Praktikum 7: Client-Server-Anwendung

Die Lernziele in diesem Praktikum sind

- Implementieren einer Client-Server-Architektur.
- Verwenden von POSIX-Message-Queues (<mqueue.h>).
- Implementieren einer Multi-Thread-Anwendung.
- Signale abfangen.

Zu realisieren ist eine einfache Client-Server-Anwendung in C++, bei der Client-Prozess und Server-Prozess mittels POSIX-Message-Queues kommunizieren.

Der Client nimmt Aufträge zur arithmetischen Berechnung in folgender Form entgegen:

```
opCode op1 op2
```

Dabei kann opCode eines der Zeichen '+' oder '-' sein. op1 und op2 sind ganze Zahlen. Der Client leitet den Auftrag zur Durchführung der eigentlichen Berechnung an den Server weiter. Nach Erhalt des Ergebnisses wird dieses vom Client auf dem Bildschirm ausgegeben. Bei Eingabe des Zeichens 'X' terminiert der Client. Sie finden den Client-Code bereits fertig im Export-Verzeichnis.

Ein beispielhafter Dialog mit dem Client könnte folgendermaßen verlaufen:

Beispiel

Ausgabe: Auftrag eingeben:

```
Eingabe: + 10 20
Ausgabe: 30
...
Ausgabe: Auftrag eingeben:
Eingabe: X
```

Aufgabe 1
Die eigentliche Berechnung führt ein Server-Prozess durch. Dieser empfängt den Rechenauftrag vom Client, interpretiert ihn und sendet das Ergebnis an den Client zurück. Ihre Aufgabe ist es den entsprechenden Server zu implementieren.

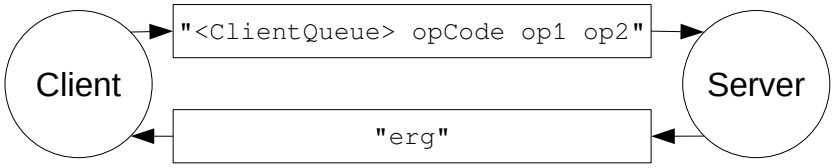

Abbildung 1: Kommunikation zwischen Client und Server mittels zweier Message-Queues.

Folgende Schritte sind nötig:

1. Die Kommunikation erfolgt über zwei Message Queues. Eine, welche vom Server erzeugt und gelöscht wird. Auf dieser übermittelt der Client den Auftrag an den Server. Der Client richtet eine Message-Queue ein, auf der er seine Antwort erwartet (Abb. 1). Den Namen dieser Message-Queue muss er an den Server übermitteln.

2. Der Server soll als paralleler Server unter Verwendung von Threads implementiert werden. Der Haupt-Thread erzeugt die Message Queue und nimmt die Aufträge der Clients entgegen. Nach Empfang eines Auftrags erzeugt er einen neuen Thread und übergibt diesem den Auftrag. Der beauftragte Thread schreibt die Ergebnisantwort in die Message Queue und terminiert.

3. Der Abbruch des Servers erfolgt mittels des `kill`-Kommandos bzw. durch Eingabe von `<Ctrl-C>`. Dadurch erhält der Server das Signal `SIGTERM` bzw. `SIGINT` und terminiert.

   In beiden Fällen ist die Message Queue vor dem eigentlichen Abbruch gelöscht werden. Dies können Sie durch einen geeigneten Signal-Handler erreichen, den sie für die beiden Signale anlegen. Nutzen Sie in C++ den Header <csignal> für den Systemaufruf `signal();`

   Ein Aufruf von `exit()` ruft den Destruktor nur für globale und statische Objekte auf, nicht für dynamische und automatische Objekte im Heap/Stack.

Hinweise:

- Standardmäßig werden Threads mit dem detached-Status `joinable() == true` erzeugt. Hierdurch werden die von dem Thread belegten Ressourcen (z.B. der Stackspeicher) erst nach dem `join()` freigegeben. Die Methode `detach()` ändert den Status in `joinable() == false`. In diesem Falle werden die Ressourcen unmittelbar nach der Terminierung des Threads freigegeben.

- Zur Durchführung einer Berechnung müssen die in der Zeichenkette enthaltenen Argumente extrahiert und in int-Zahlen konvertiert werden. Hierzu eigenen sich String-Streams, die Sie wie `cin` mit dem Eingabe-Operator verwenden.

```cpp
#include<string>
#include<sstream>
#include<iostream>

using namespace std;

int main() {
  string str = "10 20";
  stringstream sstr(str);
  int a = 0;
  int b = 0;
  sstr >> a >> b;
  cout << a << " " << b; // 10 20
}
```