Fakultät für Informatik

MNM TEAM
Munich Network Management Team

Institut für Informatik

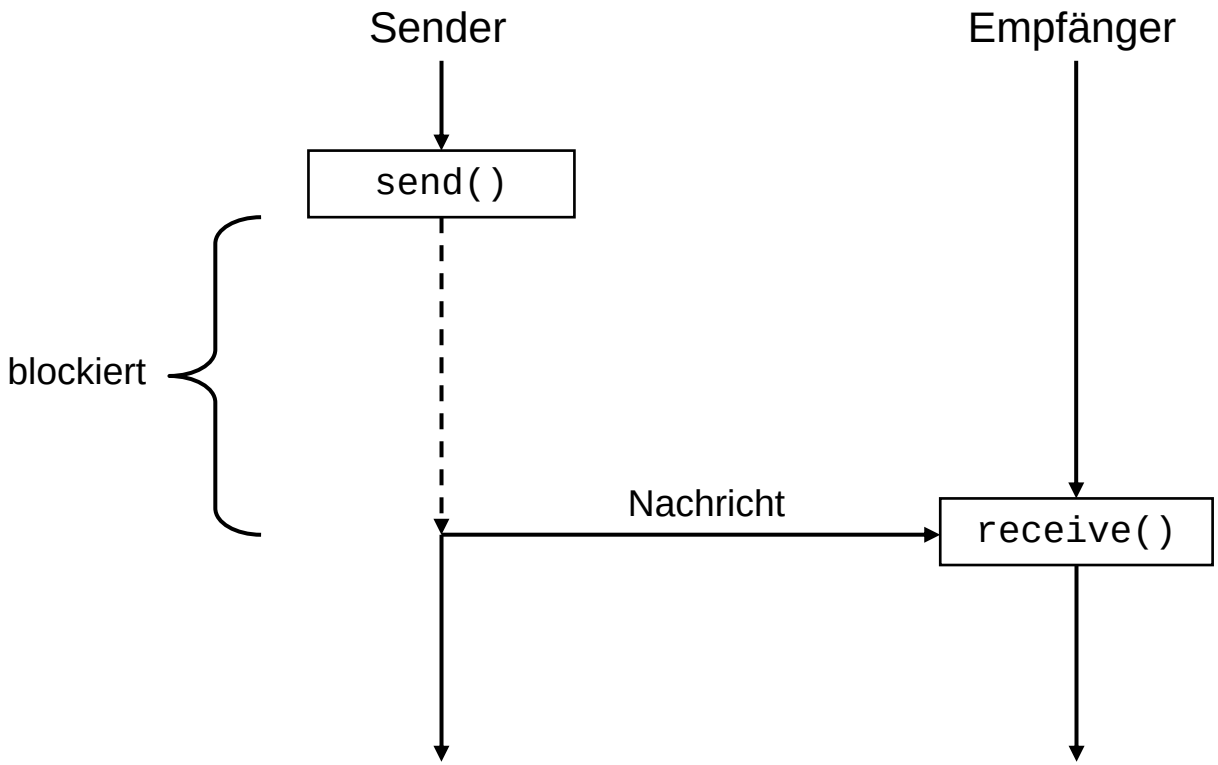
LMU

Systempraktikum im Wintersemester 2009/2010 (LMU):
Vorlesung vom 19.11. – Foliensatz 4

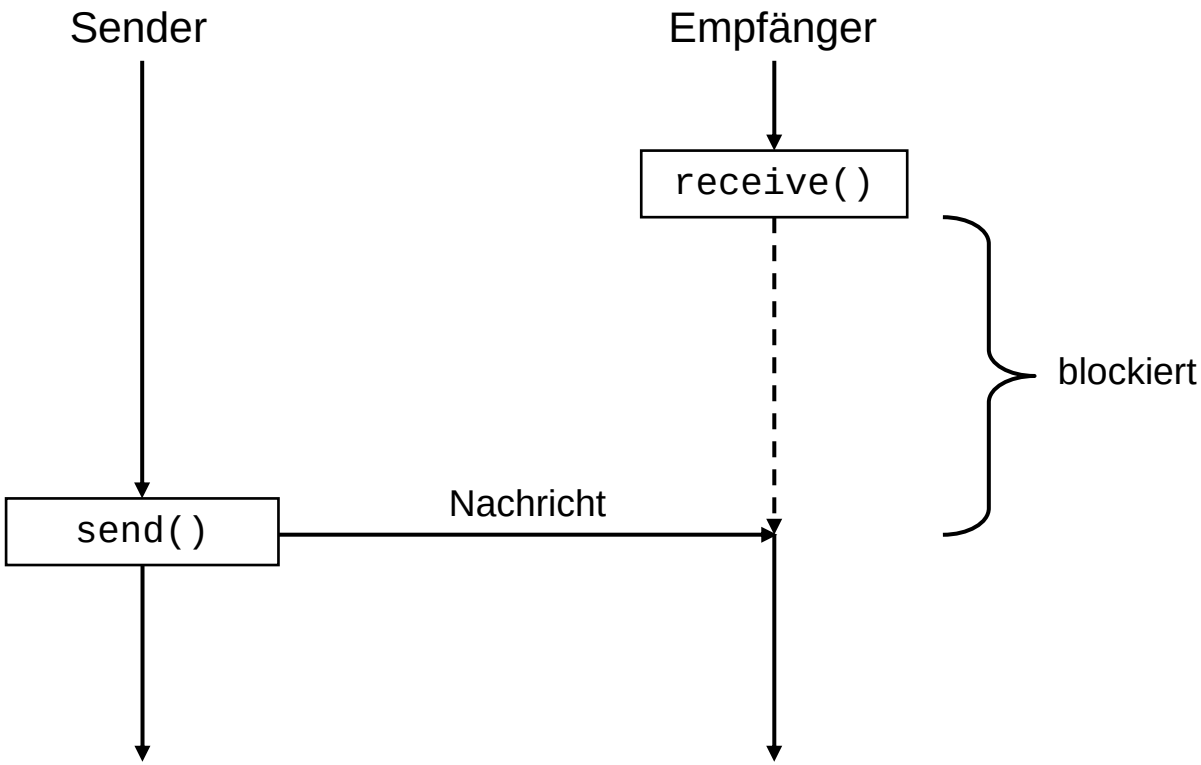
Kommunikationsmuster (T)
Interprozesskommunikation (T)
Kommunikation über Pipes (P)
Parserspezifikation und -generierung (P)

Thomas Schaaf, Nils gentschen Felde

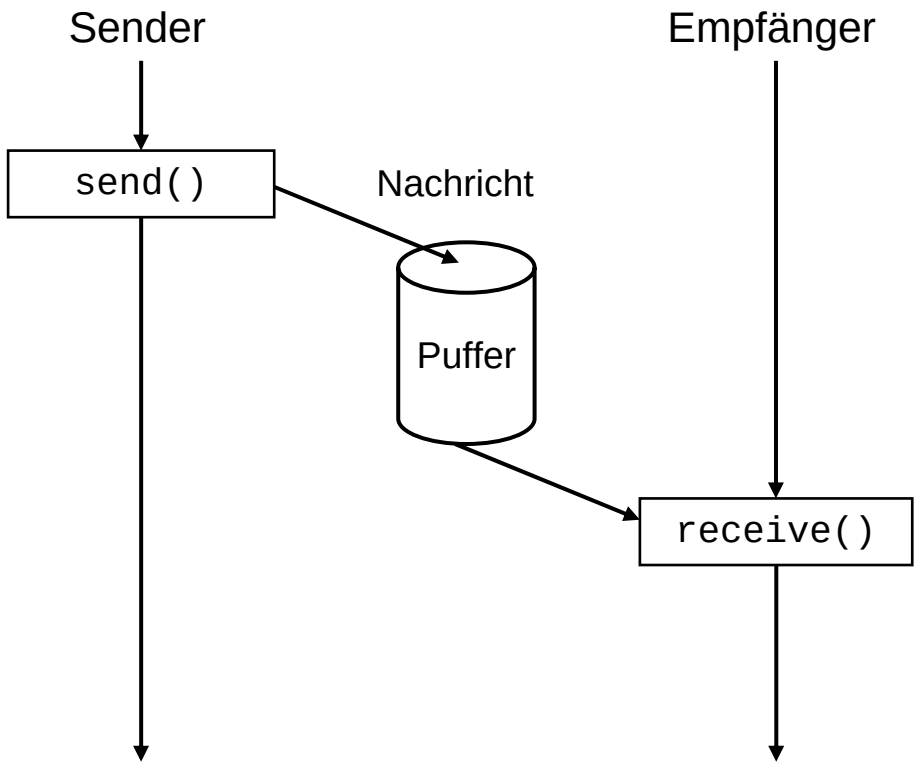
- Blockierende Kommunikation
  - 1. Blockierendes Senden:



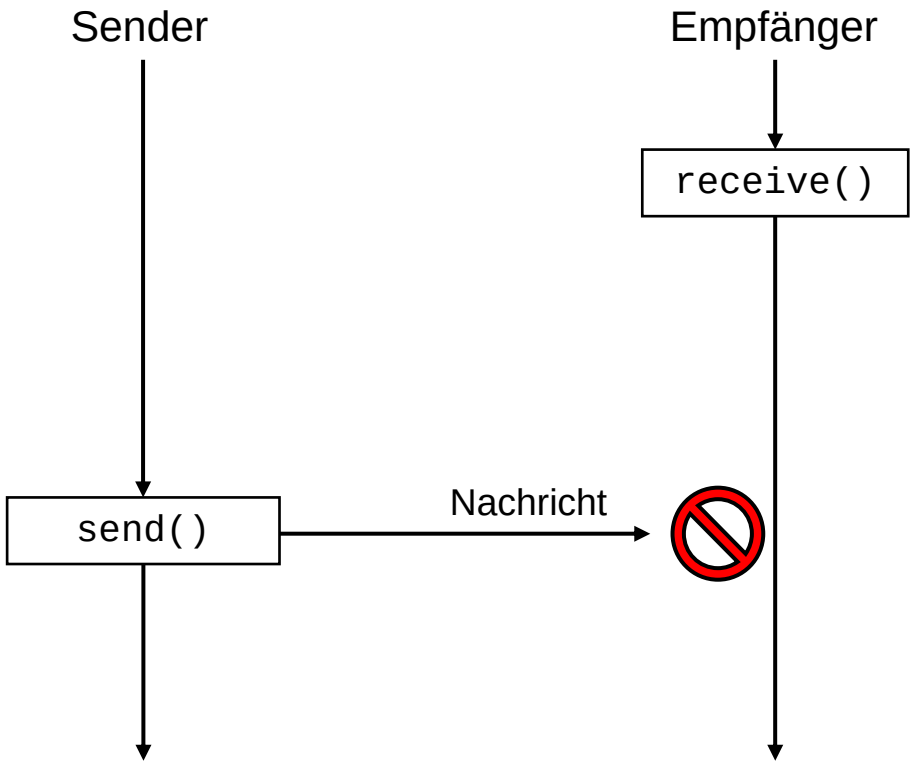
- Blockierende Kommunikation (Forts.)
  - 2. Blockierendes Empfangen:



- Nicht-blockierende Kommunikation
  - 3. Nicht-blockierendes Senden:



- Nicht-blockierende Kommunikation
  - 4. Nicht-blockierendes Empfangen:



Kommunikationsmuster

MNM TEAM

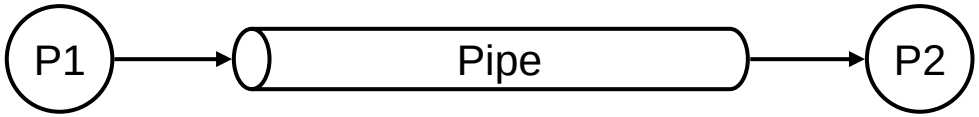
- Zusammenfassung/Varianten
  - Blockierendes Senden + blockierendes Empfangen
    → (Logische) Synchrone Kommunikation (Rendezvous-Konzept)
  - Nicht-blockierendes Senden + blockierendes Empfangen + Nachrichtenpuffer
    → Asynchrone Kommunikation
  - Blockierendes Senden + nicht-blockierendes Empfangen
    → keine zuverlässige Kommunikation
  - Nicht-blockierendes Senden + nicht-blockierendes Empfangen
    → keine zuverlässige Kommunikation

Interprozesskommunikation

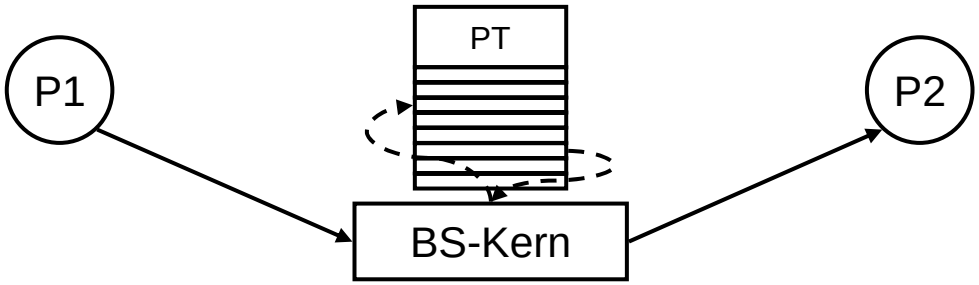
MNM TEAM

- Kommunikationsprotokolle
  - Protokoll definiert Regeln für den Informationsaustausch
  - Teile einer Protokolldefinition:
    - Verwendete Codes
    - Nachrichtenlänge
    - Nachrichtenformat
    - Form der Adressierung
    - Bestätigungen
    - Fehlercodes
    - u.a.
  - Anwendungsbeispiel: Client-Server-Kommunikation

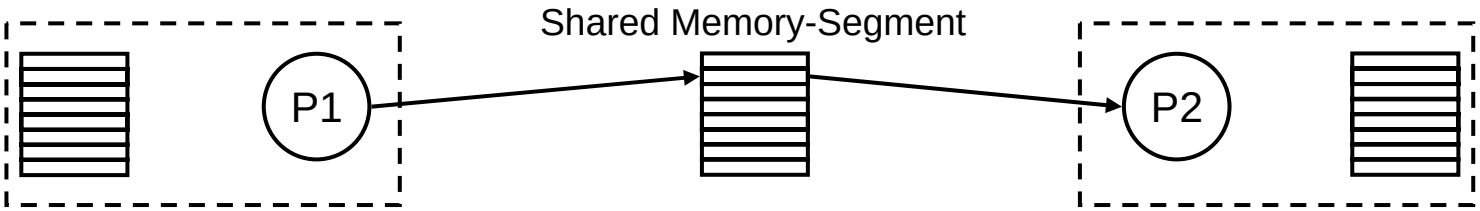
- Überblick: Interprozesskommunikation (stark abstrahiert)
  - Pipes und FIFOs:



  - Signale:



  - Speicherbasierte Kommunikation (Shared Memory-Konzept):



  - Netzbasierte Kommunikation (TCP/IP):

Kommunikation über Pipes

MNM TEAM

- Wiederholung: (Namenlose) Pipes
  - Eigenschaften einer Pipe:
    - Unidirektionale Kommunikation
    - Nachrichtenreihenfolge bleibt erhalten (FIFO)
    - Aufbau von Pipes nur zwischen "verwandten" Prozessen
  - Einrichten einer Pipe:

```
int pipe(int fd[2]);
```

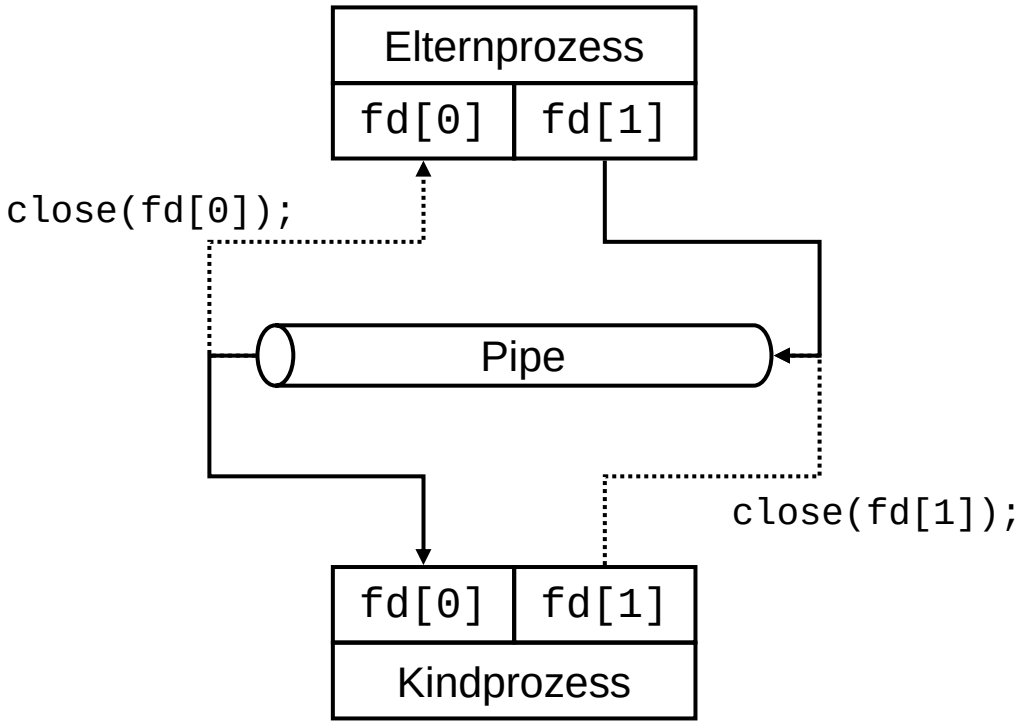
    - `fd[0]`: Filedeskriptor zum Lesen (Leseseite)
    - `fd[1]`: Filedeskriptor zum Schreiben (Schreibseite)
    - Rückgabe bei Erfolg 0, im Fehlerfall -1
  - Schließen eines Pipe-Filedeskriptors:

```
close(fd[n]);
```

  - Kindprozesse erben Pipes des Elternprozesses

Kommunikation über Pipes

- Wiederholung: Namenlose Pipe zur Interprozess-kommunikation (nach `fork()`)



Kommunikation über Pipes

MNM TEAM

- Benannte Pipes (Named Pipes, "FIFOs"):
  - Eigenschaften einer Named Pipe:
    - Unidirektionale Kommunikation
    - Nachrichtenreihenfolge bleibt erhalten (FIFO)
    - Kann ähnlich wie eine normale Datei behandelt werden
    - Kommunizierende Prozesse müssen keine gemeinsamen Vorfahren haben
    - Mehrere Prozesse können in die Pipe schreiben
  - Einrichten einer Named Pipe:

```
mkfifo(pfad, zugriffsrechte);
```

  - Rückgabe bei Erfolg 0, im Fehlerfall -1
  - Öffnen einer Named Pipe:

```
fd = open(pfad, modus);
```

  - Schließen einer Named Pipe:

```
close(fd);
```

Erstellen eines eigenen Parsers

MNM TEAM

- Grundlegende Begriffe
  - Parser/Scanner: Tool zur lexikalischen und syntaktischen Analyse von Zeichenketten gegen eine vorgegebene Sprache/Grammatik
  - Token: kleinste sinngebende Einheit in einer Sprache
  - Vereinfachte Arbeitsweise eines Parsers:
    - Durchsuchen des Eingabe-Datenstroms (z.B. Datei) nach den vorgegebenen Tokens
    - Ausführung einer bestimmten Aktion, wenn ein bestimmtes Token gefunden wurde
- Anwendungsbeispiele:
  - Syntaktische Analyse von Quelldateien einer Programmiersprache (als Teil der Übersetzung)
  - Syntaktische Analyse von Konfigurationsdateien

Erstellen eines eigenen Parsers

MNM TEAM

- Grundlegende Funktionsweise des Tools Flex
  - Input: Spezifikation des Parsers (myparser.lex)
  - Aufruf von Flex:

```
flex -omyparser.c myparser.lex
```

  - Output: Quellcode des spezifizierten Parsers (myparser.c)
  - Aufruf von GCC:

```
gcc -o myparser myparser.c -lfl
```

  - Ergebnis: Parser als ausführbares Programm (myparser)

Erstellen eines eigenen Parsers

MNM TEAM

- Parser-Spezifikation
  - Genereller Aufbau einer Spezifikationsdatei (*.lex):

```
Definitionen
%%
Regeln
%%
Funktionen
```

  - Aufbau einer Regel:

```
Regulärer Ausdruck  {Aktion}
```

- Reguläre Ausdrücke (Regular Expressions)
  - Zeichenketten, die Mengen von Zeichenketten beschreiben
  - Regulärer Ausdruck besteht aus:
    - Terminalzeichen – z.B.: 'a', 'b', 'X', '1', ' ', \n, \t, \b, …
    - Metazeichen: {, }, (,), [,], |, *, +, \, /, ., ^, %, <, >, "

Erstellen eines eigenen Parsers

MNM TEAM

- Besondere Terminalzeichen (Auszug)
  - `\n` Newline
  - `\t` Tabulator
  - `\b` Backspace

- Bedeutung einiger Metazeichen (Auszug)
  - `^` Zeilenanfang
  - `$` Zeilenende
  - `*` beliebig oft
  - `+` beliebig oft, mindestens einmal
  - `?` ein- oder keinmal
  - `.` Beliebiges Terminalzeichen (außer `\n`)
  - `\` Escape-Zeichen

Erstellen eines eigenen Parsers

MNM TEAM

- Beispiele für reguläre Ausdrücke
  - `[123]` eines der Zeichen '1', '2' oder '3'
  - `[a-zA-Z]` ein beliebiger Buchstabe
  - `[1-9]` ein beliebige Ziffer außer '0'
  - `[^1-9]` ein beliebiges Zeichen außer die Ziffern von '1' bis '9'
  - `[A-Z]*` beliebig viele Großbuchstaben
  - `[A-Z]+` ein oder mehrere Großbuchstaben
  - `[A-Z]?` ein oder kein Großbuchstabe
  - `[A-Z]{2,4}` zwei, drei oder vier Grußbuchstaben
  - `a|b` 'a' oder 'b'

Erstellen eines eigenen Parsers

MNM TEAM

- Beispiel: myparser.lex

```
/* Beispiel-Parser */

          int num_lines = 0, num_chars = 0, num_hallo = 0;

%%
\n        {++num_lines; ++num_chars;}
.         {++num_chars;}
Hallo     {++num_hallo;}
%%

int main(int argc, char *argv[]) {
  if(argc != 2) {
    printf("Aufruf: %s filename\n", argv[0]);
    return -1;
  }

  if((yyin = fopen(argv[1], "r")) == NULL) {
    perror("Fehler beim Öffnen der Datei\n");
    return -1;
  }

  yylex();
  printf("Anzahl Zeilen: %d\n", num_lines);
  printf("Anzahl Zeichen: %d\n", num_chars);
  printf("Anzahl Hallos: %d\n", num_hallo);
  return 0;
}
```

- foo.txt

```
Hallo
Welt!

Diese Datei hilft
beim Testen
des
Parsers.
```

Ausführung und Ausgabe:

./myparser foo.txt

```
Anzahl Zeilen: 8
Anzahl Zeichen: 53
Anzahl Hallos: 1
```

Zusammenfassung

MNM TEAM

- Synchrone und asynchrone Kommunikation
- Kommunikationsmuster
- Kommunikationsprotokolle
- Arten der Interprozesskommunikation

- Wiederholung: Namenlose Pipes
- Benannte Pipes (Named Pipes/FIFOs)
- Parser-Generierung mit Flex
- Parser-Spezifikation
- Reguläre Ausdrücke