen: FILE *

- Auf Grund ihrer Flexibilität eignen sich FILE * für String-basierte Ein- und Ausgabe wesentlich besser

- Erstellen eines FILE * für einen gegebenen Dateideskriptor:

```
FILE *fdopen(int fd, const char *mode);
```

- mode kann sein: "r", "w", "a", "r+", "w+", "a+"
- fd muss im entsprechenden Modus geöffnet worden sein

- Schließen des erzeugten FILE *:

```
int fclose(FILE *fp);
```

- Darunterliegender Dateideskriptor wird dabei geschlossen
- Nachfolgendes close(2) nicht notwendig

21-ClientSockets_handout



Agenda

- 1.1 Organisatorisches
- 1.2 IPC-Grundlagen
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden

21-ClientSockets_handout



Ein-/Ausgabemechanismen: Sockets und FILE *

- FILE * benutzen einen eigenen Pufferungsmechanismus

- Hat möglicherweise unerwünschtes Verhalten, wenn dieselbe FILE * für Ein- und Ausgabe verwendet wird

- Zwei separate FILE * für Empfangs- und Senderichtung erstellen

- Socket-Deskriptor vorher duplizieren, da nicht mehrere FILE * auf denselben Deskriptor verweisen sollten

```
int sock = ...;
```

```
// FILE * fuers Empfangen erstellen  
FILE *rx = fdopen(sock, "r");
```

```
// Duplikat des Socket-Deskriptors anlegen  
int sock2 = dup(sock);
```

```
// FILE * fuers Senden erstellen  
FILE *tx = fdopen(sock2, "w");
```

- Nach jeder geschriebenen Zeile mit fflush(3) das Leeren des Zwischenpuffers erzwingen

21-ClientSockets_handout



Aktive Mitarbeit erforderlich!

„Aufgabenstellung“

- Mit Hilfe des Programms Netcat (nc, ncat) eine E-Mail an die eigene Adresse senden

- Dialog mit dem Mail-Server live nachspielen
- Sonderbehandlung von Punkten am Zeilenanfang nachvollziehen

- Programm schreiben, welches den Flächeninhalt verschiedener geometrischer Formen berechnen kann

- Eingaben werden von STDIN eingelesen
- Eingabeformat: <zahl> <zahl> <form>
- Unterstützte Formen: dreieck, rechteck
- Ausgabeformat soll die eingegebene Werte und den Flächeninhalt beinhalten

21-ClientSockets_handout

