# 4A Kommunikation: Lösungen

## 4A.1 Wissens- und Verständnisfragen

*1.a.) die möglichen zeitlichen Abläufe von Kommunikationsvorgängen, die Struktur der übertragenen Daten*

*1.b.) Prozesskommunikation*

*1.c.) lokale Kommunikation, Kommunikation über das Internet*

*1.d.) Shared Memory, Message Queues*

*1.e.) dass sich der Programmierer bei Shared Memory selbst um die Synchronisation kümmern muss und bei Message Queues nicht*

*1.f.) mkfifo()*

*1.g.) Sockets*

*1.h.) ein Prozess connect() und der andere accept() ausgeführt hat*

*1.i.) Internet Domain, UNIX Domain*

*2.a.) Port*

*2.b.) Multicast*

*2.c.) Protokoll*

*2.d.) TCP, UDP*

*2.e.) msgget()*

*2.f.) Sockets*

*3.a.) Falsch. Beim Polling fragt ein Prozess wiederholt aktiv nach, ob sein Patrtner bereit ist.*

*3.b.) Wahr. Im Gegensatz zu IP und UDP ist TCP zuverlässig und garantiert eine geordnete Datenübertragung.*

*3.c.) Falsch. Es besteht weiter und muss gegebenenfalls per Hand durch ipcrm gelöscht werden.*

*3.d.) Falsch. Die Kommunikation über Stream Sockets ist verbindungsorientiert, was TCP entspricht.*

*3.e.) Falsch. HTTP nutzt TCP, und TCP nutzt wiederum IP.*

*3.f.) Wahr. Java unterscheidet die beiden Klassen ServerSocket und (für Client Sockets) Socket.*

*4.a.) schnittstellenbasiert, denn es ist keine Kommunikationsart*

*4.b.) Rendez-Vous, denn es ist kein bestimmter Kommunikationsmechanismus*

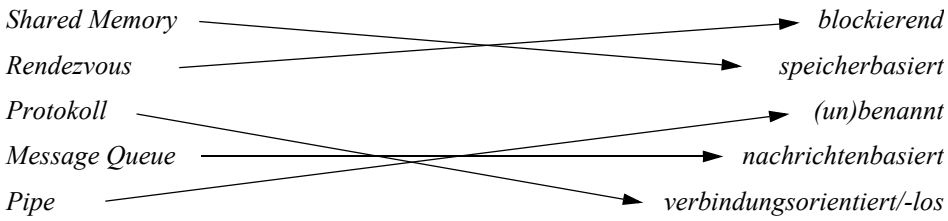
*4.c.) Semaphor, denn es ist kein Kommunikations-, sondern ein Synchronisationsmechanismus*

*4.d.) speicherbasiert, denn es ist keine Eigenschaft eines Protokolls*

*4.e.) API, denn es ist kein Protokoll*

*4.f.) wait(), denn die Funktion bezieht sich sich nicht auf Sockets.*

*5.)Begriffe und Eigenschaften:*




*6.a.) „Blocking send“, denn der Anrufer muss warten, bis sich der Angerufene meldet. Bei der Benutzung eines Anrufbeantworters ist das nicht der Fall,also „non-blocking send“.*

*6.b.) Eine Mailbox (ein „Briefkasten“), aus dem nur genau ein Prozess Nachrichten lesen kann. Message Queues, Sockets.*

*6.c.) Protokoll = Menge von Regeln für den geordneten Ablauf einer Kommunikation. Peers = Funktionseinheiten gleicher Ebene, die mit Hilfe eines Protokolls kommunizieren.*

*6.d.) Sie identifizieren unterschiedliche Anwendungen/Dienste auf demselben Computer (also unter derselben IP-Adresse).*

*6.e.) Server = Dienstanbieter = System, das nach außen Dienste anbietet. Client = Dienstnutzer = System, das diese Dienste in Anspruch nimmt.*

*6.f.) Nein. Semaphore sind ein Synchronisations-, aber kein allgemeiner Kommunikationsmechanismus.*

*6.g.) shmctl(...,IPC_RMID,...).*

*6.h.) Pipes, Message Queues.*

*6.i.) Der Empfänger kann selektiv eine bestimmte Art von Messages lesen, indem er bei seiner Lese-Operation msgrcv() den gewünschten Typ der zu lesenden Message angibt.*

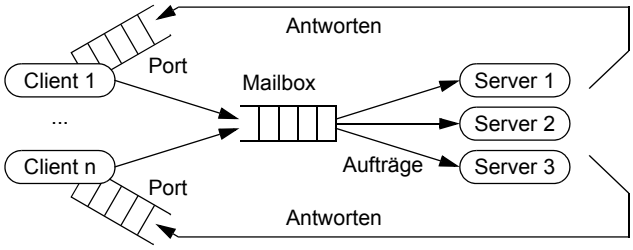
*6.j.) Sockets.*

*6.k.) Sockets.*

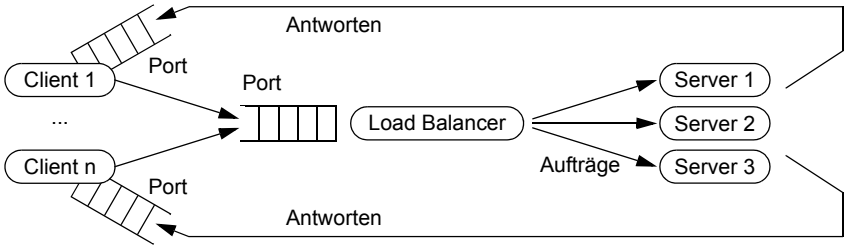
## 4A.2 Sprachunabhängige Anwendungsaufgaben

*1.)Besser in eine Mailbox, auf die dann alle Server zugreifen können. Ein Port ist nur genau einem Server zugeordnet.*

*2.)Client-Server-System mit Mailbox und Ports*

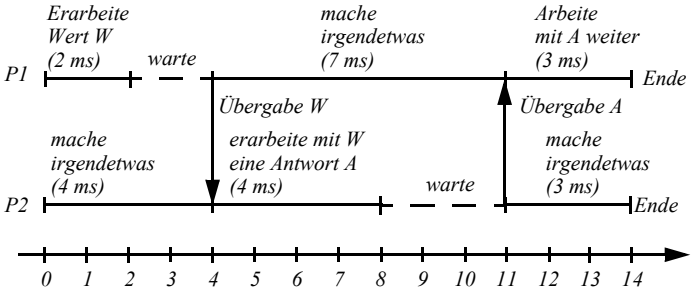



*Client-Server-System ausschließlich mit Ports (der Load Balancer ist ein Prozess, der Aufträge entgegennimmt und „Last" auf die eigentlichen Server verteilt.)*




*3.)bei nonblocking send*

*Diagramm für blocking send*




*4.)Vorteil: Bessere Strukturierung der Software, daher bessere Übersichtlichkeit und Austauschbarkeit*
*Nachteil: Zeitverluste beim Durchlauf der Pakete durch mehrere Schichten*

*5.)drei Schichten*

*obere Schicht mit Wastl und Ludwig II. als Peers*
*mittlere Schicht mit Bürgermeister und Privatsekretär als Peers*
*untere Schicht mit Postvorsteher und Briefträger als Peers*

*Bürgermeister: Brief schreiben; Privatsekretär: Brief vorlesen; Postvorsteher: Brief entgegennehmen, um ihn abzuschicken; Briefträger: Brief zustellen*

# 4A.3 Programmierung unter UNIX/Linux

*1.)UNIX/Linux-Kommunikationsmechanismen und ihre Eigenschaften:*

| | *Shared Memory* | *Pipes* | *Message Queues* | *Sockets* |
|---|---|---|---|---|
| *Erfordet eine explizit ausprogrammierte Reihenfolge-Synchronisation (z.B. durch Semaphore).* | *x* | | | |
| *Ist ein nachrichtenbasierter Mechanismus.* | | *x* | *x* | *(x) (auch strombasiert)* |
| *Ist auch zur Kommunikation zwischen zwei Computern nutzbar.* | | | | *x* |
| *Basiert oft auf TCP/IP.* | | | | *x* |
| *Kann als Basis eines lokalen Client-Server-Systems dienen.* | *x* | *x* | *x* | *x* |

*2.)Der Sender muss einer dringenden Nachricht einen anderen Typ als den normalen Nachrichten zuweisen, und der Empfänger muss zunächst versuchen, eine Nachricht dieses Typ auszulesen, bevor er versucht, andere Nachrichten zu lesen.*

*3.)Benannte Pipes. Unbenannte Pipes sind nur dem Erzeugerprozess und seinen Nachkommen bekannt; die Prozesse, die die beiden Programme ausführen, stehen aber in keiner solchen Beziehung zueinander.*

*Ja. In beiden Programmen muss in den msgget()-Aufrufen als erster Parameter jeweils dieselbe Schlüsselzahl verwendet werden.*

*Bei msgrcv() muss der letzte Parameter auf 0 bzw. IPC_NOWAIT gesetzt werden.*

*4.)Der Vaterprozess ruft printf() auf.*

*Shared Memory*

*Der Empfänger behandelt das Shared-Memory-Segment als einen Speicherbereich des Typs float; korrekt wäre aber int.*

*Die Synchronisation ist nicht korrekt, denn es ist nicht sichergestellt, dass der Vater mit der Ausgabe wartet, bis der Sohn den Wert geschrieben hat. Das kann beispielsweise durch einen wait()-Aufruf vor dem printf() erreicht werden.*

*5.)Der Vater ist der Sender der Daten.*

*Die P- und die V-Operation müssen vertauscht werden.*

*aloha*

*semctl(semid,0,IPC_RMID,0); shmctl(shmid,IPC_RMID, 0);*

*Der Vater darf Semaphor und Shared Memory erst löschen, wenn der Sohn sich beendet hat, also nicht mehr darauf zugreifen möchte.*

*6.)Die close()-Aufrufe schließen das Lese- bzw. Schreibende der Pipe.*

*Es wird eine unbenannte Pipe erzeugt.*

*Mit benannten Pipes können auch Prozesse kommunizieren, die nicht Vater und Sohn/Enkel/Urenkel/... sind.*

```
main() {
 char buffer[8];
 int fd[2];
 pipe(fd);
 if (fork()==0) {
  close(fd[0]);
  write(fd[1],"Message",8);
 exit(0);
 }
 close(fd[1]);
 read(fd[0],buffer,8);
 printf("%s\n",buffer);
}
```

*7.)Die erste Komponente der Nachricht muss ein Wert des Typs long sein. Das ist hier vergessen worden.*

*8.)Es ist nicht korrekt – Begründung siehe 4.2.3.3.*

*9.)Programm 1: socket, connect, write*
*Programm 2: socket, bind, listen, accept, read*

*lokal auf einem Computer*

*in der Variablen buf von Programm 2*

*Der Socket mit dem Deskriptor sd2 dient nur zur Entgegennahme von Verbindungsaufbauwünschen, die eigentliche Datenkommunikation verläuft über die Sockets mit den Deskriptoren sd1 und sd3.*

```
10.)#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main() {

 int sock, error, numbytes, fd;
 char buffer[1024];
 struct sockaddr_in server_addr;
 struct hostent *host;

 sock = socket(AF_INET,SOCK_STREAM,0);
```

```
server_addr.sin_family = AF_INET;
host = gethostbyname("www.nt.fh-koeln.de");
bcopy(host->h_addr,&server_addr.sin_addr,host->h_length);
server_addr.sin_port = htons(80);

error = connect(sock,(struct sockaddr *) &server_addr,sizeof(struct sockaddr_in));

if (error == -1) { ... }

write(sock,"GET /vogt/buecher/nebenlaeufigkeit/ HTTP/1.1\n",45);

write(sock,"Host: www.nt.fh-koeln.de\n",25);

write(sock,"Connection: Close\n\n",19);

fd = open("x.html",O_CREAT|O_WRONLY,0777);
do {
  numbytes = read(sock,buffer,1024);
  write(fd,buffer,numbytes);
} while (numbytes>0);
close(fd);    }
```

*Das Programm hat eine Seite von einem Webserver angefordert, und die neue Datei enthält den HTML-Code dieser Seite.*

*Schickt man das HEAD-Kommando, so enthält die neue Datei nur den Header der Webseite.*

*11.)Die beiden Sohnprozesse haben jeweils ihr eigenen, voneinander getrennten Speicherbereiche, so dass die Zuweisungen des einen Prozesses die Variablenwerte des anderen Prozesses nicht beeinflussen.*

*mit Shared Memory:*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
main() {
 int shmid;
 int *p;
 int status;
 shmid
   = shmget(IPC_PRIVATE,2*sizeof(int),IPC_CREAT|0777);
 if (fork()==0) {
  p = (int *) shmat(shmid,0,0);
  sleep(2);
  printf("Mein Bruder sagt: a=%d, b=%d\n",*p,*(p+1));
  exit(0); }
 if (fork()==0) {
  p = (int *) shmat(shmid,0,0);
  *p=1; *(p+1)=2;
  exit(0); }
 wait(&status);
```

```
 wait(&status);
 shmctl(shmid,IPC_RMID,0);
}
```

*mit Message Queue:*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
main() {
 int msqid;
 int status;
 struct {
  long mtype;
  int a,b;
 } message;
 msqid = msgget(IPC_PRIVATE,IPC_CREAT|0777);
 if (fork()==0) {
  sleep(2);
  msgrcv(msqid,&message,sizeof(message)-sizeof(long),0,0);
  printf("Mein Bruder sagt: a=%d, b=%d\n",message.a,message.b);
   exit(0); }
 if (fork()==0) {
  message.mtype = 1;
  message.a = 1; message.b = 2;
  msgsnd(msqid,&message,sizeof(message)-sizeof(long),0);
  exit(0); }
 wait(&status);
 wait(&status);
 msgctl(msqid,IPC_RMID,0);
}
```

*mit Pipes:*

```
#include <stdio.h>
#include <stdlib.h>
main() {
 int fd[2];
 int a,b;
 pipe(fd);
 if (fork()==0) {
  close(fd[1]);
  sleep(2);
  read(fd[0],&a,sizeof(int));
  read(fd[0],&b,sizeof(int));
  printf("Mein Bruder sagt: a=%d, b=%d\n",a,b);
  exit(0); }
 if (fork()==0) {
```

```
  close(fd[0]);
  a=1; b=2;
  write(fd[1],&a,sizeof(int));
  write(fd[1],&b,sizeof(int));
  exit(0); }
}
```

*mit Sockets:*

*Empfänger:*

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
main() {
 int  server_acc_sock, server_comm_sock, addr_len;
 int buf[2];
 struct sockaddr server_addr, client_addr;
 server_acc_sock = socket(AF_UNIX,SOCK_STREAM,0);
 server_addr.sa_family = AF_UNIX;
 strcpy(server_addr.sa_data,"Socket_1");
 bind(server_acc_sock,&server_addr,sizeof(struct sockaddr));
 listen(server_acc_sock,1);
 addr_len = sizeof(client_addr);
 server_comm_sock = accept(server_acc_sock,&client_addr,&addr_len);
 read(server_comm_sock,buf,2*sizeof(int));
 printf("Empfangen: %d %d\n",buf[0],buf[1]);
 close(server_acc_sock);
 close(server_comm_sock);
 unlink("Socket_1");
}
```

*Sender:*

```
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
main() {
 int client_sock, error;
 struct sockaddr server_addr;
 int buf[2];
 client_sock = socket(AF_UNIX,SOCK_STREAM,0);
 server_addr.sa_family = AF_UNIX;
 strcpy(server_addr.sa_data,"Socket_1");
 error = connect(client_sock,&server_addr,sizeof(struct sockaddr));
 if (error == -1) exit(-1);
 buf[0]=1; buf[1]=2;
```

```
 write(client_sock,buf,2*sizeof(int));
}
```

# 4A.4 Programmierung in Java

```
1.)import java.util.concurrent.*;
import java.io.*;
class Empfaenger extends Thread {
 private PipedInputStream pin;
 Empfaenger(PipedInputStream pin) {
  this.pin = pin;
 }
 public void run() {
  byte gelesen[] = new byte[5];
  try {
    pin.read(gelesen,0,5);
  } catch (IOException exc) {
     System.out.println("Fehler bei read()");
     return;
   }
  System.out.println("Gelesen: "+new String(gelesen));
 }
}
class Sender extends Thread {
 private PipedOutputStream pout;
 Sender(PipedOutputStream pout) {
  this.pout = pout;
 }
 public void run() {
  try {
    pout.write("HAL".getBytes(),0,3);
    pout.write("LO".getBytes(),0,2);
  } catch (IOException exc) {
     System.out.println("Fehler bei write()");
     return;
  }
 }
}
public class SenderEmpfaengerTest {
 public static void main(String[] args)
          throws java.io.IOException, InterruptedException  {
  PipedInputStream pin = new PipedInputStream();
  PipedOutputStream pout = new PipedOutputStream(pin);
  Sender sender = new Sender(pout);
```

```
  Empfaenger empfaenger = new Empfaenger(pin);
  sender.start();
  empfaenger.start();
 }
}
```

2.)

```
public class Server {
 public static void main(String args[]) throws java.io.IOException {
  ServerSocket sockAcc = new ServerSocket(54321);
  Socket sockComm = null;
   try {
    sockComm = sockAcc.accept();
  } catch (IOException e) { ... }
  DataInputStream inStream =
              new DataInputStream(sockComm.getInputStream());
  byte[] buf = new byte[256];
  inStream.read(buf);
  System.out.println("Server hat gelesen: "+(new String(buf)));
 }
}
public class Client {
 public static void main(String args[]) throws java.io.IOException {
  Socket sock = new Socket("www.serverknoten.de",54321);
  PrintStream outStream = new PrintStream(sock.getOutputStream());
  outStream.print("Hallo");
 }
}
```

*3.)Der Server verwendet einen Stream Socket, der Client einen Datagram Socket. Das passt nicht zusammen.*

*4.)Client und Server wollen für ihre Kommunikation, bei der sie offensichtlich beliebige Texte übertragen, Port 80 verwenden. Dieser ist aber für HTTP-basierte Kommunikation mit Web Servern reserviert.*

5.)

```
import java.io.*;
import java.net.*;
public class SocketTest {
 public static void main(String args[]) throws java.io.IOException {
  Socket sock = new Socket("www.nt.fh-koeln.de",80);
  DataInputStream inStream = new DataInputStream(sock.getInputStream());
  DataOutputStream outStream = new DataOutputStream(sock.getOutputStream());
  String befehl = new String("GET /vogt/buecher/nebenlaeufigkeit/ HTTP/1.1\n");
  befehl+="Host: www.nt.fh-koeln.de\n";
  befehl+="Connection: Close\n\n";
  outStream.writeBytes(befehl);
  int numBytes;
  byte buffer[] = new byte[1024];
  do {
```

```
   numBytes = inStream.read(buffer,0,1024);
   if (numBytes>0) {
    PrintWriter printWrit = new PrintWriter(new FileWriter("x.html",true));
    printWrit.print(new String(buffer,0,numBytes));
    printWrit.close();
   }
  }  while (numBytes>0);
 }
}
```

*6.)Server:*

```
import java.io.*;
import java.net.*;
class StreamSocketThread extends Thread {
  public void run() {
   try {
    ServerSocket servSock = new ServerSocket(55555);
    while (true) {
     Socket cliSock = servSock.accept();
     BufferedReader inStream =
        new BufferedReader(new InputStreamReader(cliSock.getInputStream()));
     PrintStream outStream = new PrintStream(cliSock.getOutputStream());
     String s = inStream.readLine();
     outStream.println(s.toUpperCase());
     cliSock.close();
    }
   } catch (IOException e) {}
  }
}
class DatagramSocketThread extends Thread {
  public void run() {
   try {
    DatagramSocket sock = new DatagramSocket(55556);
    while (true) {
     byte[] buf = new byte[256];
     DatagramPacket packIn = new DatagramPacket(buf,buf.length);
     sock.receive(packIn);
     InetAddress ipaddr = packIn.getAddress();
     int port = packIn.getPort();
     buf = (new String(
                  packIn.getData(),0,packIn.getLength())).toUpperCase().getBytes();
     DatagramPacket packOut = new DatagramPacket(buf,buf.length,ipaddr,port);
     sock.send(packOut);
    }
   } catch (IOException e) {}
  }
```

```
}
public class Server {
 public static void main(String args[]) {
  (new StreamSocketThread()).start();
  (new DatagramSocketThread()).start();
 }
}
```

*Client (mit Stream Socket):*

```
import java.io.*;
import java.net.*;
public class Client1 {
 public static void main(String args[]) throws java.io.IOException {
  Socket sock = new Socket("127.0.0.1",55555);

  PrintStream outStream = new PrintStream(sock.getOutputStream());
  BufferedReader inStream =
                new BufferedReader(new InputStreamReader(sock.getInputStream()));
  String parameter = "Hallo Leute";
  System.out.println("Client1 sendet: "+parameter);
  outStream.println(parameter);
  String result = inStream.readLine();
  System.out.println("Client1 empfaengt: "+result);
  sock.close();
  sock = new Socket("127.0.0.1",55555);
  outStream = new PrintStream(sock.getOutputStream());
  inStream = new BufferedReader(new InputStreamReader(sock.getInputStream()));
  parameter = "buchstaben";
  System.out.println("Client sendet: "+parameter);
  outStream.println(parameter);
  result = inStream.readLine();
  System.out.println("Client empfaengt: "+result);
 }
}
```

*Client (mit Datagram Socket):*

```
import java.io.*;
import java.net.*;
public class Client2 {
 public static void main(String args[]) throws java.io.IOException {
  DatagramSocket sock = new DatagramSocket();
  byte[] buf;
  String parameter = "Hallo Leute";
  System.out.println("Client2 sendet: "+parameter);
  buf = parameter.getBytes();
  InetAddress ipaddr = InetAddress.getLocalHost();
```

```
 int port = 55556;
 DatagramPacket packOut = new DatagramPacket(buf,buf.length,ipaddr,port);
 sock.send(packOut);
 buf = new byte[256];
 DatagramPacket packIn = new DatagramPacket(buf,buf.length);
 sock.receive(packIn);
 System.out.println("Client2 empfaengt: "+
                 new String(packIn.getData(),0,packIn.getLength()));
 parameter = "buchstaben";
 System.out.println("Client2 sendet: "+parameter);
 buf = parameter.getBytes();
 packOut = new DatagramPacket(buf,buf.length,ipaddr,port);
 sock.send(packOut);
 buf = new byte[256];
 packIn = new DatagramPacket(buf,buf.length);
 sock.receive(packIn);
 System.out.println("Client2 empfaengt: "+
                     new String(packIn.getData(),0,packIn.getLength()));
}
}
```

*7.)Vereinfachter Server:*

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
main() {
 struct sockaddr_in server_addr;
 int sock;
 char buffer[256];  // zur Aufnahme der empfangenen Nachricht
 // Erzeugen einer Socket, über die Nachrichten
 // von Clients entgegengenommen werden
 sock = socket(AF_INET,SOCK_DGRAM,0);
 server_addr.sin_family = AF_INET;
 server_addr.sin_addr.s_addr = inet_addr("x1.x2.x3.x4");
  // Hier IP-Adresse des Computers einsetzen, auf dem der Server läuft
 server_addr.sin_port = htons(55555);
 bind(sock,(struct sockaddr *) &server_addr,sizeof(struct sockaddr));
 recvfrom(sock,buffer,sizeof(buffer),0,0,0);
 printf("\nGelesen: %s\n\n",buffer);
 close(sock);
}
```

*Java-Client (läuft in dieser Lösung auf demselben Computer wie der Server):*

```
import java.io.*;
import java.net.*;
```

```
public class Client {
 public static void main(String args[]) throws java.io.IOException {
  DatagramSocket sock = new DatagramSocket();
  byte[] buf = new byte[256];
  buf = "Hallo\0".getBytes();
  InetAddress ipaddr = InetAddress.getLocalHost();
  int port = 55555;
  DatagramPacket packOut = new DatagramPacket(buf,buf.length,ipaddr,port);
  sock.send(packOut);
 }
}
```