

Übung zu Betriebssystemtechnik

Aufgabe 6: Nachrichtenaustausch

29. Juni 2023

Bernhard Heinloth, Phillip Raffeck & Dustin Nguyen

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Prozesse sollen über Systemaufrufe miteinander kommunizieren (Nachrichten austauschen) können

Interprozesskommunikation

App 1

App 2

App 1

...
send

App 2

...

send

- versendet eine Nachricht an einen Empfangsprozess



send

- versendet eine Nachricht an einen Empfangsprozess
- erwartet den Nachrichteneingang



send

- versendet eine Nachricht an einen Empfangsprozess
- erwartet den Nachrichteneingang

receive

- empfängt eine Nachricht von einem Sendeprozess
- erwartet den Nachrichteneingang



Nachrichtenaustausch (mit Antwort)



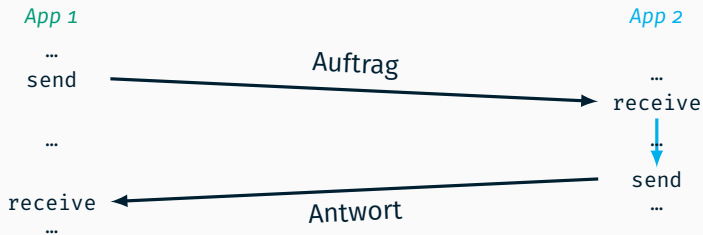
Nachrichtenaustausch (mit Antwort)

- ein Prozess agiert als Sender, der andere als Empfänger



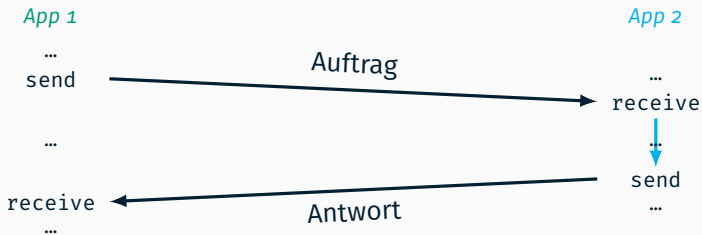
Nachrichtenaustausch (mit Antwort)

- ein Prozess agiert als Sender, der andere als Empfänger
- die Prozesse tauschen dann ihre Rollen



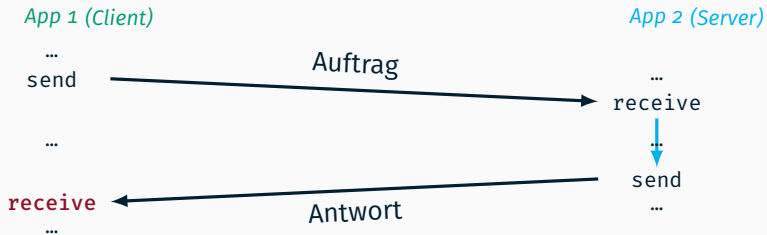
Nachrichtenaustausch (mit Antwort)

- ein Prozess agiert als Sender, der andere als Empfänger
- die Prozesse tauschen dann ihre Rollen



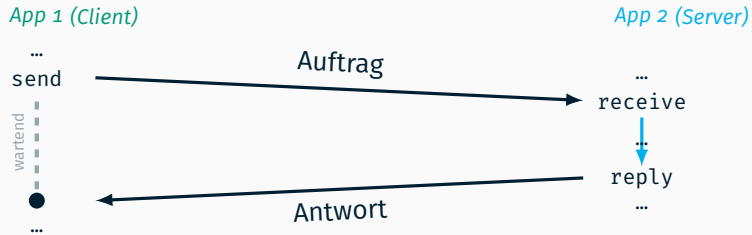
Nachrichtenaustausch (mit Antwort)

- ein Prozess agiert als Sender, der andere als Empfänger
- die Prozesse tauschen dann ihre Rollen
- aber **Verklemmungsgefahr**, falls die Prozessrollen zum Kommunikationszeitpunkt gleich sind

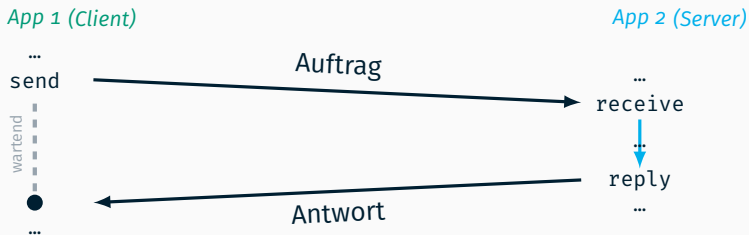


Nachrichtenaustausch (mit Antwort, z.B. Client – Server)

- ein Prozess agiert als Sender, der andere als Empfänger
- die Prozesse tauschen dann ihre Rollen
- aber **Verklemmungsgefahr**, falls die Prozessrollen zum Kommunikationszeitpunkt gleich sind
- **App 2 (Server)** muss **App 1 (Client)** vertrauen (*Denial of Service!*)



Alternative mit vorgegebener Hierarchie bei Kommunikation

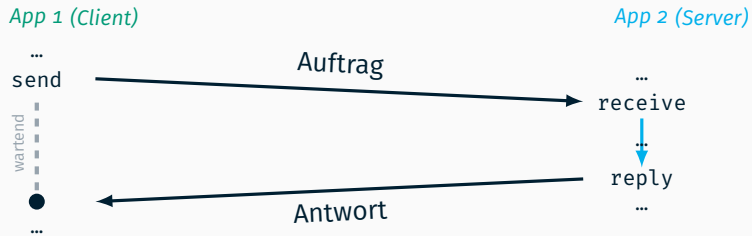


Alternative mit vorgegebener Hierarchie bei Kommunikation

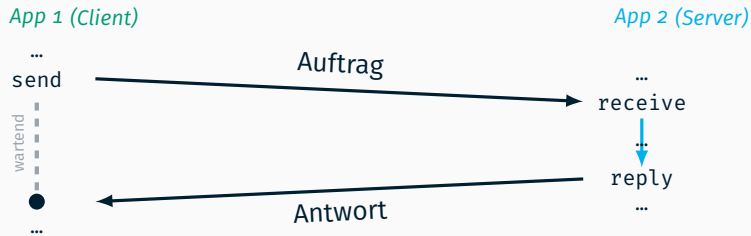
send versendet einen Auftrag an einen Serverprozess und erwartet die Auftragsbeantwortung

receive erwartet/empfängt einen Auftrag von einem Clientprozess

reply versendet eine Antwort an den beauftragenden Clientprozess



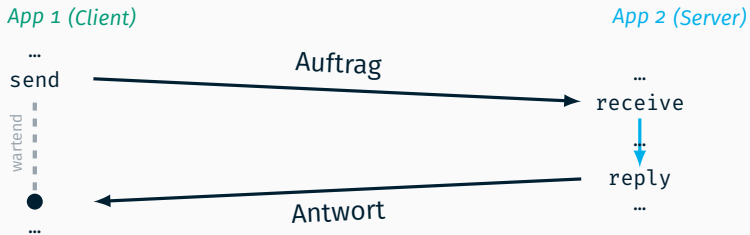
Umsetzung synchron und blockierend



Umsetzung synchron und blockierend

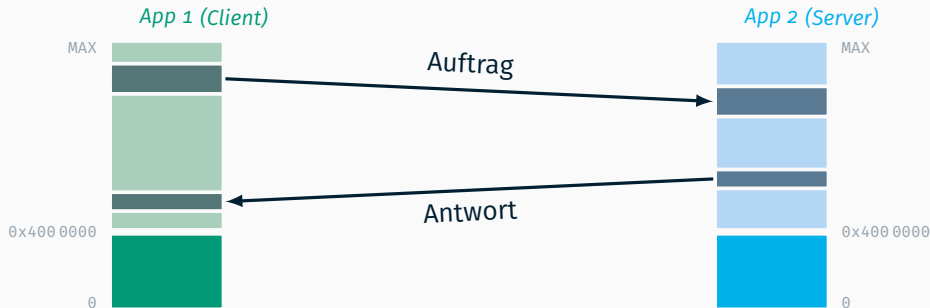
Begriffsklärung:

synchron	Prozess schläft bis IPC-Operation fertig
asynchron	Prozess beginnt IPC-Operation und kehrt zurück
blockierend	wartet bei nicht verfügbaren Betriebsmittel
nichtblockierend	schlägt bei nicht verfügbaren Betriebsmittel fehl



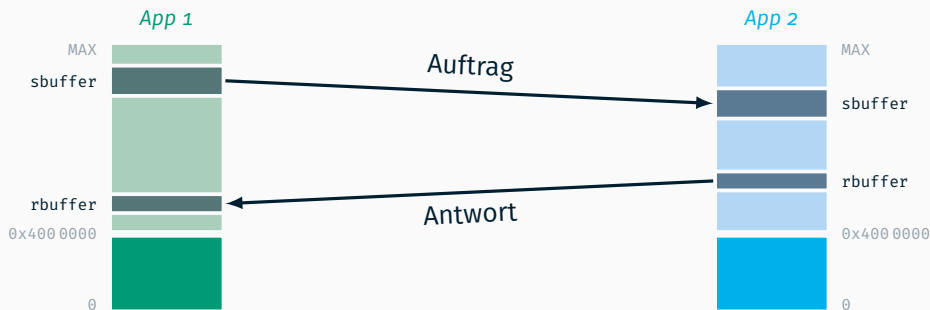
Umsetzung synchron und blockierend

- der Client wartet (passiv) im `send` auf das `receive` des Servers
- der Server wartet (passiv) im `receive` auf das `send` des Clients
- der Datentransfer erfolgt beim Rendezvous von Server und Client



Umsetzung synchron und blockierend

- der Client wartet (passiv) im `send` auf das `receive` des Servers
- der Server wartet (passiv) im `receive` auf das `send` des Clients
- der Datentransfer erfolgt beim Rendezvous von Server und Client
 - Ende-zu-Ende, ohne modellbedingte Zwischenpufferung der Nachricht
 - direkt zwischen den Adressräumen der beiden involvierten Prozesse



Umsetzung synchron und blockierend mittels

- `void send(int pid, const void *sbuffer, size_t ssize, void *rbuffer, size_t rsize);`
- `int recv(void *sbuffer, size_t ssize);`
- `void reply(int pid, const void *rbuffer, size_t rsize);`

Implementierung

Threads müssen für die Interprozesskommunikation

- eindeutig identifizierbar sein

`pid` aus Prozessverwaltung (von *Aufgabe 5*)

Threads müssen für die Interprozesskommunikation

- eindeutig identifizierbar sein

`pid` aus Prozessverwaltung (von *Aufgabe 5*)

- Nachrichten verwalten (Queues)

`msg_inbox` (unbearbeitete) Aufträge

`msg_reply` ausstehende Antworten (zu empfangenen Aufträgen)

Threads müssen für die Interprozesskommunikation

- eindeutig identifizierbar sein

`pid` aus Prozessverwaltung (von *Aufgabe 5*)

- Nachrichten verwalten (Queues)

`msg_inbox` (unbearbeitete) Aufträge

`msg_reply` ausstehende Antworten (zu empfangenen Aufträgen)

- blockieren und signalisieren

`sem_recv` Empfang einer neuen Nachricht (Auftrag)

`sem_send` abgeschlossene Bearbeitung einer Nachricht
(Antwort vorhanden)

Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```

Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```

Existiert pid?

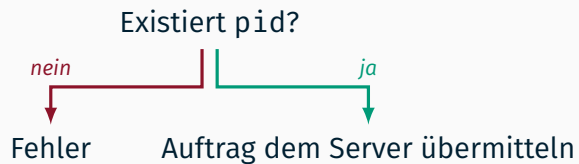
Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



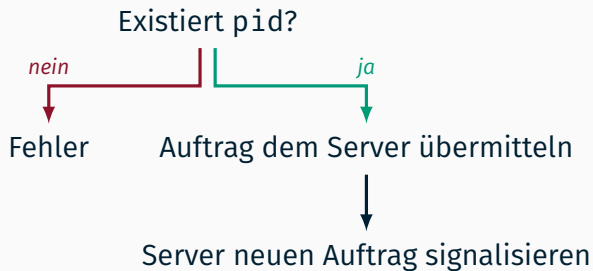
Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



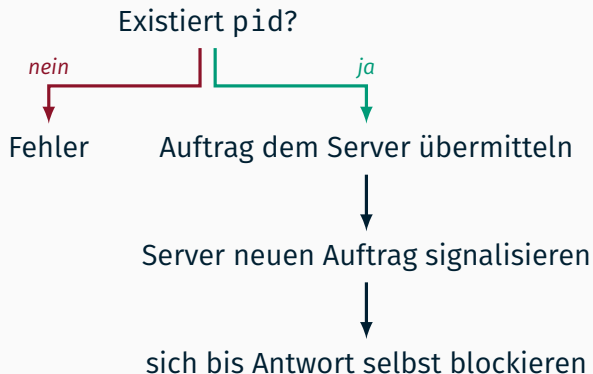
Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



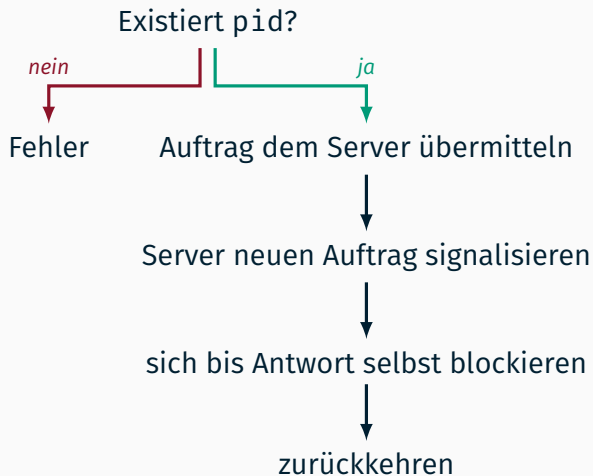
Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



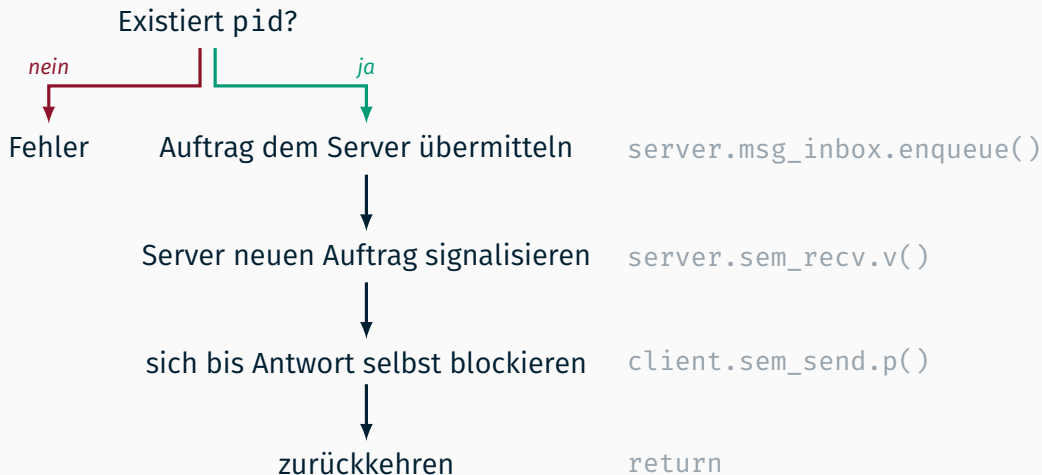
Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



Ablauf Sendeoperation

```
void send(int pid, const void *sbuffer, size_t ssize,  
          void *rbuffer, size_t rsize);
```



Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

auf neuen Auftrag warten

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

auf neuen Auftrag warten



Auftrag des Clients entgegennehmen

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

auf neuen Auftrag warten



Auftrag des Clients entgegennehmen



(Client-)Puffer zu Server kopieren

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

auf neuen Auftrag warten



Auftrag des Clients entgegennehmen



(Client-)Puffer zu Server kopieren



ausstehende Antwort merken

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```

auf neuen Auftrag warten



Auftrag des Clients entgegennehmen



(Client-)Puffer zu Server kopieren



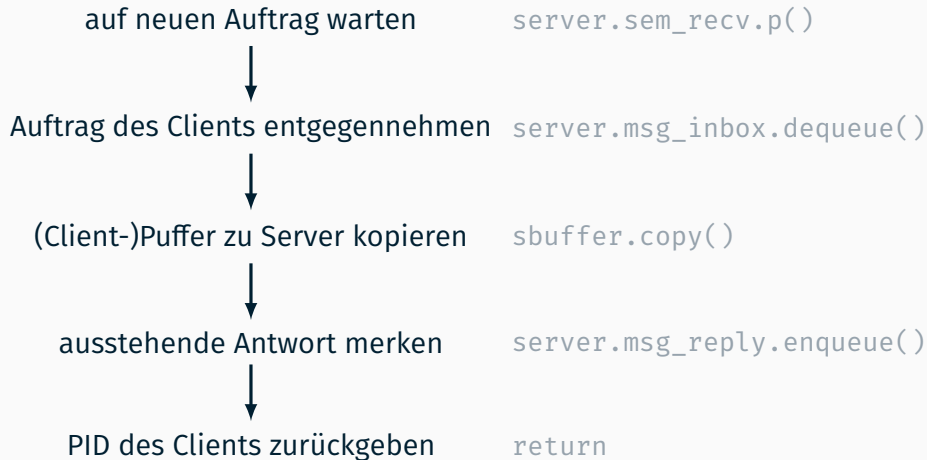
ausstehende Antwort merken



PID des Clients zurückgeben

Ablauf Empfangsoperation

```
int recv(void *sbuffer, size_t ssize);
```



Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```


Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

Gab es einen Auftrag von pid?

Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

Gab es einen Auftrag von pid?

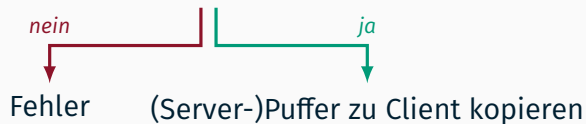


Fehler

Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

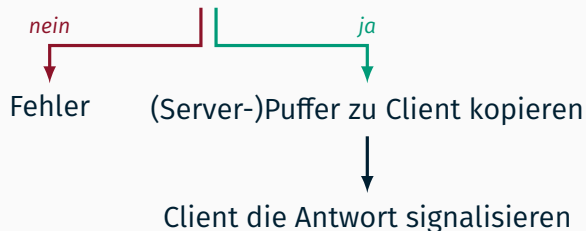
Gab es einen Auftrag von pid?



Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

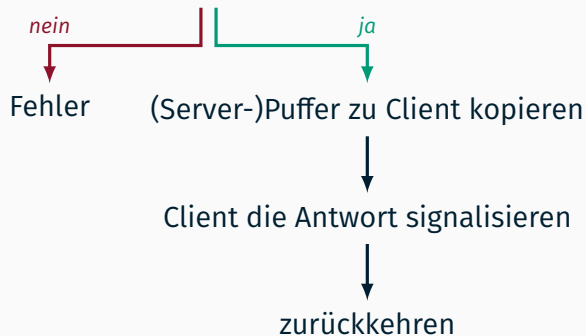
Gab es einen Auftrag von pid?



Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

Gab es einen Auftrag von pid?



Ablauf Antwortoperation

```
void reply(int pid, const void *rbuffer, size_t rsize);
```

Gab es einen Auftrag von pid?

```
server.msg_reply.contains()
```

nein

ja

Fehler

(Server-)Puffer zu Client kopieren

```
rbuffer.copy()
```

Client die Antwort signalisieren

```
client.sem_send.v()
```

zurückkehren

```
return
```

Testen von Aufgabe (5 &) 6

```
char sbuf[8194], rbuf[8194];
void main() {
    fork();
    fork();
    if (fork() == 0) {
        sbuf[0] = 3;
        char d = sbuf[8192] = getpid() % 22;
        sbuf[8193] = 1;
        send(getpid(), sbuf, 8193, rbuf, 8193);
        char m[] = "Reply_A=A_bad!\ngood";
        m[6] += rbuf[0] + sbuf[8193];
        m[8] += 4 + d;
        for(size_t i = 0; m[6] == m[8] && i < 4; i++) m[i + 10] = m[i + 15];
        write(0, m, 15);
    } else {
        int X = recv(rbuf, 8193);
        rbuf[0] = rbuf[0] + rbuf[8192];
        rbuf[8193] = 7;
        reply(X, rbuf, 8193);
    }
    exit();
}
```

Fragen?

Nächste Woche folgt die Tafelübung zur letzten Aufgabe