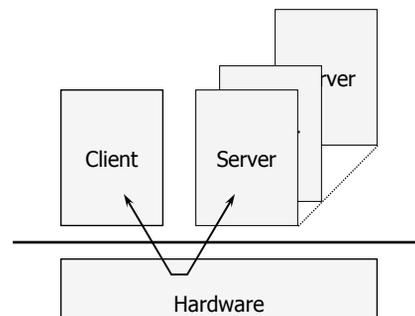


# Verteilte Systeme

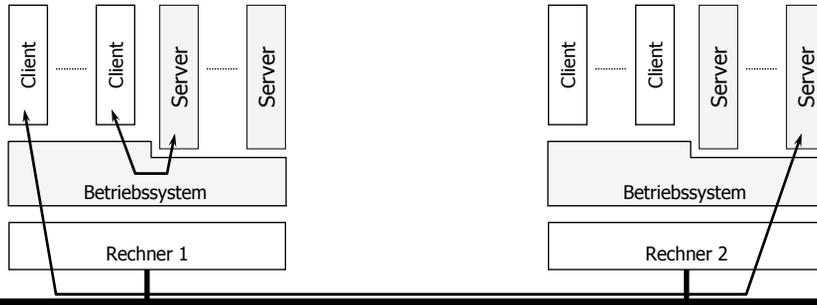
## 12. Client/Server-Systeme

### Rückblick: Client/Server auf einem Rechner

- Verbreitete Systemarchitektur
- Verlagerung von Kernfunktionen in Server-Prozesse
  - Mikrokern
  - Server im User-Modus
- Vorteile
  - Entwicklungs- und Testmethoden für Anwendungen auf Server übertragbar
  - Erweiterbarkeit, Austauschbarkeit
- Nachteile
  - Zusätzliche Kontextwechsel
  - Weitergabe von Hardware-Ereignissen (Interrupts)
  - Zugriff auf Gerätereister



### Client/Server-Architektur



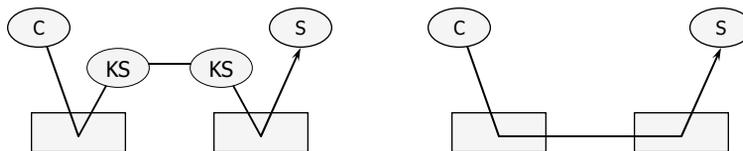
- Lokale Server
  - Dateisystem
  - Ein- und Ausgabedienste (Graphik, Kommunikation, ...)
  - Prozessverwaltung, Paging, ...

- Entfernte Server
  - Netzwerkdateisystem, z.B. NFS
  - Verteiltes Dateisystem
  - Rlogin, Telnet, ...
  - WWW-Server (httpd)
  - Uhrensynchronisation (ntp)

Verteilte Systeme, Winter 2002/03

Folie 12.3

### Realisierungsvarianten



- Kommunikationsserver
- Vorteile
  - Erweiterbarkeit
  - Einfacher Kern
- Nachteile
  - Zusätzliche Kontextwechsel
  - Höhere Nachrichtenlaufzeit

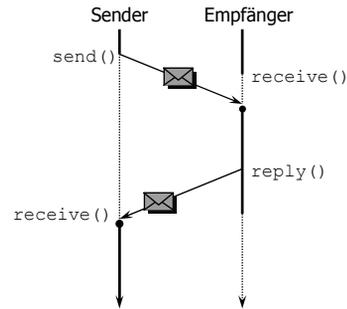
- Kernkommunikation
- Vorteile
  - Minimale Verzögerungen
  - Schnelle Übertragung
- Nachteile
  - Kernänderung bei Erweiterungen
  - Komplexer Kern
  - Routing, ...

Verteilte Systeme, Winter 2002/03

Folie 12.4

## Kommunikation in Client/Server-Architekturen

- SRSI „sehr schöne“ Abstraktion
  - `send` und `receive` nicht explizit sichtbar
  - Kapselung der Dienste in eine Art Prozeduraufruf
- Remote Procedure Call (RPC)
  - „Der“ SRSI-Vertreter
  - Idee: RPC = Lokaler Prozeduraufruf
  - Birrell und Nelson, 1984
- Grundlage für Client/Server-Systeme
  - Server: Passiv, bietet Dienste an
  - Client: Aktiv, ruft Dienste auf
- Alternative: Asynchroner entfernter Dienstaufuf

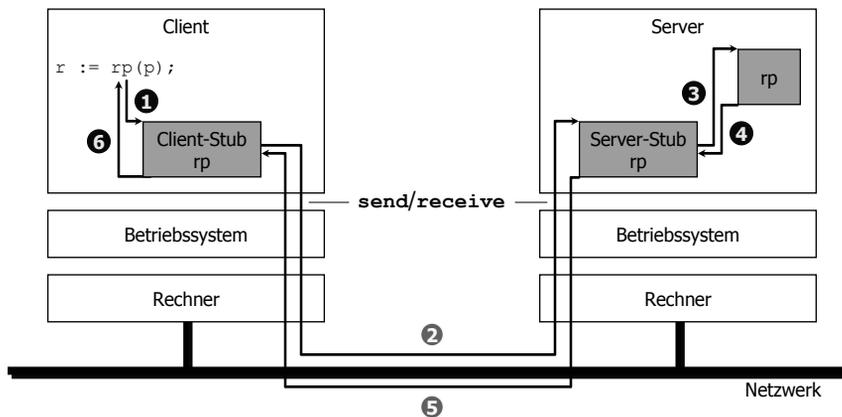
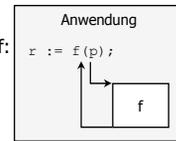


Verteilte Systeme, Winter 2002/03

Folie 12.5

## RPC-Architektur

Lokaler Prozeduraufruf:



Verteilte Systeme, Winter 2002/03

Folie 12.6

## Semantiken

- Ideale Semantik: Exactly-Once-Semantik
  - Entfernter Dienstaufwurf findet genau einmal statt
- Realität: Fehlerfälle
  - Verlust von Nachrichten
  - Server-Abstürze
- Realisierbare Semantiken
  - At-Least-Once-Semantik
    - Dienstaufwurf findet mindestens einmal statt
    - Aufträge können mehrfach ausgeführt werden
  - At-Most-Once-Semantik
    - Dienstaufwurf findet höchstens einmal statt
    - Aufträge dürfen nicht wiederholt werden

Verteilte Systeme, Winter 2002/03

Folie 12.7

## Umsetzung

### At-Least-Once

- Einfacher Server
 

```
while (1) {
  receive(job) from client;
  // Auftrag ausführen
  send(reply) to client;
}
```
- Mehrfachausführung in vielen Fällen unkritisch
  - Uhrzeit abfragen
  - Datei lesen?
- Idempotenz wünschenswert

### At-Most-Once

- Komplexer Server
 

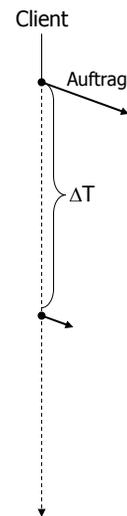
```
while (1) {
  receive(job) from client;
  if (neuer Auftrag) {
    // Auftrag ausführen
  }
  send(reply) to client;
}
```
- Wie erkennt man neuen Auftrag?
  - Zustand

Verteilte Systeme, Winter 2002/03

Folie 12.8

## Fehlerfälle im Detail

- Annahme: Alle zu einem Auftrag gehörenden Protokollnachrichten sind identifizierbar
  - Z.B. fortlaufende Auftragsnummer
- Client sendet Auftrag k und erhält längere Zeit kein Reply
- Situation beim Timeout
  - Auftragsnachricht ging verloren
  - Reply-Nachricht ging verloren
  - Reply unterwegs
  - Server vor Auftragsbearbeitung abgestürzt
  - Server während der Auftragsbearbeitung abgestürzt
  - Server bearbeitet noch Auftrag
  - ...
- In allen Fällen Nachrichtenwiederholung sinnvoll
- ... und wie geht es weiter?



Verteilte Systeme, Winter 2002/03

Folie 12.9

## Server-Abstürze

- Schnellstmöglicher Ersatz des ausgefallenen Servers
  - Election unter den Ersatzservern
  - Neustart eines Ersatzservers
- Zustandsverluste
  - Partiieller oder vollständiger Gedächtnisverlust
  - Amnesia Failures
- At-Least-Once
  - Unkritisch
- At-Most-Once
  - Ersatzserver: Wurde der Auftrag schon ausgeführt?
  - Unter Umständen aufwendiges Loggen

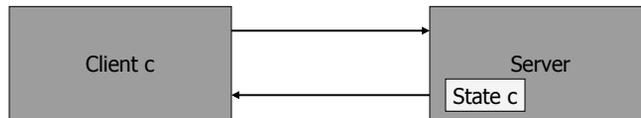


Verteilte Systeme, Winter 2002/03

Folie 12.10

## Zustandsbehafteter Server

- Relevanter Serverzustand
  - Zustand für korrekte Dienstbringung notwendig
    - Zustand der Clientverbindung
    - Information über kausal abhängige Auftragsfolgen
    - Liste der ausgeführten Aufträge (At-Most-Once)
    - ...
  - Nicht gemeint sind
    - lokale Variablen u.ä.
    - Dateien eines Dateiservers
- Primär leistungssteigernde Maßnahme

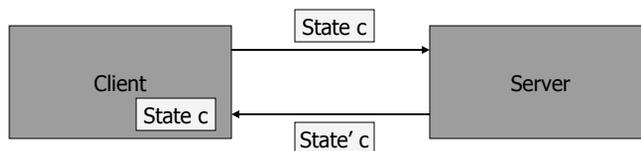


Verteilte Systeme, Winter 2002/03

Folie 12.11

## Zustandslose Server

- Umgehen der Zustandsproblematik
- Verlagerung des relevanten Zustands in den Client
  - Client speichert Zustandsinformation
  - Notwendige Zustandsinformation ist Teil des Auftrags
  - Reply aktualisiert auch client-seitigen Zustand
- Nachteile
  - Größere Nachrichten



Verteilte Systeme, Winter 2002/03

Folie 12.12

## Beispiel: Dateiserver

- Ausgangspunkt:  
Zustandsbehafteter Server
- Elemente des Serverzustands
  - Dateideskriptor (explizit)
  - Aktuelle Position innerhalb der Datei (implizit)
  - Dateipuffer (implizit)
  - ...
- Auswirkungen der Semantik
  - At-Most-Once?
  - At-Least-Once?
- Server-Absturz?

```
Client:
  fd = send(OPEN,name,READ)
      to server;
  data = send(READ,fd) to server;
  ...
  Send(CLOSE) to server;
```

```
Server:
  receive(OPEN,name,mode) from client {
    fd = open(name,mode,...);
    reply(fd) to client;
  }

  receive(READ,fd) from client {
    data = read(fd,...);
    reply(data) to client;
  }
```

Verteilte Systeme, Winter 2002/03

Folie 12.13

## Verlagerung des Serverzustands

- Kein serverseitiger Deskriptor
  - Übermittlung des symbolischen Dateinamens in jedem Auftrag
  - Speicherung der Leseposition auf Clientseite
- Öffnen und Schließen einer Datei erübrigt sich
- Auswirkungen auf Semantik
  - At-Most-Once?
  - At-Least-Once?
- Server-Absturz
- Weitere Probleme
  - Sperren von Dateien?
  - Performanz

```
Client:
  data = send(READ,name,pos)
      to server;
  ...
```

```
Server:
  receive(READ,name,mode) from client {
    fd = open(name,mode,...);
    lseek(fd,pos,...);
    data = read(fd,...);
    reply(fd) to client;
    close(fd);
  }
```

Verteilte Systeme, Winter 2002/03

Folie 12.14

## Unkritischer Serverzustand

- „Caching“ von Serverzustand auch in einem zustandslosen Server möglich
- Vollständiger Zustand aus Auftrag wieder herleitbar
- Gelegentliches Aufräumen

```
Client:
  data = send(READ,name,pos)
        to server;
  ...
```

```
Server:
  receive(READ,name,mode) from client {
    if ((name,client) in cache)
      fd = cached_fd(name,client);
    else {
      fd = open(name,mode,...);
      lseek(fd,pos,...);
    }
    data = read(fd,...);
    reply(fd) to client;
    close(fd);
  }
```

Verteilte Systeme, Winter 2002/03

Folie 12.15

## Idempotente Aufträge

- Wiederholung desselben Auftrags liefert dasselbe Ergebnis
- Viele Aufträge inhärent idempotent
- Lösung bei nicht-idempotenten Aufträgen
  - Aufträge enthalten alle notwendige Zustandsinformation
  - Zustandsloser Server
- Rätselcke:
  - Gibt es inhärent nicht-idempotente Aufträge?
  - Wie werden diese trotzdem idempotent?

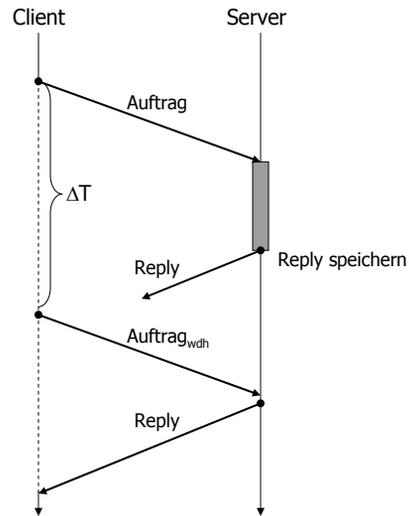


Verteilte Systeme, Winter 2002/03

Folie 12.16

## Lösung

- Speicherung fertiggestellter Aufträge
- Bei Wiederholungsauftrag gespeichertes Reply erneut zurücksenden
- Wann kann man gespeicherte Replies löschen?
- Dilemma
  - Zustandsbehafteter Server unumgänglich



Verteilte Systeme, Winter 2002/03

Folie 12.17

## Server-Realisierungen

- Iterativer Server
  - Maximal ein Auftrag in Bearbeitung
  - Vorteile
    - Einfache Realisierung
  - Nachteile
    - Unter Umständen lange Antwortzeiten
    - Reihenfolge der Bearbeitung liegt fest
- Konkurrenter Server
  - Mehrere Aufträge gleichzeitig in Bearbeitung
  - Vorteile
    - Leistungssteigerung (Multiprozessor, Auftragsreihenfolge, Blockadezeiten)
  - Nachteile
    - Komplexere Realisierung
    - Synchronisation

Verteilte Systeme, Winter 2002/03

Folie 12.18

## Realisierungsvarianten eines konkurrenten Servers

- ☐ Multiplex-Server
  - Server unterteilt größere Aufträge selbst in kleinere Einheiten
  - Explizite Verwaltung der Bearbeitungszustände
  - Problematik: Blockierende Systemaufrufe
- ☐ Multithreaded Server
  - Leichtgewichtige Threads in einem Adreßraum
  - Kommunikation über gemeinsamen Speicher
  - Speicherbasierte Synchronisation
- ☐ Multi-Process Server
  - Mehrere Prozesse
  - Typische Organisationsform: Manager und Worker-Pool
  - Kommunikation über Nachrichten
  - Verteilte Synchronisationsverfahren

Verteilte Systeme, Winter 2002/03

Folie 12.19

## Neuerzeugen oder Wiederverwenden

- ☐ Betrifft Worker in einem konkurrenten Server
- ☐ Neuerzeugung eines "Workers"
  - Vorteile
    - ☐ Gute Ressourcenbelegung
    - ☐ Einfache Implementierung
  - Nachteile
    - ☐ Höhere Latenz
- ☐ Wiederverwendung der beendeten "Worker-Hülsen"
  - Vorteile
    - ☐ Geringere Latenz
  - Nachteile
    - ☐ Belegung von Systemressourcen
    - ☐ Komplexe Implementierung

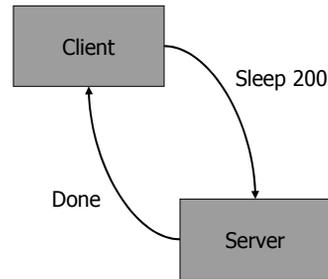


Verteilte Systeme, Winter 2002/03

Folie 12.20

## Beispiel: Der Schlafserver

- Auswirkungen der Servervarianten untersuchen
  - Kommunikationskosten nicht meßrelevant
  - Einfache TCP-Realisierung
- Server "schläft" für die angegebene Zeit
- Vergleichbar einer E/A-Blockade
  - Block lesen und schreiben
  - Netzkommunikation (Auftrag an weiteren Server)
  - Synchronisation



Verteilte Systeme, Winter 2002/03

Folie 12.21

## Der Schlafclient

```

□ int main ( int ac, char ** av ) {
    TCP_Socket sock;
    Server_Address saddr;

    host = av[1]; port = av[2];
    loops = av[3]; delay = av[4];
    saddr = Server_Address (host,port);
    sock.connect (saddr);
    watch.start();
    for (int i=0;i<loops;i++) {
        sprintf(buf,"sleep %d",delay);
        sock.send(buf,strlen(buf));
        sock.receive(buf,sizeof buf);
        assert(strcmp(buf,"done") == 0);
    }
    watch.stop();
    average_delay = watch.msecs() / loops;
}
  
```

Verteilte Systeme, Winter 2002/03

Folie 12.22

## 1. Iterativer Server (Single-Haushalt)

```
q int main ( int ac, char **av ) {
    TCP_Socket asock;
    TCP_Socket *csock;

    asock.bind();
    printf("Sleeping at port %d\n", asock.port());
    while (1) {
        csock = asock.accept();
        csock->receive(buf, sizeof buf);
        sscanf(buf, "sleep %d", &delay);
        if (delay>0) sleep(delay);
        sprintf(buf, "done");
        csock->send(buf, strlen(buf));
        delete csock;
    }
}
```

Verteilte Systeme, Winter 2002/03

Folie 12.23

## 2. Multithreaded Server (WG)

q Übung

Verteilte Systeme, Winter 2002/03

Folie 12.24

### 3. Multi-Process Server (Hotel)

```

g int main ( int ac, char **av ) {
    TCP_Socket asock;
    TCP_Socket *csock;

    asock.bind();
    printf("Sleeping at port %d\n", asock.port());
    while (1) {
        csock = asock.accept();
        if (fork() == 0) {
            csock->receive(buf, sizeof buf);
            sscanf(buf, "sleep %d", &delay);
            if (delay > 0) sleep(delay);
            sprintf(buf, "done");
            csock->send(buf, strlen(buf));
            exit();
        }
        delete csock;
    }
}

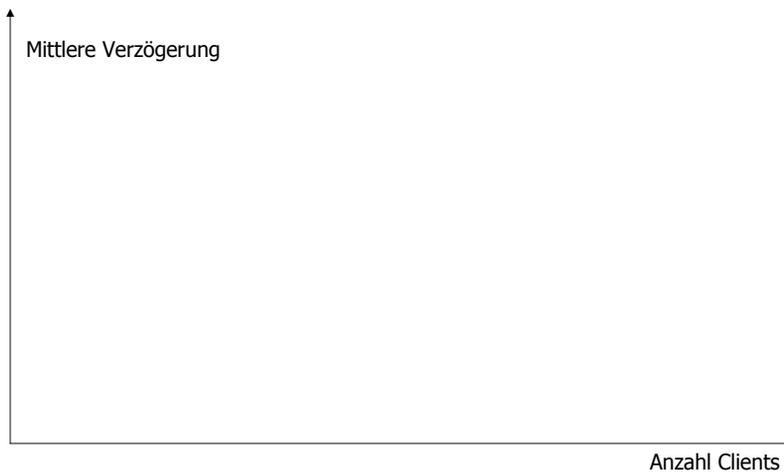
```

Verteilte Systeme, Winter 2002/03

Folie 12.25

### Geschätzte mittlere Verzögerung

Multi-Process Server ———  
 Multithreaded Server ———  
 Iterativer Server ———



Verteilte Systeme, Winter 2002/03

Folie 12.26

### Gemessene mittlere Verzögerung

