Internetanwendungstechnik (Übung)

Sockets

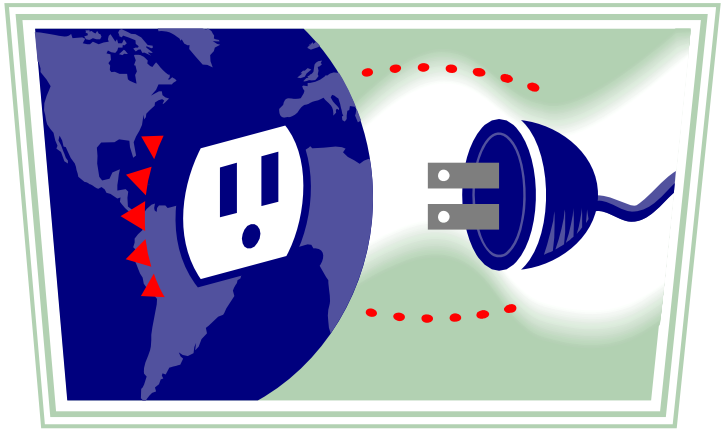
Stefan Bissell, Gero Mühl

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Kommunikations- und Betriebssysteme (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Programmierschnittstelle für TCP/IP?

- Protokolle (z.B. TCP, UDP) definieren
  - Aufbau der Nachrichten
  - Ablauf und Regeln des Nachrichtenaustausches
- Protokolle (z.B. TCP, UDP) definieren nicht
  - Schnittstelle zur Anwendung (API)

TCP
UDP
IP



Anwendung

Schnittstelle (API)

Sockets als Programmierschnittstelle

- Socket (engl. *Steckdose*) als zentrale Abstraktion
- Berkley Sockets / BSD Socket API
  - Ursprung in 4.2BSD Unix Operating System (1983)
  - De facto Standard Netzwerk API
  - Teil des Portable Operating System Interface (POSIX)
    Standards → Unterstützung durch fast aller Betriebssysteme
- Menge von C Funktionen und Datentypen
  - Senden/Empfangen von Daten
  - Verbindungsmanagement
  - Tiefgreifendere Einstellungen (Puffergrößen, etc.)
- Anbindungen für viele andere Sprachen (z.B. auch Java)

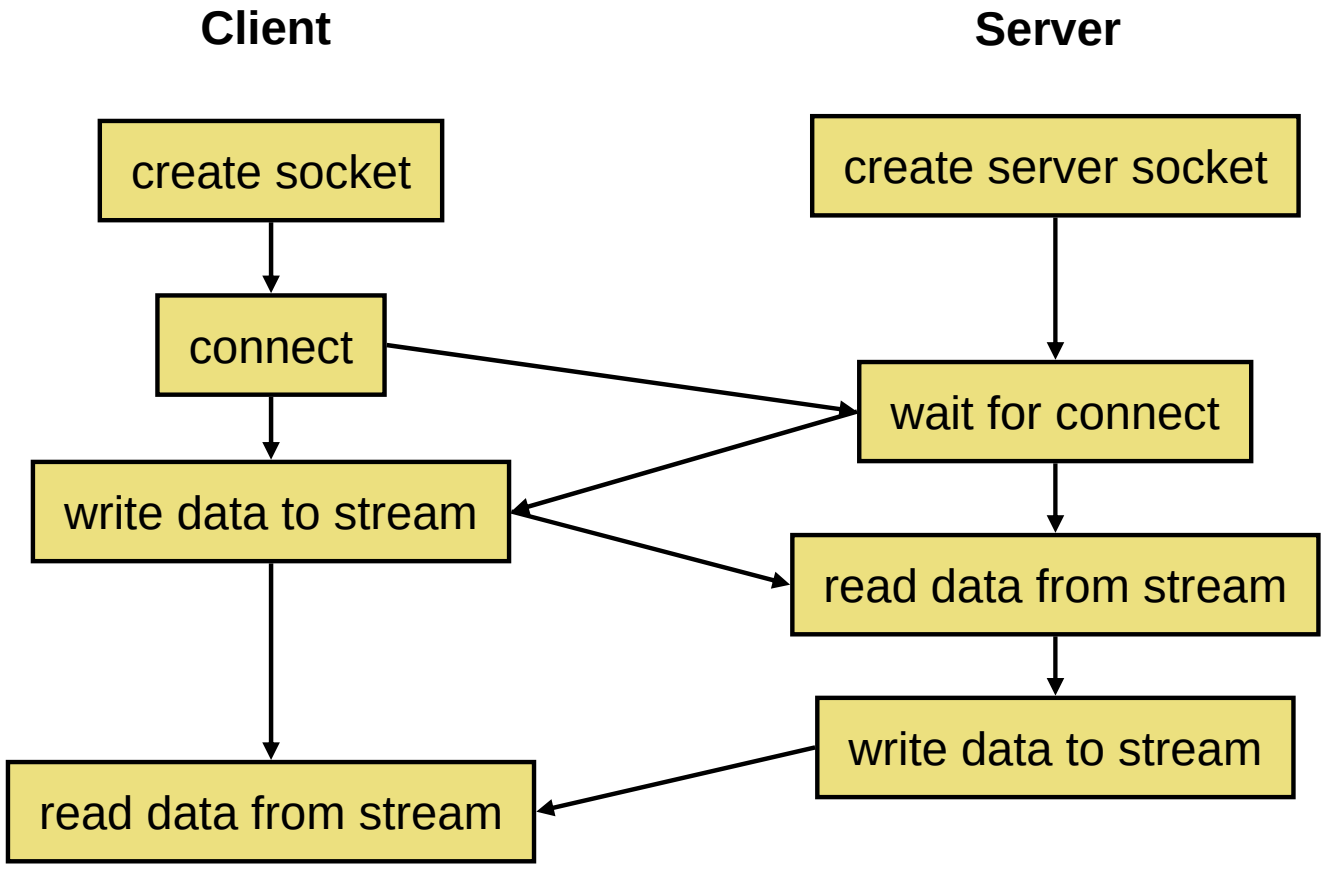
Socket-Programmierung mit Java

Socket Programmierung mit Java

- Paket *java.net* enthält unter anderem folgende Klassen
  - *Socket* TCP Client Socket
  - *ServerSocket* TCP Server Socket

  - *DatagramSocket* UDP Socket
  - *MulticastSocket* UDP Multicast Socket
  - *DatagramPacket* UDP Datagram

Struktur einer TCP-basierten Anwendung



Echo Client mit TCP

```
public class EchoClient {
   public static void main(String[] args) throws IOException {
      Socket echoSocket = new Socket(EchoServer.HOST_NAME,
            EchoServer.PORT_NUM);
      PrintWriter out = new PrintWriter(
            echoSocket.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
           new InputStreamReader(echoSocket.getInputStream()));
      BufferedReader stdIn = new BufferedReader(
            new InputStreamReader(System.in));
      String userInput;
      while ((userInput = stdIn.readLine()) != null) {
         out.println(userInput);
         System.out.println("echo: " + in.readLine());
      }
      out.close(); in.close(); stdIn.close(); echoSocket.close();
   }
}
```

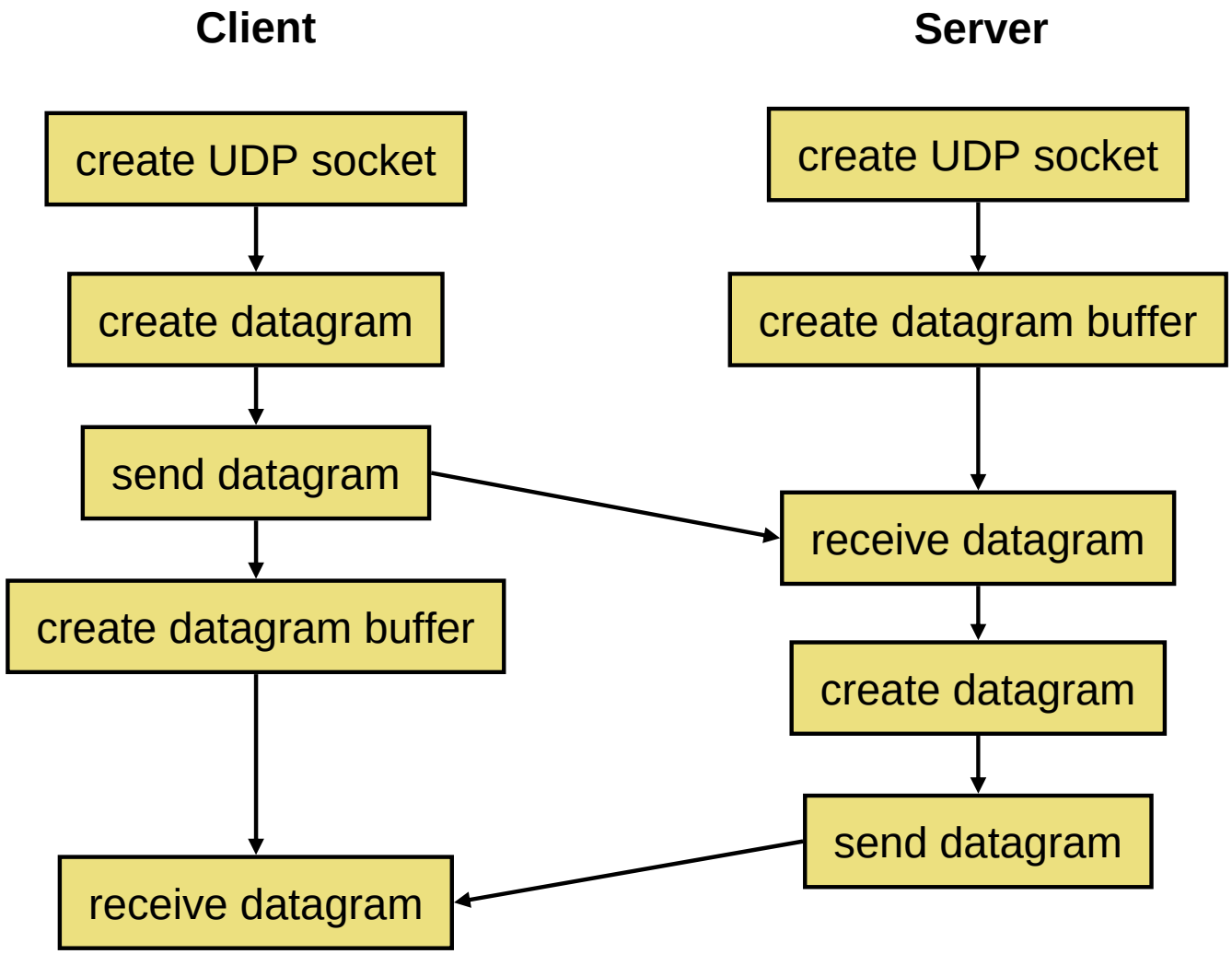
Echo Server mit TCP

```
public class EchoServer {
   public static final int PORT_NUM = 6789;
   public static final String HOST_NAME = "localhost";

   public static void main(String args[]) throws IOException {
      ServerSocket serverSocket = new ServerSocket(PORT_NUM);
      Socket socket = null;
      EchoServerThread echoServerThread = null;
      while (true) {
         socket = serverSocket.accept();
         echoServerThread = new EchoServerThread(socket);
      }
   }
}
```

Echo Server Thread mit TCP

```
public class EchoServerThread extends Thread {
   private Socket socket;
   public EchoServerThread(Socket socket) {
      this.socket = socket; this.start();
   }

   public void run() {
      try {
         BufferedReader br = new BufferedReader(
               new InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(new OutputStreamWriter(
               socket.getOutputStream()), true);
         String line = null;
         while ((line = br.readLine()) != null) {
            out.println(line);
         }
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

Struktur einer UDP-basierten Anwendung



Date Client mit UDP

```
public class DateClient {
   public static void main(String[] args) throws IOException {
      DatagramSocket socket = new DatagramSocket();
      byte[] buf = new byte[256];
      InetAddress address =
            InetAddress.getByName(DateServer.HOST_NAME);
      DatagramPacket packet = new DatagramPacket(buf, buf.length,
            address, 4445);
      socket.send(packet);
      packet = new DatagramPacket(buf, buf.length);
      socket.receive(packet);
      String received = new String(packet.getData());
      System.out.println("Date: " + received);
      socket.close();
   }
}
```

Date Server mit UDP

```
public class DateServer {
   public static final String HOST_NAME = "localhost";
   public static void main(String[] args) throws IOException {
      while (true) {
         byte[] buf = new byte[256];
         DatagramSocket socket = new DatagramSocket(4445);
         DatagramPacket packet = new DatagramPacket(buf, buf.length);
         socket.receive(packet);
         String dString = new Date().toString();
         buf = dString.getBytes();
         InetAddress address = packet.getAddress();
         int port = packet.getPort();
         packet = new DatagramPacket(buf, buf.length, address, port);
         socket.send(packet);
         socket.close();
      }
   }
}
```

Socket-Programmierung mit C

Protokolle und Socket-Typen

Verschiedene Protokollfamilien

- PF_INET
  - IPv4 Internet Protokolle
- PF_INET6
  - IPv6 Internet Protokolle
- PF_IPX
  - Novells Protokolle
- PF_UNIX, PF_LOCAL
  - Lokale Kommunikation
- PF_APPLETALK
  - Appletalk

Verschiedene Socket-Typen

- SOCK_STREAM
  - Verbindungsorientiert
  - Bytestrom
- SOCK_DGRAM
  - Verbindungslos
  - Nachrichten
- SOCK_SEQPACKET
  - Verbindungsorientiert
  - Nachrichten
- SOCK_RAW
  - Zugriff auf tiefere Schichten
  - Privilegierter Benutzer

Nicht alle Socket-Typen für alle Protokollfamilien verfügbar!

Socket-Operationen

		Client/Server
> Operationen für TCP/UDP		
> socket	Erzeugt einen neuen Socket	C/S
> bind	Bindet Socket an Portnummer	C/S
> close	Schließt einen Socket	C/S
> Operationen für TCP		
> listen	Bereitschaft zum Verbindungsaufbau	S
> accept	Annehmen einer neuen Verbindung	S
> connect	Verbindung aufbauen	C
> send	Daten senden	C/S
> recv	Daten empfangen	C/S
> shutdown	Verbindungsabbau	C/S
> Operationen für UDP		
> sendto	Daten an entfernten Endpunkt senden	
> recvfrom	Daten von entferntem Endpunkt empfangen	

Erzeugen eines Sockets

- int **socket** (int *domain*, int *type*, int *protocol*);
  - Erzeugt neuen Socket und liefert zugehörigen Socket-Deskriptor
  - **domain**: Protokollfamilie (z.B. PF_INET, PF_INET4)
  - **type**: Socket-Typ (z.B. SOCK_STREAM, SOCK_DGRAM)
  - **protocol**: Protokollnummer (z.B. TCP, UDP)
  - **return**: Socket-Deskriptor bei Erfolg, sonst -1

- Beispiel TCP
  - int sd = **socket** (PF_INET, SOCK_STREAM, 0);
  - Protokoll unspezifiziert → Wahl des Default-Protokolls für gegebene Kombination aus Protokollfamilie und Socket-Typ

Socket-Adressen

- Bestimmen Adresse des Kommunikationsendpunktes
  - struct **sockaddr** {

```
    unsigned short      sa_family;     // Adressfamilie (AF_xxx)
    char                sa_data[14];   // 14 Bytes Protokolladresse
};
```

- Angepasste Typen für einzelne Protokollfamilien (z.B. IPv4)
  - struct s**ockaddr_in** {

```
    short int           sin_family;    // immer AF_INET
    unsigned short int  sin_port;      // Port-Nummer
    struct in_addr      sin_addr;      // IP-Adresse
    unsigned char       sin_zero[8];   // Gleiche Größe wie sockaddr
};
```

Verbindungsaufbau

- Passives Verbinden (Server)
  - Warten auf Verbindungsaufbau durch Client
  - Socket dient nur der Annahme der Verbindungswünsche
  - Erzeugung eines neuen Sockets für jede angenommene Client-Verbindung
- Aktives Verbinden (Client)
  - Explizite Verbindung zu bekannten (meist entfernten) Kommunikationsendpunkt
  - Lokale Adresse (meist) beliebig

Binden von Sockets (Server)

- int **bind** (int *sd*, struct sockaddr **my_addr*, socklen_t *addrlen*);
  - Weist dem Socket einen Kommunikationsendpunkt (lokale Adresse) zu
  - **sd**: Socket-Deskriptor
  - **my_addr**: lokale Socket-Adresse (Kommunikationsendpunkt)
  - **addrlen**: Größe von *addr* in Bytes
  - **return**: 0 bei Erfolg, sonst -1

Warten auf Verbindungswünsche (Server)

- int **listen**(int *sd*, int *backlog*);
  - Wartet auf Verbindungswünsche am Socket
  - **sd:** Socket-Deskriptor
  - **backlog**: max. Anzahl noch nicht angenommener Verbindungswünsche (max. Länge der Warteschlange)
  - **return**: 0 bei Erfolg, sonst -1

Annehmen von Verbindungen (Server)

- int **accept**(int *sd*, struct sockaddr **addr*, socklen_t **addrlen*);
  - Annehmen von Verbindungswünschen an Socket
  - Erzeugen eines neuen Sockets für angenommene Verbindung
  - **sd**: Socket-Deskriptor (Socket im Listen-Mode)
  - **addr**: Leere Socket-Addressstruktur für Adresse des Partners
  - **addrlen**: Größe von *addr* in Bytes
  - **return**: bei Erfolg den Socket-Deskriptor der angenommenen Verbindung, sonst -1

Verbindungsaufbau (Client)

- int **connect**(int *sd*, const struct sockaddr **addr*, socklen_t *addrlen*);
  - Aufbauen einer Verbindung zu einem (entfernten) Socket
  - **socket**: Socket-Deskriptor
  - **addr**: Socket-Adresse des Partners zu dem die Verbindung aufgebaut werden soll
  - **addrlen**: Größe von *addr* in Bytes

Ablauf des Verbindungsaufbaus

Server

- sd = socket(…);
- addr.sin_family = AF_INET;
  addr.sin_addr.s_addr =
  INADDR_ANY;
  addr.sin_port = htons(80);
  memset(&(addr.sin_zero),
  '\0', 8);
- bind(sd, addr, sizeof(addr));
- listen(sd, 10);
- cd = accept(sd, caddr,
  &addrlen)

Client

- sd = socket(…);
- addr.sin_family = AF_INET;
  addr.sin_addr.s_addr =
  inet_addr("10.0.0.1");
  addr.sin_port = htons(80);
  memset(&(addr.sin_zero),
  '\0', 8);
- connect(sd, &addr,
  sizeof(addr));

Senden von Daten

- ssize_t **send**(int *sd*, const void **buf*, size_t *len*, int *flags*);
  - Senden einer Nachricht oder Menge von Bytes
  - **sd**: Socket-Deskriptor
  - **buf**: Sendepuffer mit zu übertragenden Daten
  - **len**: Anzahl der zu sendenden Bytes
  - **flags**: z.B. MSG_DONTWAIT, MSG_MORE
  - **return**: Anzahl gesendeter Bytes bei Erfolg, sonst -1
- ssize_t **sendto**(…, const struct sockaddr **to*, socklen_t *tolen*);
  - Sendet Nachricht an Zielendpunkt (z.B. bei UDP)
  - Weitere Parameter analog zu *send*
  - **to**: Socket-Adresse des Ziels
  - **tolen**: Größe von *to* in Bytes
- Alternativ: **write**(…);
  - Nutzung des Socket-Deskriptors als File-Deskriptor

Empfangen von Daten

- ssize_t **recv**(int *sd,* void **buf*, size_t *len*, int *flags*);
  - Empfängt Nachricht oder Menge von Bytes
  - **sd**: Socket-Deskriptor
  - **buf**: Puffer zum Ablegen der empfangenen Daten
  - **len**: Anzahl zu empfangener Bytes
  - **flags**: z.B. MSG_DONTWAIT, MSG_PEEK
  - **return**: Anzahl der tatsächlich gelesenen Daten, sonst -1
- ssize_t **recvfrom**(…, struct sockaddr *from, socklen_t *fromlen);
  - Empfängt Daten von angegebenen Endpunkt
  - Weitere Parameter analog zu *recv*
  - **from**: Socket-Adresse des Senders
  - **fromlen**: Größe von *from* in Bytes
- Alternativ: **read**(…)
  - Nutzung des Socket-Deskriptors als File-Deskriptor

Verbindungsabbau

- int **shutdown**(int *sd*, int *how*);
  - (Einseitiges) Schließen einer bidirektionalen Verbindung
  - **sd**: Socket-Deskriptor
  - **how**: Bestimmt den zu schließenden Teil der Verbindung
    - SHUT_RD: Unterbindet weitere Empfangsoperationen
    - SHUT_WR: Unterbindet weitere Sendeoperationen
    - SHUT_RDWR: Schließt gesamte Verbindung
  - **return**: 0 bei Erfolg, sonst -1

Freigabe

- int **close**(int *sd*);
  - Freigabe belegter Betriebsmittel
  - **sd**: Socket-Deskriptor (File-Deskriptor)
  - **return**: 0 bei Erfolg, sonst -1

Nützliches

- **gethostbyname**(…);
  - Namensauflösung des Hostnamens via DNS
  - Liefert IP-Adresse
- **getsockname**(…);
  - Liefert lokale Socket-Adresse (Kommunikationsendpunkt)
- **getpeername**(…);
  - Liefert entfernte Socket-Adresse (Kommunikationsendpunkt)
- **getsockopt**(…); / **setsockopt**(…);
  - Lesen/Setzen verschiedener Optionen
  - Optionen meist protokollspezifisch

Nützliches

- **poll**(…);
  - Abfragen des Zustandes (I/O-Bereitschaft) eines Sockets
- **select**(…);
  - Auswahl des nächsten bereiten Sockets
- Header Files
  - sys/types.h
  - sys/socket.h
  - netinet/in.h
  - arpa/inet.h
  - netdb.h

Literatur

- Socket-Programmierung mit Java
  - Java Tutorial, Trail: Coustom Networking http://java.sun.com/docs/books/tutorial/networking/index.html
- C-Programmierung
  - M. Werner. C-Crashcourse. (Unix-Praktikum im WS 2006/07). http://kbs.cs.tu-berlin.de/~mwerner/lect/unix/1-ccc.pdf
  - B. Hall. Beej's Guide to C Programming. http://beej.us/guide/bgc/
- Netzwerk-Programmierung
  - Unix Manual Pages.
  - B. Hall. **Beej's Guide to Network Programming Using Internet Sockets.** http://beej.us/guide/bgnet/
  - MSDN Library. Windows Sockets 2. http://msdn2.microsoft.com/en-us/library//ms740673.aspx

Fragen?