

Zusammenspiel SharedMemory und Multi-Threading am Beispiel einer Bankinstituts

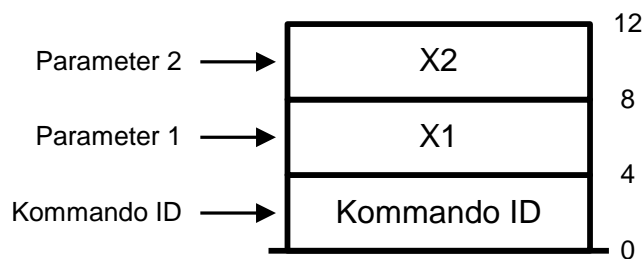
Überblick:

Wir betrachten eine verteilte Anwendung (Architektur: *Client-Server* Modell), die 20 Girokonten verwaltet. Der Server ist über eine *SharedMemory*-Schnittstelle ansprechbar. Ein Client kann im speziellen

- ein Girokonto anlegen,
- auf dieses Konto einen Geldbetrag einzahlen oder abheben und
- den Kontostand abfragen.

Aufbau SharedMemory-Bereich:

Der SharedMemory-Bereich umfasst 12 Bytes und ist wie folgt zu strukturieren:



Spezifikation der Kommandos

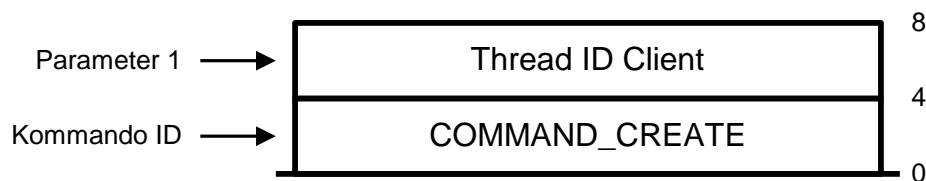
Es können 4 Kommandos im SharedMemory-Bereich eingetragen werden. Jedes Kommando wird durch Initiative des Clients eingetragen und wird entweder

- vom Server bearbeitet, ohne dass eine Rückmeldung zum Client erfolgt oder
- vom Server bearbeitet und mit einer Rückmeldung – die mit der Win32-API-Funktion `PostThreadMessage` durchgeführt wird – quittiert.

Die Kommandos besitzen folgende Werte:

```
#define COMMAND_CREATE      1 // Girokonto eröffnen
#define COMMAND_DEPOSIT     2 // Geld einzahlen
#define COMMAND_WITHDRAW    3 // Geld abheben
#define COMMAND_BALANCE     4 // Kontostand abfragen
```

Kommando „COMMAND_CREATE“:



Die Kommando ID ist im ersten Parameter abzulegen, die ID eines client-seitigen Threads im zweiten Parameter. Die Girokontonummer – das Resultat dieses Kommandos – ist folglich dem Client-Thread mit `PostThreadMessage` zuzustellen.

Kommando „COMMAND_DEPOSIT“:



Die Kommando ID ist im ersten Parameter abzulegen. Im zweiten Parameter ist die Girokontonummer zu spezifizieren, im dritten Parameter die Höhe des Geldbetrags. Der spezifizierte Geldbetrag ist in das Konto einzuzahlen.

Beachte: Dieses Kommando führt keine Rückmeldung zum Client aus.

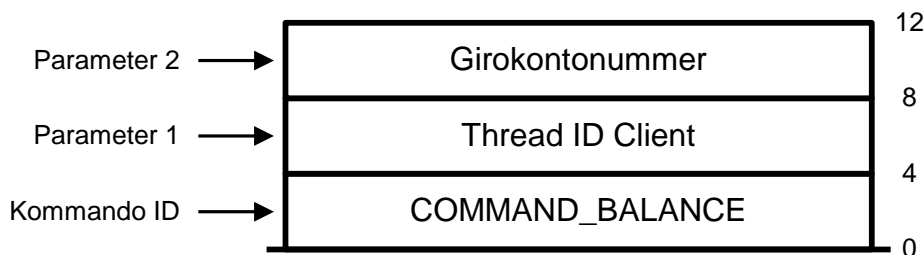
Kommando „COMMAND_WITHDRAW“:



Die Kommando ID ist im ersten Parameter abzulegen. Im zweiten Parameter ist die Girokontonummer zu spezifizieren, im dritten Parameter die Höhe des Geldbetrags. Der spezifizierte Geldbetrag ist vom Konto abzuheben.

Beachte: Dieses Kommando führt keine Rückmeldung zum Client aus.

Kommando „COMMAND_BALANCE“:



Die Kommando ID ist im ersten Parameter abzulegen, die ID eines client-seitigen Threads im zweiten Parameter. Die Höhe des Girokontos – das Resultat dieses Kommandos – ist folglich dem Client-Thread mit `PostThreadMessage` zuzustellen.

Architektur des Servers:

Der Server ist als Windows-Konsolenapplikation zu realisieren. Der Hauptteil dieser Anwendung *könnte* wie folgt realisiert sein (Pseudocode):

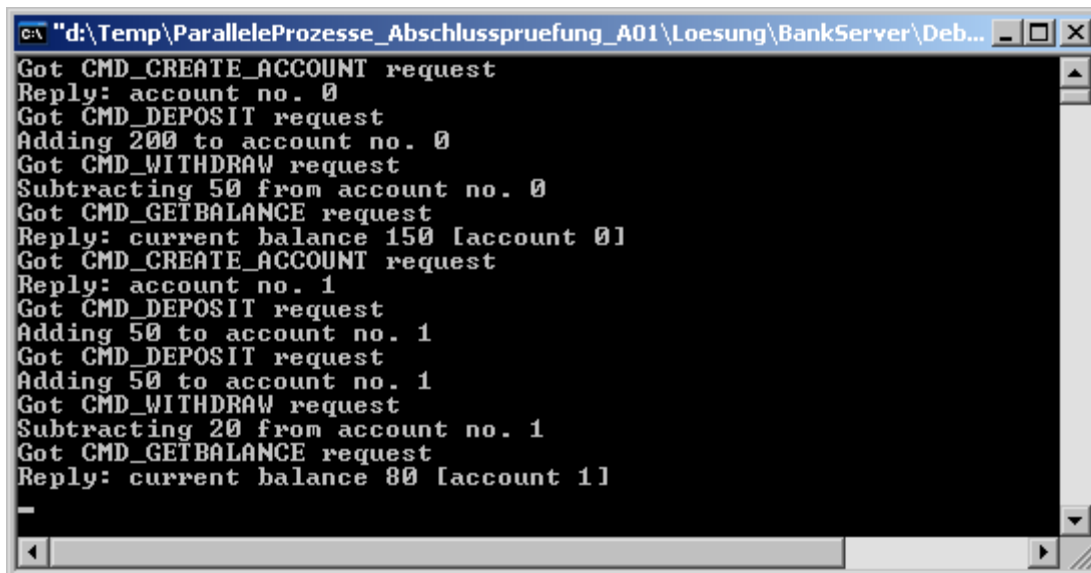
```
while (true)
{
    1. retrieve request from shared memory object (if any exists)

    2. handle request (if any)

    3. wait 50 milli seconds for the next request
}
```

Ein Clientprozess muss zur Übermittlung eines Kommandos an den Server zunächst auf das Shared-Memory-Objekt zugreifen und dort die entsprechende Kommando-ID eintragen. Der Server liest diese ID *zyklisch* aus und führt bei Bedarf die entsprechende Aktion aus.

In der Konsole des Serverprozesses sollten alle Ereignisse im Server wie etwa das Anmelden eines Clients, das Einrichten eines Girokontos usw. protokolliert werden, so dass ein Bediener die Aktivitäten des Servers verfolgen kann:



```

c:\d:\Temp\ParalleleProzesse_Abschlusspruefung_A01\Loesung\BankServer\Deb...
Got CMD_CREATE_ACCOUNT request
Reply: account no. 0
Got CMD_DEPOSIT request
Adding 200 to account no. 0
Got CMD_WITHDRAW request
Subtracting 50 from account no. 0
Got CMD_GETBALANCE request
Reply: current balance 150 [account 0]
Got CMD_CREATE_ACCOUNT request
Reply: account no. 1
Got CMD_DEPOSIT request
Adding 50 to account no. 1
Got CMD_DEPOSIT request
Adding 50 to account no. 1
Got CMD_WITHDRAW request
Subtracting 20 from account no. 1
Got CMD_GETBALANCE request
Reply: current balance 80 [account 1]
-
```

Beachte:

Beim Auslesen eines Kommandos durch den Server ist dieses unbedingt zu löschen – zum Beispiel durch das Pseudokommando `COMMAND_NO_REQUEST` zu ersetzen – andernfalls würde der Server nach Ablauf einer Zeitscheibe dasselbe Kommando zum zweiten Mal auslesen und ausführen!

```
#define COMMAND_NO_REQUEST    0    // no request
```

Architektur des Clients:

Der Client ist ebenfalls als NT-Konsolenapplikation zu realisieren. Mit Hilfe simpler `printf`-Anweisungen sollte folgender Dialog mit dem Benutzer möglich sein:

- Geldbetrag einzahlen (Geldbetrag von der Konsole einlesen)
- Geldbetrag abheben (Geldbetrag von der Konsole einlesen)
- Kontostand abfragen

Ohne Dialog – also eigenständig beim Hochfahren oder Schließen des Clients – sollten folgende Aktionen ausgeführt werden:

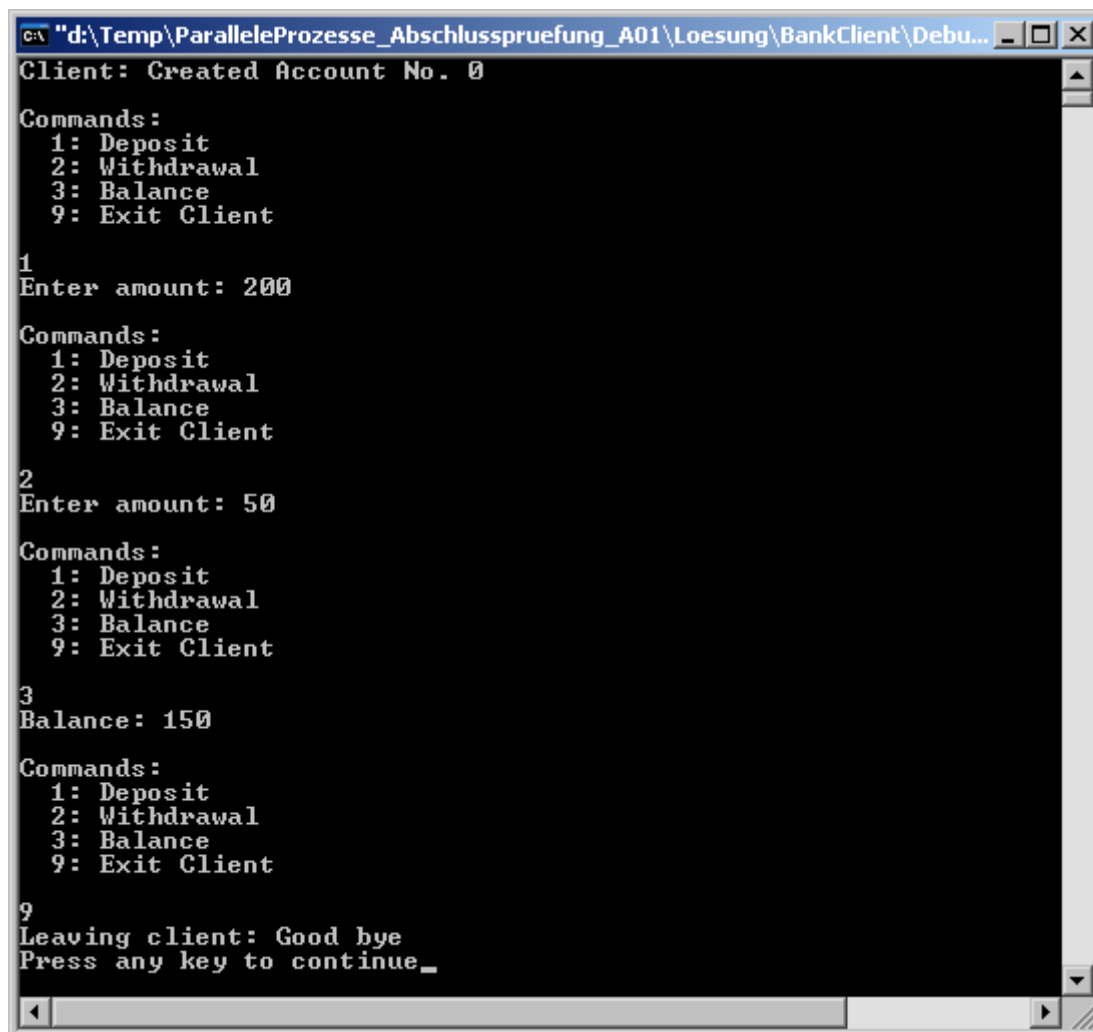
Im Hochlauf:

- Verbindungsaufbau zum SharedMemory-Objekt
- Girokonto anlegen

Bei Beendigung der Anwendung:

- Verbindungsabbau des SharedMemory-Objekt

Die Oberfläche des Clients sollte in etwa so aussehen:



```
Client: Created Account No. 0

Commands:
1: Deposit
2: Withdrawal
3: Balance
9: Exit Client

1
Enter amount: 200

Commands:
1: Deposit
2: Withdrawal
3: Balance
9: Exit Client

2
Enter amount: 50

Commands:
1: Deposit
2: Withdrawal
3: Balance
9: Exit Client

3
Balance: 150

Commands:
1: Deposit
2: Withdrawal
3: Balance
9: Exit Client

9
Leaving client: Good bye
Press any key to continue_
```