

Script generated by TTT

Title: Grundlagen_Betriebssysteme (13.01.2012)

Date: Fri Jan 13 08:30:30 CET 2012

Duration: 82:50 min

Pages: 23

Disjunkte Prozesse, d.h. Prozesse, die völlig isoliert voneinander ablaufen, stellen eher die Ausnahme dar. Häufig finden Wechselwirkungen zwischen den Prozessen statt \Rightarrow Prozesse interagieren. Die Unterstützung der Prozessinteraktion stellt einen unverzichtbaren Dienst dar.

Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Mechanismen von Rechensystemen zum Austausch von Informationen zwischen Prozessen.

Kommunikationsarten.

- nachrichtenbasierte Kommunikation, insbesondere Client-Server-Modell.
- Netzwerkprogrammierung auf der Basis von Ports und Sockets.

[Einführung](#)
[Nachrichtenbasierte Kommunikation](#)
[Client-Server-Modell](#)
[Netzwerkprogrammierung](#)

Generated by Targeteam

Nachrichtenbasierte Kommunikation

Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschicken und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive`.

[Elementare Kommunikationsmodelle](#)
[Erzeuger-Verbraucher Problem](#)
[Modellierung durch ein Petrinetz](#)
[Ports](#)
[Kanäle](#)
[Ströme](#)
[Pipes](#)

Generated by Targeteam

Sender wird lediglich bis zur Ablieferung der Meldung an das Nachrichtensystem (Kommunikationssystem) blockiert.

```
sequenceDiagram
    participant S as Sender S
    participant ND as Nachrichtendienst ND
    participant E as Empfänger E
    S->>ND: send
    ND->>E: Meldung
    E-->>ND: receive
```

Nachrichtendienst des Betriebssystems puffert Nachricht;

Sender S kann seine Ausführung fortsetzen, sobald Nachricht N in den Nachrichtenpuffer des ND eingetragen ist.

S wartet *nicht*, bis E die Nachricht empfangen hat.

Empfänger E zeigt durch `receive` an, dass er am Empfang der Nachricht N interessiert ist.

Empfänger wird blockiert, bis Sender Nachricht bereit stellt.

Generated by Targeteam



Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:
 generelles Muster der Nachrichtenkommunikation.
 zeitliche Kopplung der beteiligten Prozesse.

Klassifikationsschema

Meldung

[Asynchrone Meldung](#)

[Synchrone Meldung](#)

Auftrag

[Synchrone Auftrag](#)

[Asynchrone Auftrag](#)

[Vorteile/Nachteile asynchrones Senden](#)

Generated by Targem



Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:
 generelles Muster der Nachrichtenkommunikation.
 zeitliche Kopplung der beteiligten Prozesse.

Klassifikationsschema

Meldung

[Asynchrone Meldung](#)

[Synchrone Meldung](#)

Auftrag

[Synchrone Auftrag](#)

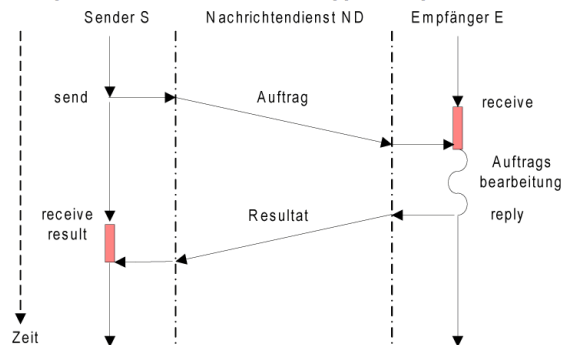
[Asynchrone Auftrag](#)

[Vorteile/Nachteile asynchrones Senden](#)

Generated by Targem



Auftrag und Resultat werden als Paar unabhängiger Meldungen verschickt.



Generated by Targem



Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:
 generelles Muster der Nachrichtenkommunikation.
 zeitliche Kopplung der beteiligten Prozesse.

Klassifikationsschema

Meldung

[Asynchrone Meldung](#)

[Synchrone Meldung](#)

Auftrag

[Synchrone Auftrag](#)

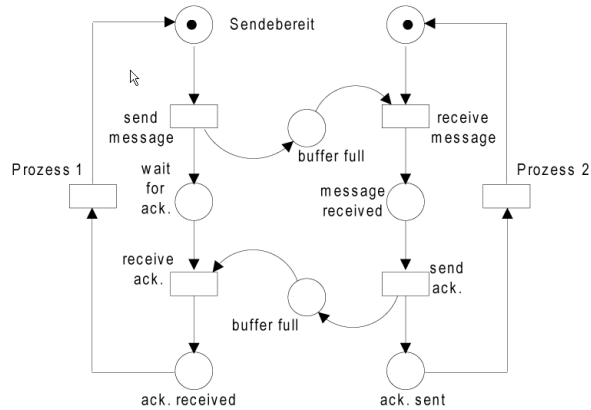
[Asynchrone Auftrag](#)

[Vorteile/Nachteile asynchrones Senden](#)

Generated by Targem



Petri-Netze dienen häufig zur Modellierung von Kommunikationsabläufen, sogenannten Kommunikationsprotokollen. Sie ermöglichen die Analyse der Protokolle, z.B. Erkennung von Verklemmungen. Modellierung einer synchronen Kommunikation:



Problem: unendliches Warten

Pragmatische Lösung mit Hilfe von Timeouts



Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschicken und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive`.

[Elementare Kommunikationsmodelle](#)

[Erzeuger-Verbraucher Problem](#)

[Modellierung durch ein Petrinetz](#)

[Ports](#)

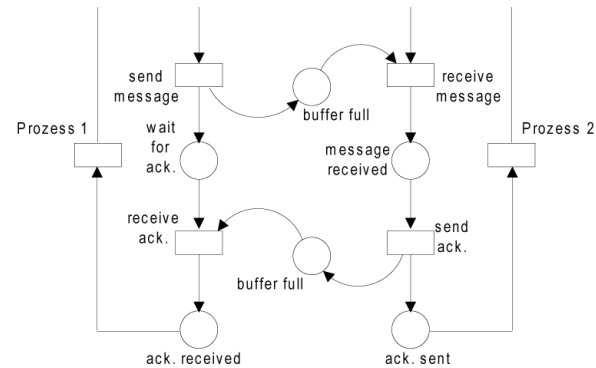
[Kanäle](#)

[Ströme](#)

[Pipes](#)

Generated by Targem

Modellierung durch ein Petrinetz



Problem: unendliches Warten

Pragmatische Lösung mit Hilfe von Timeouts

Sender bzw. Empfänger **warten nur eine festgelegte Zeit**; Sender: falls kein Acknowledgement (Quittung) eintrifft, z.B. erneutes Senden.

Probleme dabei? u.a. Duplikate müssen vom Empfänger erkannt werden; gesendete Nachrichten kommen zu spät an, sind veraltet etc.

Generated by Targem



Bisher bestand zwischen Sender und Empfänger eine feste Beziehung, die über Prozessidentifikatoren (z.B. Namen oder Nummer) hergestellt wurde. Nachteile

Prozessnummern ändern sich mit jedem Neustart.

Prozessnamen sind nicht eindeutig, z.B. falls Programm mehrmals gestartet wurde.

⇒ Deshalb Senden von Nachrichten an **Ports**. Sie stellen Endpunkte einer Kommunikation dar. Sie können bei Bedarf dynamisch eingerichtet und gelöscht werden. Dazu existieren folgende Funktionen:

```
portID = createPort();
deletePort(portID);
send(E.portID, message);
receive(portID, message);
```

ein Port ist mit dem Adressraum des zugehörigen Prozesses verbunden.

der Empfängerprozess kann sender-spezifische Ports einrichten.

ein Rechner mit einer IP-Adresse unterstützt mehrere tausend Ports.

der Name des Ports ist für einen Rechner eindeutig.

die Portnummern 1 - 1023 sind fest reserviert für bestimmte Protokolle (bzw. deren Applikationen).

[Übersicht: fest zugeordnete Ports](#)

Generated by Targem

Protokoll	Port	Beschreibung
FTP	21	Kommandos für Dateitransfer (get, put)
Telnet	23	interaktive Terminal-Sitzung mit entferntem Rechner
SMTP	25	Senden von Email zwischen Rechnern
time	37	Time-Server liefert aktuelle Zeit
finger	79	liefert Informationen über einen Benutzer
HTTP	80	Protokoll des World Wide Web
POP3	110	Zugang zu Email durch einen sporadisch verbundenen Client
RMI	1099	Zugang zum Registrieren von entfernten Java Objekten.

Generated by Targem



Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschieben und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive`.

[Elementare Kommunikationsmodelle](#)

[Erzeuger-Verbraucher Problem](#)

[Modellierung durch ein Petrinetz](#)

[Ports](#)

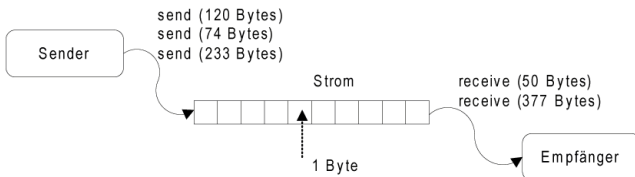
[Kanäle](#)

[Ströme](#)

[Pipes](#)

Generated by Targem

Ströme (engl. streams) sind eine Abstraktion von Kanälen. Sie verdecken die tatsächlichen Nachrichtengrenzen.



BS-Dienste: Verbindungsauf- und -abbau, schreiben in Strom, lesen aus Strom.

Dienste für Dateizugriffe oder Zugriffe auf Geräte: spezielle Ausprägung der stromorientierten Kommunikation.

I/O in Java basiert auf Ströme.

Klasse `java.io.OutputStream` zum Schreiben von Daten

Klasse `java.io.InputStream` zum Lesen von Daten

Spezialisierungen z.B. durch `FileOutputStream`, `BufferedOutputStream` oder `FileInputStream`.

Generated by Targem

Realisierung von Strömen über Pipe-Konzept (z.B. in Unix, Windows); Strom zwischen 2 Kommunikationspartnern

unidirektional

FIFO-artiger Datentransfer mit Operationen: `open pipe`, `read`, `write`.

gepuffert und zuverlässig.

Pipes können als Dateien ohne Plattenbeteiligung betrachtet werden.



Ordnung der Zeichen bleibt erhalten (Zeichenstrom)

Blockieren bei voller Pipe (`write`) und leerer Pipe (`read`)

Beispiele für Unix Pipes

```
ls -l | head
```

```
grep "name" datei.txt | sort | more
```

```
cat datei.txt | awk '{print $1}' | more
```

[Prinzipieller Aufbau](#)

[Beispielnutzung einer Pipe](#)

[Nutzung Pipes](#)

Der pipe-Systemaufruf erzeugt ein i-node Objekt sowie 2 File-Objekte. Damit können Prozesse mit read() und write() Systemaufrufen darauf zugreifen.

```

struct pipe_inode_info {
    wait_queue_head_t wait; /* Warteschlange blockierter Prozesse, die auf
    Pipe zugreifen wollen */
    char *base; /* Adresse des Kernel Buffers */
    unsigned int len; /* Anzahl geschriebener, aber noch nicht gelesener
    Bytes */
    unsigned int start; /* Leseposition im Buffer */
    unsigned int readers; /* Anzahl lesender Prozesse */
    unsigned int writers; /* Anzahl schreibender Prozesse */
    unsigned int waiting_writers;
    unsigned int r_counter; /* Anzahl der Prozesse, die auf neue Zeichen in
    der Pipe warten */
    unsigned int w_counter; /* Anzahl der Prozesse, die Zeichen in die Pipe
    schreiben möchten */
}

```

Zugriffe auf Pipe müssen intern über Semaphore geregelt werden.

Generated by Targem

Beispielnutzung einer Pipe

Die Pipe wird im Vaterprozess angelegt, der anschließend 2 Kindprozesse startet, die über eine Pipe kommunizieren.

Systemaufruf dup() ermöglicht Umlenkung von Standard-Ein-/Ausgabe auf Pipe

```

main (int argc, char *argv[]) {
    int fhandle[2]; int rc;
    pipe(fhandle); /* erzeugt Pipe*/
    rc = fork();
    if (rc ==0) {
        close(0); /* Schließen Standard Eingabe */
        dup(fhandle[0]); /* Umlenken von fhandle[0] auf Standard Eingabe */
        close(fhandle[0]); /* Schließen der nicht mehr benötigten fhandle[0] */
        /* der nun gestartete Prozess liest bei Zugriff auf Standard
        Eingabe aus der Pipe */
        rc = execl("kind1", "Kind 1", "\0");
    } else {
        rc = fork(); /* Vaterprozess */
        if (rc == 0) {
            close(1); /* Schließen Standard Ausgabe */
            dup(fhandle[1]); /* Umlenken von fhandle[1] auf Standard Ausgabe */
            close(fhandle[1]); /* Schließen der nicht mehr benötigten

```

Beispielnutzung einer Pipe

```

int fhandle[2]; int rc;
pipe(fhandle); /* erzeugt Pipe*/
rc = fork();
if (rc ==0) {
    close(0); /* Schließen Standard Eingabe */
    dup(fhandle[0]); /* Umlenken von fhandle[0] auf Standard Eingabe */
    close(fhandle[0]); /* Schließen der nicht mehr benötigten fhandle[0] */
    /* der nun gestartete Prozess liest bei Zugriff auf Standard
    Eingabe aus der Pipe */
    rc = execl("kind1", "Kind 1", "\0");
} else {
    rc = fork(); /* Vaterprozess */
    if (rc == 0) {
        close(1); /* Schließen Standard Ausgabe */
        dup(fhandle[1]); /* Umlenken von fhandle[1] auf Standard Ausgabe */
        close(fhandle[1]); /* Schließen der nicht mehr benötigten
        fhandle[1] */
        /* der nun gestartete Prozess schreibt bei Zugriff auf
        Standard Ausgabe in die Pipe; zwischen den Kindern ist
        dadurch eine unidirektionale Kommunikation mittels Pipes
        erzeugt worden */
        rc = execl("kind2", "Kind 2", "\0");

```

Handwritten notes: Kind 1 (next to the first child block), Vater (next to the parent block), Kind 2 (next to the second child block). A red line underlines the comment: "Jetzt erfolgt Kommunikation über Std-Obj".

Beispielnutzung einer Pipe

```

    close(fhandle[1]); /* Schließen der nicht mehr benötigten
    fhandle[1] */
    /* der nun gestartete Prozess schreibt bei Zugriff auf
    Standard Ausgabe in die Pipe; zwischen den Kindern ist
    dadurch eine unidirektionale Kommunikation mittels Pipes
    erzeugt worden */
    rc = execl("kind2", "Kind 2", "\0");
}
sleep(1);
exit(0);
}

```

Handwritten notes: Kind 1 (next to the first child block), Vater (next to the parent block), Kind 2 (next to the second child block). A red line underlines the comment: "Jetzt erfolgt Kommunikation über Std-Obj".

Named Pipes

Problem: nur Prozesse, die über `fork()` eng miteinander verwandt sind, können im Beispielprogramm kommunizieren.

Einführung von Named Pipes, die mittels `mknod` erzeugt werden.

ermöglicht die Kommunikation zwischen Prozessen, die nicht miteinander verwandt sind.

```
main (int argc, char *argv[]) {
    int rc;
    mknod("myfifo", 0600, S_IFIFO);
    if (rc == 0) {
        int fd;
        fd = open("myfifo", O_WRONLY);
        write(fd, "Satz 1\n", 7);
        close(fd);
        exit(0)
    } else {
        int fd;
        char block[20];
        fd = open("myfifo", O_RDONLY);
        rc = read(fd, block, 7);
        write(1, ":", 1);
        if (rc > 0) { write(1, block, rc); 0
    }
```

Beispielnutzung einer Pipe

Die Pipe wird im Vaterprozess angelegt, der anschließend 2 Kindprozesse startet, die über eine Pipe kommunizieren.

Systemaufruf `dup()` ermöglicht Umlenkung von Standard-Ein-/Ausgabe auf Pipe

```
main (int argc, char *argv[]) {
    int fhandle[2]; int rc;
    pipe(fhandle); /* erzeugt Pipe*/
    rc = fork();
    if (rc == 0) {
        close(0); /* Schließen Standard Eingabe */
        dup(fhandle[0]); /* Umlenken von fhandle[0] auf Standard Eingabe */
        close(fhandle[0]); /* Schließen der nicht mehr benötigten fhandle[0] */
        /* der nun gestartete Prozess liest bei Zugriff auf Standard
        Eingabe aus der Pipe */
        rc = execl("kind1", "Kind 1", "\0");
    } else {
        rc = fork(); /* Vaterprozess */
        if (rc == 0) {
            close(1); /* Schließen Standard Ausgabe */
            dup(fhandle[1]); /* Umlenken von fhandle[1] auf Standard Ausgabe */
            close(fhandle[1]); /* Schließen der nicht mehr benötigten
```

Named Pipes

Problem: nur Prozesse, die über `fork()` eng miteinander verwandt sind, können im Beispielprogramm kommunizieren.

Einführung von Named Pipes, die mittels `mknod` erzeugt werden.

ermöglicht die Kommunikation zwischen Prozessen, die nicht miteinander verwandt sind.

```
main (int argc, char *argv[]) {
    int rc;
    mknod("myfifo", 0600, S_IFIFO);
    if (rc == 0) {
        int fd;
        fd = open("myfifo", O_WRONLY);
        write(fd, "Satz 1\n", 7);
        close(fd);
        exit(0)
    } else {
        int fd;
        char block[20];
        fd = open("myfifo", O_RDONLY);
        rc = read(fd, block, 7);
        write(1, ":", 1);
        if (rc > 0) { write(1, block, rc); 0
    }
```

The screenshot shows a Windows Internet Explorer browser window. The address bar contains the URL `C:\www\gbs-ws1112\flash\gbs_course6_2.7.3.html`. The page title is "Named Pipes". The content of the page is identical to the "Named Pipes" page shown in the other screenshots, including the problem statement, the introduction of Named Pipes, and the C code snippet. The browser's taskbar at the bottom shows the Start button and several open windows, including "Named Pipes - Windo..." and "IN009_Grundlagen...". The system tray shows the time as 09:53.