J 10. Übung

In dieser Aufgabe soll das in der Vorlesung und in den vergangenen Übungen erworbene Wissen von Ihnen in einen Gesamtkontex gestellt werden.

- Sie haben in der Vorlesung kennengelernt, welche Probleme bei der Verwendung des Fernaufrufs auftreten und welche Lösungsansätze existieren. Stellen Sie sich vor, sie werden vor die Aufgabe gestellt, ein RPC-System zu entwerfen und zu implementieren. Diskutieren Sie die zu treffenden Entwurfsentscheidungen in Abhängigkeit von bestimmten Systemgegebenheiten und Anforderungen.

Diese Übung erfordert **IHRE aktive Teilnahme**!

J.1 Problemfelder

- Allgemeine Aspekte
- Unterstützte Programmiersprachen
- Notwendigkeit einer IDL
- Aufrufsemantik
- Orphan-Behandlung
- Transportprotokoll
- Verteilte Adressierung
- Ausführungsschemata
- Transparenz des RPC-Systems
- Sonstiges

J.2 Allgemeine Aspekte

- Welche Fehler sollen tolerierbar sein?
  - Kommunikationsfehler: Nachrichtenverlust, (-verdoppelung, -reihenfolge); Netzwerkpartition; Verfälschung von Nachrichten
  - Rechnerausfälle: Dauer, wie viele gleichzeitig, wer (Client/Server)?
- Effizienz
  - Abwägung: Zeitaufwand für die Implementierung vs. Effizienz, Zuverlässigkeit
- Betriebssystem-/Ablaufsystem-Support
  - Unterstützung beim Recovery
  - Bereitstellung von Koordinierungsmechanismen

J.3 Unterstützte Programmiersprachen

- OO-Programmiersprachen?
  - Vorteil Kapselung (kein direkter Zugriff auf Daten); keine globalen Daten;
  - Objekte können sich für Call-by-Value selbst serialisieren (Marshalling); einfach bei Unterstützung durch das Typsystem z.B. in Java, schwieriger in C++
  - Garbage Collection (falls vorhanden) vereinfacht dynamische Speicherverwaltung erheblich!
- Homogen/Heterogen
  - Homogen manches einfacher (z.B. Java RMI: keine IDL notwendig), weniger zu lösende Probleme

J.4 Notwendigkeit einer IDL

- ◆ Nicht erforderlich in homogenen Systemen, falls Typsystem der Sprache bereits hinreichend mächtig ist
  - Schnittstellenbeschreibung in Programmiersprache selbst
- ◆ Ansonsten erforderlich
  - Festlegung des genauen Datentypes
    (z.B. ist `"char *"` Zeichen, String oder Array?)
  - Art der Parameterübergabe
    (in/out, by value oder by reference)

J.5 Aufrufsemantik

- Welche Semantik brauche ich überhaupt wann?
  - ◆ Maybe (bzw. "best effort")
    - Geeignet, wenn sowieso Fehlerbehandlung durch Client selbst oder durch Anwender im interaktiven Fall
  - ◆ At most once
    - Es wird sichergestellt, dass der Aufruf maximal einmal ausgeführt wird;
      Wiederholtes senden von Requestnachrichten mit eindeutigen IDs, Erkennung von Wiederholungen, ggf. erneutes Senden bereits ermittelter Ergebnisse
  - ◆ At least once
    - Aufruf kann mehrfach ausgeführt werden (i.d.R. begrenzte Wiederholung von Requestaufrufen, bis Ergebnis da).
    - Geeignet bei idempotenten Operationen
  - ◆ Last-of-many/Last-one
    - Sicherstellung, dass Ergebnis vom letzten Aufruf zurückgeliefert wird
  - ◆ Exactly-Once

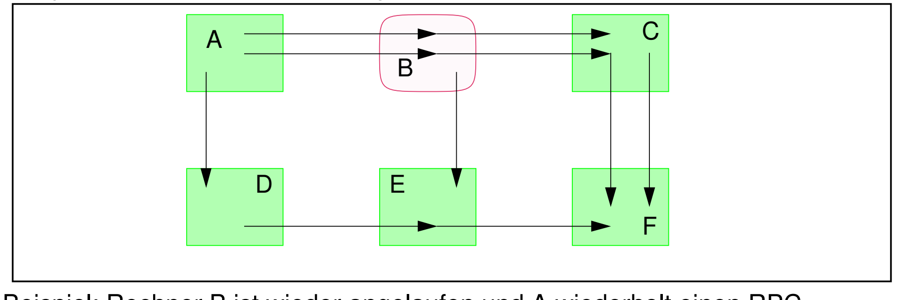
J.6 Orphan-Behandlung

- Was sind Orphans und wie entstehen sie?

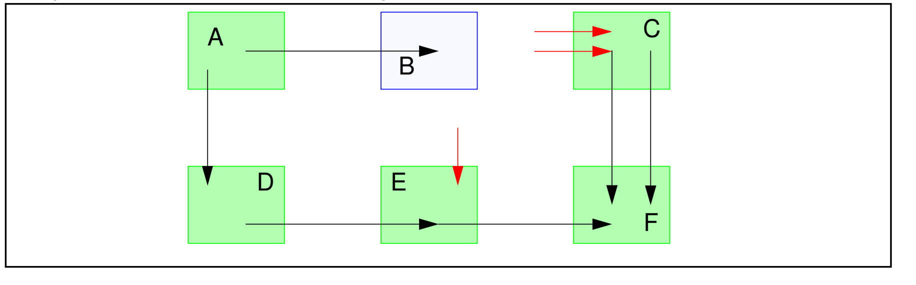
  Orphans oder verwaiste Aufrufe sind unerwünschte Ausführungen von Aufrufen und können durch das Ablaufen des letzten Timeouts, durch den Ausfall eines beteiligten Rechners (Clients) oder durch den Ausfall der Kommunikationsverbindung entstehen. Orphans sind die Teile von RPC-Aufrufen auf Servern, die keinen Kontakt mehr zu ihren Clients haben.

  Um auch beim RPC die last-one Semantik des lokalen Prozeduraufrufs aufrechtzuerhalten, muss vor dem Neustart oder dem Wiederaufsetzen eines verteilten Programms sichergestellt werden, dass alle Aktivitäten vorhergehender und nun verwaister RPCs abgebrochen wurden!

Beispiel: Rechner B fällt kurzzeitig aus.



Beispiel: Rechner B ist wieder angelaufen und A wiederholt einen RPC.



- Behandlung der Orphans
  - ◆ Gezieltes Aufräumen (Extermination)
    - Rechner hat sich persistent gemerkt, auf welchen anderen Rechnern er Fernaufrufe ausgeführt hat. Diese werden nun gezielt terminiert
    - Rekursive Fortsetzung; Problem bei mehrern Ausfällen!
  - ◆ Epochenbasierte Verfahren (Reincarnation)
    - Nach Wiederanlauf eines Rechners beginnt eine neue Epoche. Sobald ein Rechner von einer (im vgl. zur lokalen) neueren Epoche erfährt, bricht er alle Fernaufrufe ab (einfache Variante), oder nur die Fernaufrufe, von denen der Auftraggeber nicht mehr existiert (aktives Rückfragen; "gentle Reincarnation")
  - ◆ Zeitbasierte Verfahren (Expiration / Deadlining)
    - Jeder Fernaufruf besitzt ein Zeitlimit, nach dessen Ablauf er entweder automatisch beendet wird, oder wieder ein Rückfrage erfolgt, ob der Auftraggeber noch existiert
    - Nach Wiederanlauf muss nur maximales Timeout abgewartet werden; Verfahren setzt eine brauchbare Zeitsynchronisation voraus
  - ◆ Kombination verschiedener Verfahren möglich

J.7 Transportprotokoll

- ◆ Was bietet bereits das Betriebssystem? In einfachen Systemen (z.B. Prozessautomatisierung) ist oft nur unsicherer Paketdienst gegeben
- ◆ TCP
  - Erleichtert Implementierung (z.B. keine Reihenfolgevertauschungen, man muss sich selbst nicht um Wiederholungen von Pakten kümmern - modulo Verbindungsabbrüche)
  - Problem: Overhead
- ◆ UDP
  - UDP oder äquivalenter Datagramm-Dienst ist i.d.R. vorhanden
  - Programmierer muss sich um alles selbst kümmern
  - Erlaubt sehr effiziente Implementierung (Ich bezahle nur, was ich wirklich brauche)

J.8 Adressierung

- ◆ Direkte Adressierung von Prodzeduren/Objekten ("aktive Nachrichten")
- ◆ Direkte Adressierung von Prozessen
- ◆ Indirekte Adressierung: Ports
- ◆ Indirekte Adressierung: Mailbox

J.9 Ausführungsschemata

- ◆ Primitver Fall: Jeweils nur ein Aktivitätsträger (Thread):
  - Knoten, der Fernaufruf ausführt, blockiert sich während auf eine Antwort auf seinen Fernaufruf wartet; kein Rückruf möglich
- ◆ Verbesserte Variante: Immer ein Aktivitätsträger zum Entgegennehmen und Ausführen von Aufrufen
  - Rückruf wird ermöglicht
  - Verklemmung in Spezialsituationen möglich (z.B. Semaphore mit blockierenden RPC-Aufrufen; zyklische RPC-Ketten)
- ◆ Flexibelste Variante: Thread-Pool oder dynamische Thread-Erzeugung
  - Mehrere eingehende RPC-Aufruf können parallel verarbeitet werden, keine Verklemmung mehr möglich

J.10 Transparenz

- Netzwerk~ (Zugriffs~, Orts~)
- Nebenläufigkeits~
- Replikations~
- Fehler~
- Migrations~
- Leistungs~
- Skalierungs~