30 Jahre Server

Von Transaktionssystemen zu Web-Services

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Lehrstuhl für Informatik 6 (Datenbanksysteme)

Prof. Dr. Klaus Meyer-Wegener

Anlass

- "Java (EJB, ….) ist ja so langsam!"
- "Aber CICS ist inzwischen ganz schön schnell!"

- "Was ist CICS?"

Server: Erscheinungsformen

- Transaktionssystem ("OLTP")
- Datei-Server (NFS, ….)
- RPC-Server ("Client-Server-Betrieb")
- Datenbank-Server (SQL, Sybase)
- Objekt-Server (CORBA)
- Web-Server
- Application Server
- EJB Container
- Web Services
- ….

Server: verallgemeinert

- Auftrag – Berechnung – Antwort (Ergebnis)
  - Request - Response
- verschiedenste Protokolle:
  - Fernaufruf (Remote Procedure Call, RPC)
  - HTTP
  - RMI
  - ….
- verbindungsorientiert oder verbindungslos
- ein wichtiges zusätzliches Konzept: "Session"
  - Zusammenhang zwischen aufeinander folgenden Aufträgen
  - Server muss Kontext verwalten und benutzen
  - Berechnung meldet mit Antwort: Session geht weiter oder nicht

Aufgabe

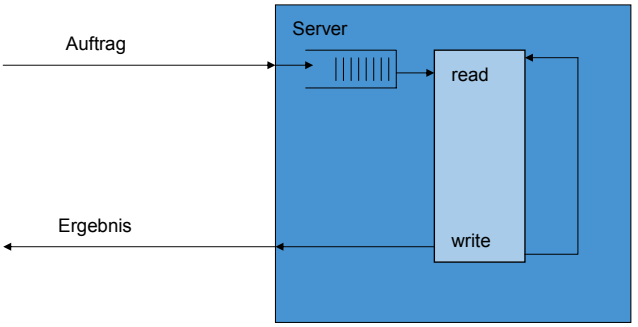
- **Wie macht man den Server effizient?**
  - Tausende von Anfragen
  - Tausende von Kontexten
- **Prozess-Strukturen im Betriebssystem**
  - Verwaltungsaufwand?
- **Programmierung der Anwendungen**
  - so einfach wie möglich
- **zusätzliche Programme ("Middleware")**
  - Transaktionsmonitore
  - Container
  - Dispatcher
  - ….

Server-Programmierung

- **Wie schreibt man die Programme, die**
  - einen Auftrag entgegennehmen,
  - die Berechnung durchführen und
  - die Antwort mit dem Ergebnis zurücksenden?
- **Zugeständnisse an die Effizienz**
  - Kontext anders behandeln als lokale Variablen
  - mehrere kooperierende Programme schreiben für die eine Aufgabe
  - Mehrfachausführung des Programms berücksichtigen ("Threads" erzeugen)
  - Synchronisation von Zugriffen auf gemeinsame Daten
- **oder nichts von alledem?**
  - wenn die Middleware das alles übernimmt ….

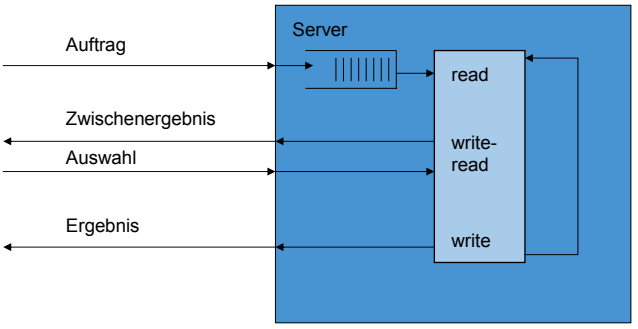
Server-Implementierung

- Wie führt man die Programme aus?
- einfachste Lösung:



Server-Implementierung (2)

- Berücksichtigung des Kontexts:



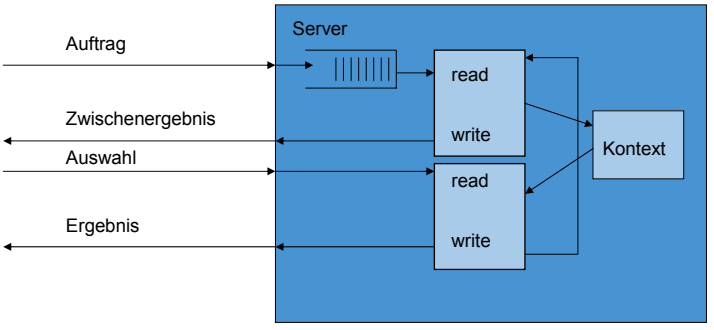
Server-Implementierung (3)

- **einfachste Art der Programmierung:**
  - immer nur ein Auftrag zur Zeit
  - Kontext direkt im Programm
- **ineffizient!**
  - Wartezeit auf nächste Eingabe ("Auswahl") ungenutzt
  - Wartezeiten während der Bearbeitung
    - Ein-/Ausgabe
    - Aufträge an andere Server (!)

    ebenfalls ungenutzt

Kontext separieren

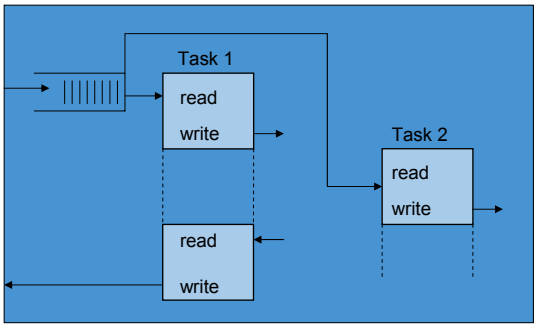
- **explizite Verwaltung**



- Programmierung wird aufwändiger

Multi-Tasking

- **Wartezeiten in der Berechnung nutzen:**



  - Wer programmiert das?

Prozesse

- **soweit ein Prozess pro Server**
  - überschaubar
  - komplexe interne Programmstrukturen
  - braucht hohe Priorität
  - bei Seitenfehler oder Zeitscheibenende wird der gesamte Server inaktiv
- **ein Prozess pro Auftrag?**
  - CGI ….
  - Verwaltungsaufwand zu hoch
- **mehrere Prozesse pro Server (10 – 100)**
  - skalierbar
  - Prozesskommunikation: Kontexte müssen "wandern"
  - interne Programmstrukturen einfach
  - Verwaltungsaufwand mittelmäßig

Threads

- **Ablaufeinheit innerhalb eines Prozesses**
  - Verwaltung deutlich einfacher als beim Prozess
  - Erzeugen und Vernichten oder Pool bereithalten
- **Scheduling zweiter Stufe**
  - Betriebssysteme oder Subsystem, z.B. TP-Monitor oder DBVS
  - Umschaltung bei Ein-/Ausgabe und Aufruf anderer Server, evtl. auch bei Seitenfehler, aber nicht bei Zeitscheibenablauf
- **gleicher Adressraum**
  - effizienter Datenaustausch (Kontexte)
  - Synchronisation in vielen Fällen durch Scheduling
- **bessere Bezeichnung wäre: Task**

Programmverwaltung

- **mehrere Tasks führen das gleiche Programm aus**
  - jeder erzeugt einen "Thread" durch ein Programm
- **einfachste Variante:**
  - jeder erhält eigene Kopie ("Single-Threading")
  - hoher Speicherplatzbedarf (Seitenwechselrate steigt)
- **besser:**
  - **gemeinsame Kopie** benutzen ("Multi-Threading")
  - geht nur unter bestimmten Voraussetzungen:
    - alle Variablen im Kontext, Trennung von Daten- und Code-Segment
    - keine Code-Modifikation
  - Aufgabe des Compilers

Namen

- **Server**
  - n:m-Beziehung mit Netzknoten
  - ein Server auf mehreren Netzknoten bedeutet: austauschbar
  - URL
- **Funktion**
  - n:1-Beziehung mit Servern
- **Programm**
  - n:m-Beziehung mit Funktionen
  - extern nicht sichtbar
  - interne Zuordnung von Programmen zu Funktionen oder auch nur Teilen (Abschnitten) davon
- **zahlreiche Dienste verfügbar**
  - X.500, LDAP, Jini, UDDI, ….

Transaktionen

- **Fehler und Ausfälle berücksichtigen**
  - Berechnung im Server begonnen, aber noch nicht vollständig
  - Client kann diesen Zwischenzustand weder verstehen noch verlassen
  - manueller Eingriff durch Server-Administration?
- **"Transaktion" als Dienstleistung des Servers (und der Middleware)**
  - Alles oder nichts – vom System gewährleistet:
    - unvollständige Berechnungen rückgängig machen, so dass der Client den Auftrag wiederholen kann
    - vollständige Berechnungen wiederherstellen
  - nicht der Programmierung überlassen, sondern in der Middleware anbieten

Ziel

- **bei Einsatz aller genannten Techniken zur Steigerung der Effizienz**
  - Programmverkettung
  - Multi-Tasking
  - Multi-Threading
  - ablaufinvariante Programme
- **Programmierung dennoch so einfach wie möglich**
  - *einen* Auftrag bearbeiten
  - Kontext = lokale Variablen
  - Isolierung: gleichzeitig laufende Aktivitäten nicht sichtbar
  - Ablaufinvarianz des Codes ist Sache des Compilers
- **Aufgabe für Middleware**
  - insbesondere: Strategien änderbar (optimierbar)

Allgemeine Server-Strukturen und -Techniken

- **einsetzbar für:**
  - Transaktionssysteme (Flugbuchung, Kontobuchung, SAP)
  - Datei-Server
  - RPC-Server ("Client-Server-Betrieb")
  - Datenbank-Server (SQL, Sybase)
  - Web-Server
  - Application Server
  - EJB Container
  - Web Services
  - ….
- **und wird dort (teilweise) eingesetzt**
- **Bedarf an Leistungssteigerung ungebrochen**
- **trotzdem Trennung Anwendungslogik - Infrastruktur**