Kommunikation und Synchronisation von Prozessen

1. Hintergrundinformation
2. Umsetzungsmöglichkeiten

Teil 1 Hintergrundinformation

Grundansatz

Vorspanntext Lehrplan 12.2: Betrachtet werden Situationen im Internet, d. h. Kommunikation von Prozessen im Internet und Synchronisation von parallel abzuarbeitenden Serveraufträgen. Das bedeutet:

- Praktische Kommunikation erfolgt mit TCP/IP.
- Der Prozessbegriff muss in diesem Kapitel „nebenher“ eingeführt werden.
- Parallelität wird durch leichtgewichtige Prozesse (Threads) erreicht.

Netztopologie

- Typische Topologien und deren Grundeigenschaften müssen bekannt sein; keine tiefer gehenden Details.
- Erkenntnis, dass das Internet aus Teilnetzen zusammengesetzt ist, ist notwendig, um die Notwendigkeit für Schichtenkommunikation zu erkennen.
- Aus den Teilnetzen ergibt sich die Notwendigkeit für Routing; das Wie wird nicht thematisiert.

Kommunikation

- Kommunikation ist ein allgemeiner Begriff; betrachtet wird aber nur Netzwerkkommunikation
- Voraussetzung sind eindeutige Bezeichner
- Protokolle regeln die Kommunikation; sie sind formale Sprachen, die Dialoge beschreiben (Anknüpfung an 12.1)
- Schichten sind notwendig, damit nicht jeder den Aufbau des Gesamtnetzwerks kennen muss; ISO/OSI-Modell ist schön, muss aber nicht sein.
- IP-Adressen und DNS werden für praktische Kommunikation benötigt.

Parallele Prozesse

- Prozessbegriff als ausführbare Aufgabe; leicht- und schwergewichtige Prozesse (Datenraum)
- Konzept der Threads für leichtgewichtige Prozesse
- keine Schedulingstrategien oder ähnliches

Nebenläufige Prozesse

- Parallele Prozesse operieren auf gemeinsamen Ressourcen (z. B. Daten)
- Kritischer Abschnitt: Zugriff auf gemeinsame Ressourcen, problematisch, wenn Änderungen passieren.
- Sequenzdiagramme zur Betrachtung kritischer Situationen
- keine Petrinetze o. ä.

Monitorkonzept

- Semaphore entsprechen Maschinensprache, Monitore entsprechen Hochsprache
- Einfache Ausschlüsse, aber auch Warten auf anderen Prozess und nötige Konzeptergänzungen (passives Warten)
- Verklemmung nur als Möglichkeit, keine Konzepte zur Verhinderung oder Vermeidung

Implemetierung

- Nebenläufige Prozesse müssen implementiert werden (Threads, Monitore, passives Warten)
- Kommunikation über TCP/IP sollte implementiert werden (Handlungsorientierter Ansatz); hier sind gute Möglichkeiten zur Binnendifferenzierung.

Teil 2 Umsetzungsvorschläge

Unterrichtssequenz

- Alle geforderten Lehrplanteile müssen entsprechend enthalten sein.
- Reihenfolge gemäß der Abfolge der Fragestellungen, nicht gemäß der Spiegelstriche (Begründung der Schritte aus der zu lösenden Aufgabe)
- Hoher Anteil an Schüleraktivität; Theorie ist kein Selbstzweck

Unterrichtssequenz

1. Kommunikation in Rechnernetzen 2h
   (Historischer Start, Protokoll, Topologie , Schichten, Routing)
2. Prozesskommunikation im Internet 7h
   (IP-Stack, typische Protokolle, IP-Adressen / Ports, Verbindungen)
3. Parallele Prozesse 4h
   (Prozesskonzept, Threads)
4. Synchronisation - Monitore 7h
   (Nebenläufigkeit, Monitore, passives Warten, Verklemmung)

Summe: 20h

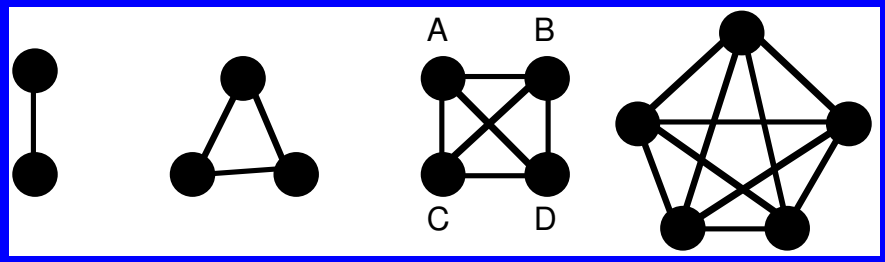
1 Kommunikation in Rechnernetzen

Aus dem historischen Beginn (Dateiübertragung) wird der Bedarf für „Protokolle“ hergeleitet: Feststellen der benötigten Semantik, festlegen einer geeigneten Syntax.
Ausschnitt:

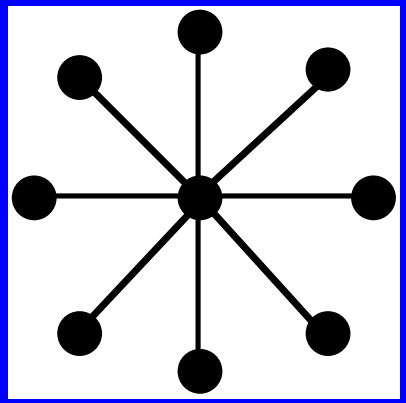
Partner	Formalsprache	Kommentar
1	"schicken" "<cr><lf>"	sendet den Wunsch, eine Datei zu senden
2	"verstanden" "<cr><lf>"	meldet ãverstandenÒ
1	Dateiname	sendet den Zielnamen
2	"verstanden" "<cr><lf>"	meldet ãverstandenÒ
1	Inhalt	sendet den Dateiinhalt
2	"verstanden" "<cr><lf>"	meldet ãverstandenÒ

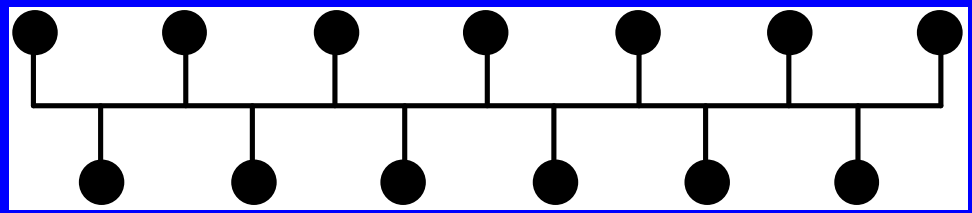
1 Kommunikation in Rechnernetzen

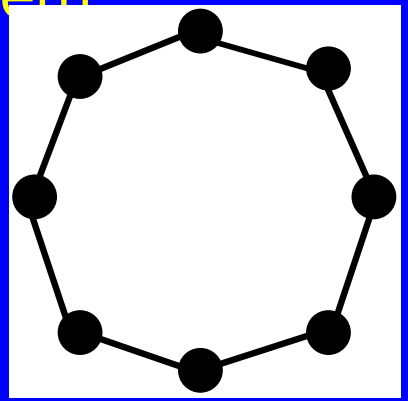
Die zunehmende Rechnerzahl lässt Point-to-Point nicht zu.



Es entwickeln sich Netze mit benannten Rechnern

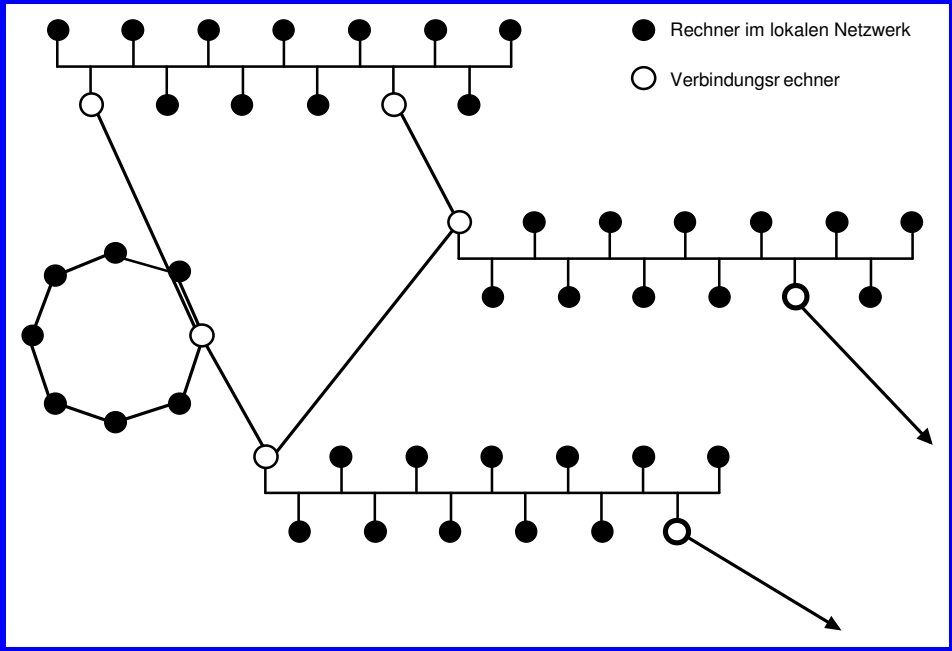






1 Kommunikation in Rechnernetzen

Diese werden zu größeren Einheiten verbunden.



1 Kommunikation in Rechnernetzen

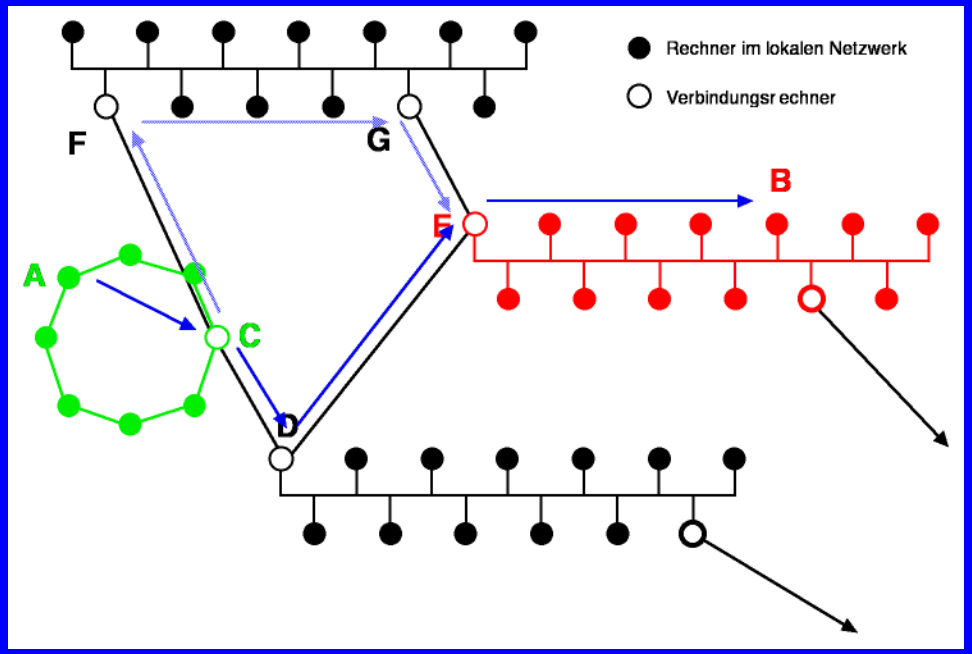
Zwei Rechner A und B wollen kommunizieren, um
Details sollen sich andere kümmern =>
Schichtenmodelle, z. B. ISO/OSI Referenzmodell

OSI-Schicht		Orientierung
7	Anwendung (application)	anwendungsorientiert
6	Darstellung (presentation)	
5	Sitzung (session)	
4	Transport (transport)	transportorientiert
3	Vermittlung (network)	
2	Sicherung (data Link)	
1	BitŸbertragung (physical)	

Zur Anzeige wird der QuickTime™ Dekompressor „“ benötigt.

1 Kommunikation in Rechnernetzen

Routing ist ein Problem der unteren Schichten



2 Prozesskommunikation im Internet

Übertragung der allgemeinen Konzepte auf das Internet:

- Rechnerbezeichner -> IP-Adressen (Ergänzung DNS)
- Ports als ergänzendes Prinzip
- IP-Stack mit Protokollen

	TCP/IP-Schicht	**typische Protokolle**	**OSI-Schicht**
4	Anwendung	http, ftp, smtp, pop3, imap	5Ğ7
3	Transport	tcp, udp, tsl	4
2	Vermittlung	ip	3
1	Medien	ethernet, token-ring, diverse Funkprotokolle	1Ğ2

2 Prozesskommunikation im Internet

- Client - Server - Konzept
- Kommunikation mit der Klasse SOCKET
- Sprachspezifische Ausprägung

SOCKET
Anbinden (portnummer) Warten () Verbinden (adresse, portnummer) Lesen () Schreiben (daten) Schliessen ()

Ein typischer Serverprozess hat damit folgenden Kernteil:

```
// Verbindungsobjekt erzeugen
verbindung1 = new SOCKET()
// Verbindungsobjekt bei der Portverwaltung registrieren
verbindung1. Anbinden (<meinePortNummer>)
tue
        // Auf einen Anruf warten; die Methode gibt dann ein
        // Objekt zurück, das die konkrete Verbindung beschreibt.
        verbindung2 = verbindung1.Warten()
        <führe Auftrag von verbindung2 aus>
solange nicht <Bedingung für Beenden des Servers>
```

Entsprechend kurz ist auch der Kern des Clients:

```
// Verbindungsobjekt erzeugen
verbindung = new SOCKET()
// Mit dem Server Verbinden
verbindung.Verbinden(<serveradresse>, <serverPortNummer>)
<Arbeit erledigen>
// Verbindung beenden
verbindung.Schliessen()
```

2 Prozesskommunikation im Internet

Ablauf der Aufgaben:

- Test mit Standardprogrammen (z. B. Lesen einer Webseite mit telnet / putty)
- Implementieren des Beispiels in Java
- Implementieren eines einfachen Servers (z. B. Chatserver) und Test (mit Telnet)
- Implementieren des zugehörigen Klienten

3 Parallele Prozesse

- Grundidee aus der parallelen Abarbeitung der Aufträge (z. B. Webserver, Chatserver)
- Prozess als eigenständig ausführbare Aufgabe
- Threadkonzept in Java

Klasse für eigene Threads:

```
class MEINTHREAD extends Thread
{
...
public void run()
{
<Hier wird die Arbeit erledigt.>
}
...
}
```

Anlegen und Starten:

```
new MEINTHREAD().start();
```

3 Parallele Prozesse

- Zuerst Aufgaben ohne Netzwerk, z. B. Roboter
- Erweiterung begonnener Aufgaben (ChatServer)

```
void AnrufAnnahme()
{
try
{
    ServerSocket server = new ServerSocket(34000);
    while (true)
    {
        Socket anruf = server.accept();
        new CLIENTTHREAD(anruf).start();
    }
}
catch (Exception e)
{
    e.printStackTrace();
}
}
```

```
class CLIENTTHREAD extends Thread
{
private OutputStream out;    // Datenströme für die
private InputStream in;      // Kommunikation
private Socket socket;
CLIENTTHREAD(Socket so) throws IOException
{   // Im Konstruktor wird die Verbindung übergeben
    socket = so;
    out = so.getOutputStream();
    out.flush();
    in = so.getInputStream());
}
public void run()
{
    do
    {
        <Abarbeiten der Aufträge des Klienten>
    }
    while (! <Endebedingung>);
    try
    {
        socket.close();
    }
    catch (Exception e)
    {
        // nichts zu tun
    }
}
}
```

4 Synchronisation - Monitore

- Bei der Beobachtung der im letzen Kapitel ergänzten Aufgaben fallen Fehler auf => genaue Analyse der Situation nötig
- Identifikation der Probleme mithilfe von Sequenzdiagrammen
- Begriff des kritischen Abschnitts

Zur Anzeige wird der QuickTime™ Dekompressor „“ benötigt.

4 Synchronisation - Monitore

- Wissen, dass Softwarelösungen nicht möglich sind
- Semaphore (voriges Jahr) zu unhandlich; neues Konzept: Monitor
- Umsetzung in Java: `synchronized`

Zur Anzeige wird der QuickTime™ Dekompressor „“ benötigt.

Zur Anzeige wird der QuickTime™ Dekompressor „“ benötigt.

4 Synchronisation - Monitore

- Problem des Busy-Wait beim typischen Erzeuger-Verbraucher-Problem
- Konzept: Warten und Benachrichtigen
- Umsetzung in Java: *wait*, *notify*, *notifyAll*
- Vervollständigung der Beispielprogramme
- mögliche Verklemmungen

Zur Anzeige wird der QuickTime™ Dekompressor „“ benötigt.

ENDE

Zur Anzeige wird der QuickTime™
Dekompressor „“
benötigt.