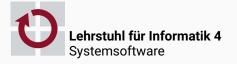
Übungen zu Systemprogrammierung 2

Ü1 – Interprozesskommunikation mit Sockets

Wintersemester 2025/26

Jürgen Kleinöder, Thomas Preisner, Tobias Häberlein, Ole Wiedemann

Lehrstuhl für Informatik 4 Friedrich-Alexander-Universität Erlangen-Nürnberg







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



1.1 Organisatorisches

- 1.2 IPC-Grundlager
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwender

Organisatorisches



Kontakt

- Forum: https://fsi.cs.fau.de/forum/18
- Tutoren: i4sp@cs.fau.de
- Organisatoren: i4sp-orga@cs.fau.de

Übungsbetrieb

Termine siehe:

https://sys.cs.fau.de/lehre/current/sp2/uebung#termine

GitLab-Repository

- vgl. SP1 Ü0 "Einführung"
- /proj/i4sp2/bin/mkrepo und git

Programmieraufgaben



- Durchgängige Verwendung von 64-Bit
- Programmiersprache ISO C11
- Betriebssystemstandard SUSv4 (= POSIX.1-2008)

Kompilieren & Linken

- Standard-Flags in SP:
 - -std=c11 -pedantic -Wall -Werror -D_XOPEN_SOURCE=700
 -fsanitize=undefined -fno-sanitize-recover -g
- In SP2 konsequente Benutzung von Makefiles

Programmieraufgaben



- Durchgängige Verwendung von 64-Bit
- Programmiersprache ISO C11
- Betriebssystemstandard SUSv4 (= POSIX.1-2008)

Kompilieren & Linken

- Standard-Flags in SP:
 - -std=c11 -pedantic -Wall -Werror -D_XOPEN_SOURCE=700 -fsanitize=undefined -fno-sanitize-recover -g
- In SP2 konsequente Benutzung von Makefiles

Robuste Programme

Ausführliches Testen und Fehlerbehandlung nicht vergessen!

Miniklausur-Termin



- Termin für die Miniklausur: 14.11.2025 in H20
- Zur Abschätzung der Teilnehmerzahl Anmeldung notwendig
 - Möglich bis zum 11.11.2025
 - Veranstaltungstermin in WAFFEL
 - ⇒ https://waffel.cs.fau.de/signup?course=499
 - Bitte meldet euch wieder ab, falls ihr nicht teilnehmen wollt
- Erlaubte Hilfsmittel: ein beidseitig handbeschriebener **DIN-A5**-Zettel



1.1 Organisatorisches

1.2 IPC-Grundlagen

- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwenden

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

Der Server ist (in der Regel) als normaler Benutzerprozess realisiert.

Client

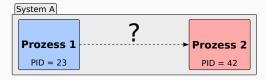
- 1. Schickt eine **Anforderung an einen Server**
- 2. Wartet auf eine Antwort

Server

- Akzeptiert Anforderungen, die von außen kommen
- 2. **Führt** einen angebotenen **Dienst aus**
- Schickt das Ergebnis zurück zum Sender der Anforderung

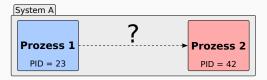
? Wie findet ein Client den gewünschten Dienstanbieter?

- ? Wie findet ein Client den gewünschten Dienstanbieter?
 - Intuitiv: über dessen Prozess-ID

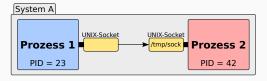


• Problem: Prozesse werden dynamisch erzeugt/beendet; PID ändert sich

- ? 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

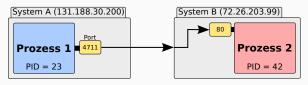


Prozess 2 ist über den Dienstnamen (hier: einen Dateinamen) erreichbar

? Wie findet ein Client nun den gewünschten Dienstanbieter?

? 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)



IPv4

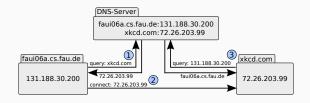
- 32-Bit-Adressraum (\approx 4 Milliarden Adressen)
- Notation: 4 mit . getrennte Byte-Werte in Dezimaldarstellung
 - z.B. 131.188.30.200
- Nicht zukunftsfähig wegen des zu kleinen Adressraums

Nachfolgeprotokoll IPv6

- 128-Bit-Adressraum (\approx 3,4 · 10³⁸ Adressen)
- 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: 0:0:0:0:0:0:0:1 = ::1

- IP-Adressen sind nicht leicht zu merken
- ... und ändern sich, wenn man einen Rechner in ein anderes Rechenzentrum umzieht

- 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



- 1. Forward lookup: Rechnername \rightarrow IP-Adresse
- 2. Kommunikationsaufbau
- 3. Reverse lookup (im Beispiel optional): IP-Adresse ightarrow Rechnername

- 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
 - Liste der definierten Ports und Protokolle:
 - https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
 - /etc/services(services(5))

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 -duplizierung
- Datenpakete können eventuell in falscher Reihenfolge ankommen
- Grenzen von Datenpaketen bleiben erhalten
- Implementierung: User Datagram Protocol (UDP)

- Beim Austausch von binären Datenwörtern zwischen verschiedenen Rechnern muss Einigkeit über die verwendete Byteorder herrschen!
- Beispiel: little endian (z.B. x86) und big endian (z.B. SPARC, MIPS)

Wert	0x17c3a15c				"xkcd"					
big endian	17	с3	a1	5c		Х	k	С	d	\0
little endian	5C	a1	с3	17		Х	k	С	d	\0
	niedrige					hohe				
	Adressen					Adressen				

- Beim Austausch von binären Datenwörtern zwischen verschiedenen Rechnern muss Einigkeit über die verwendete Byteorder herrschen!
- Beispiel: little endian (z.B. x86) und big endian (z.B. SPARC, MIPS)



- **Definierter Standard:** Netzwerk-Byteorder = big endian
 - Konvertierung von Integer-Werten vor dem Senden und nach dem Empfangen nötig
- Keine Endianess falls byteweise interpretiert (ASCII, UTF-8)



- 1.1 Organisatorisches
- 1.2 IPC-Grundlager
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwender

- 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 Speicher, Reihenfolge und Zustellung vom Betriebssystem sichergestellt
 - Über Rechnergrenzen hinweg: z. B. TCP/UDP-Socket
 - → Adressierung über IP-Adresse + Port, nachrichtenbasierte Kommunikation, Reihenfolge und Zustellung ggf. durch Protokollmechanismen sichergestellt (z.B. TCP)
- Beim Verbindungsaufbau sind die entsprechenden Parameter zu wählen
- Anschließende Verwendung unabhängig von Verbindungsart
 - ... egal, ob der Kommunikationspartner ein Prozess auf dem selben Rechner oder am anderen Ende der Welt ist

Sockets werden mit dem Systemaufruf socket(3p) angelegt:

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

- domain, z. B.:
 - AF_UNIX: UNIX-Dom\u00e4ne (\infty lokal)
 - AF_INET: IPv4-Domäne
 - AF_INET6: IPv6-Domäne (kompatibel zu IPv4, wenn vom OS unterstützt)
- type innerhalb der gewählten Domäne:
 - SOCK_STREAM: Stream-Socket (→ TCP)
 - SOCK_DGRAM: Datagramm-Socket (~ UDP)
- protocol:
 - 0: Standard-Protokoll für gewählte Kombination
 (z. B. TCP/IP bei AF_INET6 + SOCK_STREAM)
- Ergebnis ist ein numerischer Socket-Deskriptor
 - Entspricht einem Datei-Deskriptor und unterstützt (bei Stream-Sockets)
 die selben Operationen: read(3p), write(3p), close(3p), ...

connect(3p) meldet Verbindungswunsch an Server:

- 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 (Grund dafür in der nächsten Übung)
- 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

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

- 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(3p)) 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(3p)

```
struct addrinfo {
                 ai_flags; // Flags zur Auswahl (hints)
 int
                 ai_family; // z. B. AF_INET6
 int
 int
                 ai socktype; // z. B. SOCK STREAM
 int
                 ai_protocol; // Protokollnummer
 socklen t
          ai_addrlen; // Größe von ai_addr
 struct sockaddr *ai_addr; // Adresse für connect()
                *ai_canonname; // Offizieller Hostname (FQDN)
 char
 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(3p)
 verwendbar
- ai_addr, ai_addrlen für connect(3p) verwendbar

```
struct addrinfo hints = {
  .ai_socktype = SOCK_STREAM, // Nur TCP-Sockets
  .ai_family = AF_UNSPEC, // Beliebige Adressfamilie
  .ai flags = AI ADDRCONFIG, // Nur lokal verfügbare Adresstypen
}; // C: alle anderen Elemente der Struktur werden implizit genullt
struct addrinfo *head;
getaddrinfo("xkcd.com", "80", &hints, &head);
// Fehlerbehandlung! Rückgabewert 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 ietzt für die Kommunikation mit dem Server benutzt werden
```



- 1.1 Organisatorisches
- 1.2 IPC-Grundlager
- 1.3 Betriebssystemschnittstelle zur IPC
- 1.4 Ein-/Ausgabemechanismen
- 1.5 Gelerntes anwender

Ein-/Ausgabemechanismen



- Nach dem Verbindungsaufbau 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	Array, Elementgröße, -anzahl	fread(),fwrite()
		zeichenorientiert	Einzelbyte	getc(), putc()
		zeilenorientiert	'\0'-terminierter String	fgets(), fputs()
		formatiert	Formatstring, beliebige Variablen	<pre>fscanf(), fprintf()</pre>

Ein-/Ausgabemechanismen: FILE*



- Auf Grund ihrer Flexibilität eignen sich FILE* für String-basierte Einund 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+"
 - "r": lesen, "w": abschneiden(truncate)+schreiben, "a": anhängen
 - "+": Zusatzbedeutung, s. fopen(3p)
 - fd muss mit kompatiblem Modus geöffnet worden sein
- Schließen des erzeugten FILE*: int fclose(FILE *fp);
 - Darunterliegender Dateideskriptor wird dabei geschlossen
 - Kein zusätzlicher Aufruf von close(3p)

Ein-/Ausgabemechanismen: Sockets und FILE*



- FILE* benutzen einen eigenen Pufferungsmechanismus
 - Hat möglicherweise unerwünschtes Verhalten, wenn derselbe 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 dürfen

```
int sock = ...;
// FILE * fürs Empfangen erstellen
FILE *rx = fdopen(sock, "r");
// Duplikat des Socket-Deskriptors anlegen
int sock_copy = dup(sock);
// FILE * fürs Senden erstellen
FILE *tx = fdopen(sock_copy, "w");
```

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



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

Aktive Mitarbeit erforderlich!



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

Aktive Mitarbeit erforderlich!



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 Eingabe zeichenweise nach Groß-/Kleinschreibung konvertiert ausgibt

- Eingaben werden von **stdin** eingelesen
- Eingabeformat: 'u'/'l' <beliebig viele zeichen>
- Ausgabe: alle Zeichen der Eingabe als Groß('u')- bzw.
 Kleinbuchstaben('l')