G Interprozeßkommunikation

G.1 Überblick

- ursprüngliche UNIX IPC-Mechanismen (UNIX V7)
  - ◆ Signale
  - ◆ wait / exit-Wert
  - ◆ Pipes
  - ◆ Dateien
- UNIX System V - Erweiterungen
  - ◆ Named Pipes
  - ◆ Shared Memory
  - ◆ Messages
  - ◆ Semaphore
- Berkeley-UNIX Erweiterungen
  - ◆ Sockets
- MACH-IPC
  - ◆ Ports & Messages

G.2 Sockets

- Endpunkte einer Kommunikationsverbindung
- Arbeitsweise: FIFO, bidirektional
- Attribute:
  - ➤ **Name** (durch *Binding*)
  - ➤ **Communication Domain**
  - ➤ **Typ**
  - ➤ **Protokoll**

1 Communication Domain und Protokoll

- **Communication Domain** legt die **Protokoll-Familie**, in der die Kommunikation stattfindet, fest
- durch die Protokoll-Familie wird gleichzeitig auch die Adressierungsstruktur **(Adreß-Familie)** festgelegt
(war unabhängig geplant, wurde aber nie getrennt)
- das **Protokoll**-Attribut wählt das Protokoll innerhalb der Familie aus
- ursprünglich (bis BSD 4.3) existierten nur zwei Communication Domains
  - ➤ UNIX-Domain (AF_UNIX)
  - ➤ Internet-Domain (AF_INET)
- nur AF_INET ist generell vorhanden
daneben derzeit ca. 25 AF definiert
(ISO-Protokolle, DECnet, SNA, Appletalk, …)

1 Communication Domain und Protokoll (2)

- **UNIX Domain**
  - ◆ für lokale Prozeßkommunikation
  - ◆ bidirektionale Pipes
  - ◆ Namen aus dem UNIX Dateisystem
    - – Eintrag im Dateisystem (Dateityp Socket)
    - – verwendbar wie Named Pipes
    - – Anlegen und Öffnen nicht mit *mknod( )* und *open( )*, sondern über die normale Socket-Schnittstelle
    - – Schließen und Löschen normal über *close( )* und *unlink( )*
- **Internet Domain**
  - ◆ für Kommunikation über Rechnernetz
  - ◆ Protokolle: **TCP/IP** oder **UDP/IP**
  - ◆ Namen: **IP-Adressen** und **Port-Nummern**

2 Socket Typen

- **Stream-Sockets**
  - unterstützen bidirektionalen, zuverlässigen Datenfluß
  - gesicherte Kommunikation (gegen Verlust und Duplizierung von Daten)
  - die Ordnung der gesendeten Daten bleibt erhalten
  - Vergleichbar mit einer *pipe* - allerdings bidirektional (UNIX-Domain- und Internet-Domain-Sockets mit TCP/IP)
- **Datagramm-Sockets**
  - unterstützen bidirektionalen Datenfluß
  - Datentransfer unsicher (Verlust und Duplizierung möglich)
  - die Reihenfolge der ankommenden Datenpakete stimmt nicht sicher mit der der abgehenden Datenpakete überein
  - Grenzen von Datenpaketen bleiben im Gegensatz zu **Stream-Socket** - Verbindungen erhalten (Internet-Domain Sockets mit UDP/IP)

2 Socket Typen (2)

- **Raw-Sockets**
  - ermöglichen Zugriff auf das darunterliegende Kommunikationsprotokoll (z. B. IP); sind nicht für den allgemeinen Gebrauch gedacht.
- **Kern-Sockets**
  - *Bi-direktionale, asynchrone* Kommunikation mit Kern-Subsystemem (z.B. Routing-Tabellen-Verwaltung)
  - *Broadcast* von Kern an User-Level-Prozesse möglich
- **SEQPACKET-Sockets**
  - gesicherter Paket-Strom (Paketgrenzen bleiben erhalten, aber sichere Übertragung, wie bei Stream-Sockets)

★ nur Stream- und Datagramm-Sockets sind allgemein implementiert!

3 Client-Server Modell

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

- Server
  - **akzeptieren Anforderungen**, die von der Kommunikationsschnittstelle kommen
  - **führen** ihren angebotenen **Dienst aus**
  - **schicken** das **Ergebnis zurück** zum Sender der Anforderung
  - *Server* sind normalerweise als normale Benutzerprozesse realisiert
- Client
  - ein Programm wird ein ***Client***, sobald es
    - eine **Anforderung an einen Server** schickt und
    - auf eine Antwort wartet

4 Generieren eines Sockets

- Sockets werden mit dem Systemaufruf *socket(2)* angelegt

```
s = socket(Domain, Typ, Protokoll);
```

5 Namensgebung

- Sockets werden ohne Namen generiert
- durch den Systemaufruf *bind(2)* wird einem Socket ein Name zugeordnet

```
bind (s, Name, Namenslänge);
```

6 Verbindungsanforderungen annehmen

- ein Socket wird durch ***listen(2)*** auf Verbindungsanforderungen vorbereitet, die in einer Warteschlange zwischengespeichert werden
- die angeforderten Verbindungen werden einzeln aufgebaut/ zurückgewiesen

```
listen (s, queuelength);
```

7 Verbindungsaufbau und Kommunikation (Stream-Sockets)

- Über einen Socket ist eine Kommunikation zwischen Prozessen möglich:
  - ◆ **UNIX-Domain**-Sockets: Kommunikation lokal auf einem Rechner
  - ◆ **Internet**-Sockets: Kommunikation über Netzwerk(e)
- Der Kommunikationsaufbau ist asymmetrisch - ein Prozeß agiert als ***Client***, der andere als ***Server***.

7 Verbindungsaufbau und Kommunikation (Stream-Sockets) (2)

- Ablauf:

Client

```
...
connect(s, &server,
        sizeof(server))

...
write(s, buf, 100);
...
read(s, buf1, 20);
```

Server

```
...
listen(s, 5);
s_new = accept(s, from,
          sizeof(from));

...
read(s_new, buf, 100);
...
write(s_new, bufx, 20);
```


8 Stream-Sockets — Systemschnittstelle

- Verbindungsaufbau

listen(2) teilt dem System mit, daß man bereit ist, Verbindungen auf dem Socket entgegenzunehmen

accept(2) nimmt eine Verbindung an und generiert einen neuen Socket (***s_new***), der für die Kommunikation verwendet werden kann - steht kein Verbindungwunsch an, blockiert *accept*

connect(2) baut die Verbindung vom ***client*** zum ***server*** auf

- Kommunikation

read(2), write(2) verhalten sich wie gewohnt

send(2), recv(2) wie *write( )* bzw. *read(2)* + zusätzliche Funktionalität (Daten ansehen ohne zu lesen, *out-of-band-data, ...).*

- Verbindungsabbau

close(2) Socket schließen

shutdown(2) Elimieren von bereitliegenden Daten vor einem *close( )*

9 Verbindungslose Sockets

- Für Kommunikation über Datagramm-Sockets kein Verbindungsaufbau notwendig
- Systemaufrufe

  sendto(2) Datagramm senden

  recvfrom(2) Datagramm empfangen
- **Besonderheit: *Broadcasts*** über Datagramm-Sockets (Internet Domain)

10 weitere Systemaufrufe der Socketschnittstelle

getpeername(2)
Namen der mit dem Socket verbundenen Gegenstelle abfragen

getsockname(2)
Namen eines Sockets abfragen

getsockopt(2), setsockopt(2)
Parametrierung eines Sockets abfragen / setzen

- weitere Manual-Seiten (Solaris): ***socket(5), in(5), tcp(7p), ip(7p), udp(7p)***