Technische Hochschule Ingolstadt
Fakultät Informatik
Prof. Dr. Christian Facchi

Verteilte Systeme
WS 2024/25
Übungsblatt 3

1) Aufgabe (Reader/Writer):

Im Folgenden soll das typische Reader/Writer-Problem behandelt werden, wie dies zum Beispiel bei einem Zugriff auf einen Datensatz bei einer Reisebuchung vorkommen kann. Benutzen Sie als Implementierungsgrundlage den Algorithmus aus der Vorlesung, Kapitel III.6. Verwenden Sie an Stelle der Variablen ein Semaphor aus dem sie mit semctl und der Option GETVALL den Wert auslesen, bzw. SETVAL den Wert schreiben. Stellen Sie durch Synchronisation mit Semaphoren sicher, dass nur maximal ein Prozess schreibenden Zugriff hat und hierbei insbesondere kein lesender Prozess erlaubt ist. Die Anzahl der gleichzeitig lesenden Prozesse ist unbeschränkt. Verwenden Sie hierzu die folgenden Semaphore:

1. WRITER: Stellt sicher, dass nur maximal ein Prozess schreibt und keiner liest
2. READER: Ein Zähler der die Anzahl der lesenden Prozesse speichert.
   Achtung: dieses Semaphor wird nicht zum Blockieren verwendet, sondern nur als globale/r Variable/Zähler.
3. MUTEX: Zur Sicherstellung der Atomarität der Synchronisationsaktionen, falls erforderlich.

Gehen Sie hierbei wie folgt schrittweise vor.

a) Entwickeln Sie den Rahmen mit dem Aufruf von 5 Leseprozessen und 2 Schreibprozesse. Implementieren Sie jeweils die Funktionen der Lese-und Schreibprozesse, die in einer Schleife jeweils dreimal eine Zyklus durchlaufen:
   Simulierter Zugriff auf Variable; 1 Sekunde warten; unkritischer Bereich; 1 Sekunde warten.
   Schützen Sie den kritischen Bereich nicht mit Semaphoren. Der Zugriff erfolgt unsynchronisiert. Überprüfen Sie den Algorithmus mit geeigneten Ausgaben.

b) Implementieren Sie die Lösung aus Kapitel III.6 und überprüfen Sie deren Funktion.

2) Aufgabe (RPC):

Es soll eine Addition, Multiplikation und Quadrierung einer Zahl an einem anderen Rechner mit RPC implementiert werden. Gehen Sie hierzu wie folgt schrittweise vor:

a) Erstellen Sie Datei math.x zur Beschreibung der Basisdefinitionen (Datentyp, RPC-Identifikator, Versionsnummer) mit folgenden Inhalt:

```
/* math.x */

struct intpair {
        int a;
        int b;
};

program MATHPROG {
        version MATHVERS {
                int ADD(intpair) = 1;
                int MULTIPLY(intpair) = 2;
                int CUBE(int) = 3;
        }= 1;           /*version*/
} = 0x2001000; /*id userdefined*/
```

   Erzeugen Sie mit dem Aufruf
   `rpcgen math.x`
   die Dateien: `math.h math_svc.c math_clnt.c math_xdr.c`.

b) Erstellen Sie nun den Code für den Server in der Datei `server.c`. Binden Sie hierzu den lokalen Header `math.h (include „math.h")` und den Systemheader `rpc/rpc.h` ein. Erzeugen Sie nun die dementsprechenden Serverprozeduren. Als Beispiel dient die Signatur der Prozedur zum Addieren:

```
int * add_1_svc (pair, rqstp)
     intpair * pair;
     struct svc_req *rqstp;  /*not necessary in this example*/
```

Die Erweiterung _1_svc wird von rpcgen generiert. Hierbei entspricht die 1 der Version. Den ersten Parameter `a` können Sie mit `pair->a` ansprechen.
Das Binary erzeugen Sie mit folgender Anweisung:

```
cc -o math_server math_svc.c math_xdr.c server.c -lnsl
```

c) Hinweis: Es kann sein, dass das Packet `tirpc` an Ihrem Rechner installiert ist. In diesem Fall setzen sie beim Compilieren die Option `-I/usr/include/tirpc` und binden die entsprechende Bibliothek mit `-ltirpc` zusätzlich zu `nsl` ein.

d) Erstellen Sie den Code für den Client. Binden Sie hierzu ebenfalls die in Aufgabe b) genannten Header ein. Interpretieren Sie die Argumente der Kommandozeile wie folgt: Das erste Argument ist der Name des Servers, das zweite und dritte Argument jeweils zwei ganze Zahlen, deren Summe berechnet werden soll.
Den Zugriff auf den Server erzielen Sie durch die Verwendung eines Client-Handles cl (CLIENT *), den Sie mit der Funktion clnt_create erzeugen:
`clnt_create (servername, MATHPROG, MATHVERS, "tcp");`
Die Prozedur add_1 auf dem Server rufen sie z.B. mit `add_1 (&numbers, cl)` auf. Hierbei ist `numbers` von der Sorte `intpair`.
Das Binary erzeugen Sie mit folgender Anweisung:

```
cc -o  remo_math math_clnt.c math_xdr.c client.c -lnsl
```

Der Aufruf erfolgt dann z.B. mit::

```
remo_math localhost 3 4
```

Hinweis: Es kann sein, dass das Packet tirpc an Ihrem Rechner installiert ist. In diesem Fall setzen sie beim Compilieren die Option `-I/usr/include/tirpc` und binden die entsprechende Bibliothek mit `-ltirpc` zusätzlich zu `nsl` ein.

e) Testen Sie das zuerst auf Ihrem Rechner, indem Sie als Server `localhost` verwenden. Anschließend testen Sie dann die Anwendung auf verschiedenen Rechnern.

Hinweis: Vergessen Sie nicht den Server-Prozess zu starten.

Es kann auch erforderlich sein, dass rpcbind zu installieren ist:

```
sudo apt-get install rpcbind
sudo –i service rpcbind stop
sudo –i rpcbind –i –w
sudo –i service rpcbind start
```

Bei manchen LINUX-Installationen kann man das erneute Starten des rpcbin services umgehen, indem man den Master mit sudo aufruft

```
sudo ./math_server
```

Hinweise: Falls Sie Server und Client auf verschiedenen Rechnern installieren wollen, müssen Sie bei der Virtualbox die Option Netzwerkbrücke (networkbridge) aktivieren. Somit erhalten Sie eine von außen zugängliche Netzwerkadresse.
Achtung: Bei einer rechnerübergreifenden Verbindung ist, wenn beide Server und Client auf verschiedenen Rechner jeweils virtualisiert laufen eine Verbindung leider nicht möglich. Bei plain LINUX wäre das schon möglich.
Es kann sein, dass Sie rpcgen überreden müssen modernere Prototypen zu generieren. Dazu verwenden Sie die Option -C. kann auch sein, dass Sie den C-Compiler mit einem älteren Sprachstandard übersetzen lassen müssen: -std=c89
Es kann dass bei der Virtual box, dass bei network settings (devices/network) dass der Adapter2 als Bridge Adapter das simulierte device verwendet werden muss. Zudem Promiscus mode auf allow all zu stellen. Bitte die ipadresse des zweiten Adapters nehmen.