5. Interprozess-Kommunikation

5.1 Sockets

Sockets sind Datenkanäle, über die Prozesse Daten austauschen können. Dies kann lokal, insbesondere aber auch über Rechnergrenzen hinweg und plattformübergreifend erfolgen.

Sockets sind allgemein verfügbar und häufig die Basis für komplexere Multi-Prozeß- und Netzwerkanwendungen. Selbst bei Verfügbarkeit einer Middleware-Lösung ist das Verständnis von Sockets wichtig.

Anmerkungen:

Freie Ports sollten Sie ab 10000 aufwärts finden.

Sollten Sie nach Neustart Ihrer Implementierung den Fehler erhalten, dass ein Port in Verwendung ist, so müssen Sie entweder mit dem Neustart etwas warten oder die Socket-Option SO_LINGER serverseitig setzen. Diese bewirkt, dass der Socket auch dann geschlossen (freigegeben) wird, wenn noch Daten anhängig, d.h. nicht geschrieben oder gelesen wurden.

Verwenden Sie TCP als Kommunikationsprotokoll.

Aufgabe 1: Socket-Kommunikation

Schauen Sie sich folgende Befehle an:

- **Client:** socket(), connect(), send(), recv(), close()
- **Server:** socket(), bind(), listen(), accept(), send(), recv(), close()

Vollziehen Sie das Beispiel in der Python-Dokumentation nach und experimentieren Sie damit. Übertragen Sie auch Integer-Zahlen.

Anm. 1: Im Modul *struct* finden Sie Funktionen zum Packen und Entpacken von Zahlen in und aus binären Strings, die Sie dann als binäre Bytefolgen versenden und empfangen können.

Anm. 2: Von Console kann man mit *input()* oder *raw_input()* lesen.

Aufgabe 2: Socket-Puffer

1. Was geschieht, wenn Sie Bytes in einen Socket schreiben, OHNE dass der Empfänger diese liest?
2. Wie können Sie die Pufferung des Sockets ändern?
3. Welche maximale Pufferung ist erreichbar?

Aufgabe 3: Client-Server mit fork() und Sockets

Erstellen Sie einen Server, der

- beliebig vielen Clients
- mit beliebig vielen Anfragen

bedienen kann. Dies ist eine echte **Multi-Prozeß-Client-Server-Netzwerkanwendung.**

Client:

1. Verbindungsaufbau zum Server
2. Schleife:
   a) Auswahlmenü für verschiedene Operationen mit Operandenabfrage
   b) Senden einer Kennung für
      - Addition zweier Zahlen
      - Multiplikation zweier Zahlen
      - Addition zweier Strings
      - das Verlassen der Schleife (CLIENT-ENDE)
        Verlassen der Schleife (break)
   c) Senden des ersten Operanden
   d) Senden des zweiten Operanden
   e) Lesen des Ergebnisses vom Server
   f) Ausgabe des Ergebnisses
3. Schließen der Socket-Verbindung

Server:

1. Passiver Verbindungsaufbau (socket(), bind(), listen())
2. Schleife:
   a) Verbindungen annehmen und Verbindungssocket liefern (accept())
   b) fork()
   c) Eltern-Prozeß: Schließen des Verbindungssockets
   d) Child-Prozeß:
      - Schließen des Serversockets
      - Schleife:
        - Abwickeln der Kommunikation mit dem Client bis zur Kennung CLIENT-ENDE
      - Schließen des Verbindungssockets
      - Beenden des Child-Prozesses

Damit haben Sie die Grundstruktur einer Client-Server-Multiprozeß-Anwendung kennengelernt. Gratulation!