

Prozess- und Thread-Programmierung mit C++ unter Unix

Simulation eines Telebankingsystems

Michael Rennecke

Dozentin: Dipl.-Inform. Annett Thüring

Halle, 2006

Inhaltsverzeichnis

1	Aufgabenstellung	1
2	Lösung	2
2.1	Überlegungen aus der Aufgabenstellung	2
2.2	Aufbau des Server	2
2.3	Überlegungen zum Client	3
3	Kommunikationsmodell	3
3.1	Ablauf	3
3.2	Nachrichtenformate	4
3.2.1	Client-Authentifikationsserver-Kommunikation	4
3.2.2	Client-Transaktionsserver-Kommunikation	4
3.2.3	Server-Client-Kommunikation	4
3.3	Einschätzung dieses Modells	5
4	Implementierung	5
5	Einschätzung der Lösung	5
5.1	Laufzeiten	5
5.2	Verbesserungsmöglichkeiten	6

1 Aufgabenstellung

Programmieren Sie die Anwendung Telebanking. Die Kunden wenden sich über ein Terminal (zum Beispiel einen Geldautomaten) an den Server einer Bank. Schreiben Sie die Anwendung möglichst realistisch!

Aufgaben des Client:

- Nimm die Kontonummer und eine 4-stellige PIN des Kunden entgegen.
- Biete dem Kunden ein Menü an, in dem er bestimmte Auszahlungsbeträge wählen kann: 50, 100, 200, 500 oder einen anderen Betrag.
- Biete dem Kunden eine Kontostandsabfrage an.
- Biete dem Kunden eine Abbruchfunktion an.
- Gib dem Kunden mögliche Fehlermeldungen aus:
 - „Falsche Kontonummer, PIN, Erneut versuchen!“
 - „Vorgang abgebrochen. Konto hat einen zu geringen Betrag!“
 - „Auszahlung erfolgt! Danke für Ihren Besuch!“

Aufgaben des Servers:

- Verwaltet Kontonummern, PINs und Kontostände.
- Führt Authentifizierung durch.
- Führt Transaktionen (Geld abheben) durch.
- Generiert die Meldungen.

Stellen Sie sicher, dass eine gleichzeitige Nutzung mehrerer Geldautomaten (jedoch von verschiedenen Personen) durch den Server gewährleistet ist. Realisieren Sie die Server-Anwendung mit Threads (falls möglich, dann auch mit Prozessen). Stellen Sie den wechselseitigen Zugriff der Threads (Prozesse) auf die Nutzerdaten sicher. Wählen Sie ein geeignetes Konzept zum Datenaustausch zwischen den einzelnen Geldautomaten und dem Server.

2 Lösung

2.1 Überlegungen aus der Aufgabenstellung

Da der Server und der Client 2 voneinander unabhängige Programme sind kommt für die Kommunikation zwischen den beiden nur *named Pipes*¹ und *Message-Queues*² in Frage. Die anderen Mittel der Interprozesskommunikation kann man nicht anwenden, da es sich um völlig verschiedene Programme handelt, d.h. man kann sie nicht aus *einen* Vaterprozess heraus starten.

Weiterhin ist es nötig, dass *viele* Clients auf *einen* Server zugreifen. Hierbei ist der wechselseitige Ausschluss sicher zu stellen. Wie man sicher leicht fest stellt, muss man an dieser Stelle die Bidirektionalität³ der Kommunikation beachten.

2.2 Aufbau des Server

Aus der Tatsache, dass viele Clients auf einen Server zugreifen, ist es von Vorteil, wenn man den Server in mehrere Prozesse splittet. Man sollte die Prozesse nicht mit *fork()* erzeugen, da hierbei ein großer Overhead entsteht. Denn alle Sproprozesse erben die gesamte Funktionalität des Vaterprozess. Also empfiehlt es sich den Server aus eigenständigen Programmen aufzubauen, für die Kommunikation der Teile des Serveres untereinander greifen die gleichen Maßnahmen wie bei der Kommunikation Server-Client. Hierbei handelt es sich aber um eine 1 zu 1-Verbindung. Also gibt es hier nicht viel zu beachten. Es empfiehlt sich den Server in folgende Teile zu gliedern:

1. **Authentifizierungsserver**

Führt die Authentifizierung des Clientnutzers durch. Dies geschieht durch den Abgleich von Kontonummer und PIN. Die Tupel Kontonummer PIN sollte man in einer Datei speichern.

2. **Transaktionenserver**

Dieser führt die eigentlichen Transaktionen durch, d.h. das Abheben des Geldes. Hierfür sollte man eine Datei bereit stellen in der die Tupel (Kontonummer, Kontostand) stehen. Beim Abheben muss natürlich der Kontostand aktualisiert werden. Wenn man nur den Kontostand wissen möchte liest man einfach den entsprechenden Eintrag aus.

Natürlich kann man den Server in mehr Teile splitten, aber für die Erfüllung der Aufgabenstellung reicht diese Einteilung. Somit erreicht man folgendes: Wenn man die falsche PIN eingibt, können schon die Anfragen an den Transaktionenserver bearbeitet werden. Außerdem werden die Programme kleiner und dadurch übersichtlicher was zu einer besseren Wartbarkeit führt.

¹einen gepufferten Datenstrom zwischen zwei Prozessen nach dem First In - First Out-Prinzip.

²prioritätsgesteuerte Warteschlangen

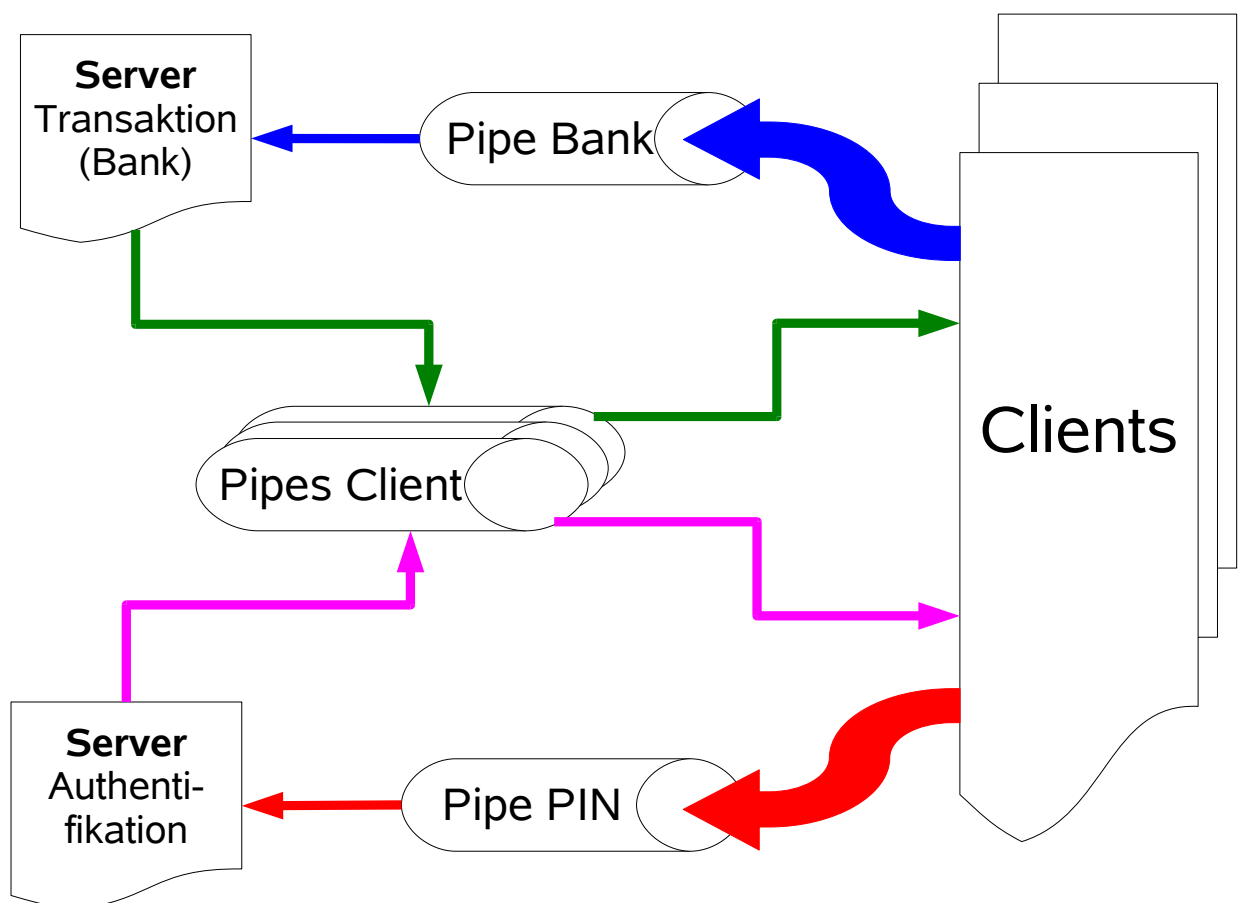
³eine Datenübertragung in beide Richtungen von Punkt zu Punkt

2.3 Überlegungen zum Client

Beim Client gibt es eigentlich nichts zu beachten. Man muss nur ein entsprechende Schnittstelle bereit stellen für die Mensch-Maschine-Kommunikation. Für Testzwecke ist es sehr nützlich, wenn man zusätzlich einen Client hat, den man über Parameter steuern kann und ohne (sichtbare) Schnittstelle für die Mensch-Maschine-Kommunikation. Denn egal wie spartanisch man eine solche Schnittstelle ausstattet, verbraucht sie doch Ressourcen des Rechners. Man könnte auch einen Client programmieren, der alle Merkmale in sich vereinigt. Aus Gründen der Übersichtlichkeit ist es jedoch besser 2 Clients zu schreiben.

3 Kommunikationsmodell

3.1 Ablauf



Der Client stellt seine Anfrage an den Authentifikationsserver. Dabei schreiben *alle* Clients in *eine* named Pipe (dicker roter Pfeil), welche vom Authentifikationsserver verwaltet wird. Der Authentifikationsserver liest diese Pipe aus und verarbeitet den Inhalt. Nun

sendet der Authentifikationsserver eine Antwort (magenta farbener Pfad) an den Client über eine Pipe. Diese Pipe verwaltet der Client und *jeder* Client besitzt seine eingene „Antwortpipe“, d.h. der Authentifikationsserver muss die passende Pipe für jeden Client wählen. Je nach dem wie die Antwort lautet wird der Vorgang der Authentifikation wiederholt oder die Anfrage an den Transaktionsserver gestellt.

Für die Anfrage an den Transaktionsserver, schreiben *alle* Clients in *eine* named Pipe (dicker blauer Pfeil), welche vom Transaktionsserver verwaltet wird. Dieser arbeitet dann die Anfrage ab und sendet eine Antwort über die *selbe* Pipe über die die Client-Authentifikationsserver-Kommunikation statt findet. Das kann man machen, weil es keine weitere Kommunikation zwischen Authentifizierungsserver und Client gibt und die Pipe leer ist. Das ist der grüne Pfad.

3.2 Nachrichtenformate

3.2.1 Client-Authentifikationsserver-Kommunikation

(roter Pfad)

Datenpaket:

PID	Kontonummer	PIN
-----	-------------	-----

Alle Daten werden ASCII-codiert übertragen. Mit der PID⁴ wird der Client identifiziert und der Benutzer wird mit dem Tupel Kontonummer, PIN authentifiziert. Durch die PID kann der Authentifikationsserver die Pipe zum Client identifizieren.

3.2.2 Client-Transaktionsserver-Kommunikation

(blauer Pfad)

Datenpaket:

PID	Kontonummer	Betrag
-----	-------------	--------

Alle Daten werden ASCII-codiert übertragen. Die PID erfüllt den selben Zweck wie bei der Client-Authentifikationsserver-Kommunikation. Wenn der Betrag positive, ganzzahlige Werte enthält wird der entsprechende Betrag ausgezahlt, sofern möglich, und abgebucht. Dies macht der Transaktionsserver. Bei einem negativen Wert gibt der Transaktionsserver den Kontostand zurück.

3.2.3 Server-Client-Kommunikation

(magenta farbener und grüner Pfad)

Die Antworten vom Authentifizierungs- und Transaktionsserver werden als normaler ASCII-Text übertragen. Es handelt sich hierbei nur um Statusmeldungen.

⁴Prozessidentifikationsnummer, wird vom Betriebssystem vergeben und ist eindeutig

3.3 Einschätzung dieses Modells

- **Pro**

1. Modell kann mit named Pipes oder mit Message-Queues umgesetzt werden
2. es entsteht kein Kommunikationsoverhead
3. leichte Implementierung

- **Contra**

Bei diesem Modell sind keinerlei Sicherheitsmaßnahmen eingebaut. Wenn man den Client hackt, könnte man ohne Authentifizierung Geld abheben. Ich habe absichtlich darauf verzichtet da diese von der grundlegenden Aufgabenstellung nur ablenkt und alles unnötig verkompliziert.

4 Implementierung

Bei der Implementierung habe ich mich auf named Pipes beschränkt. Da diese in ihrer Benutzung recht einfach zu handhaben sind. Weiterhin ist mir aufgefallen, dass sie Geschwindigkeitsvorteile gegenüber den Message-Queues hatten. Dieser Umstand kann aber der Aufgabenstellung geschuldet sein, d.h. man kann diese Aussage nicht pauschalisieren. Der Authentifizierungs- und Transaktionsserver arbeiten in einer Endlosschleife. In dieser wird nachgesehen ob sich was in der Pipe befindet. Wenn dies der Fall ist, so wird der Auftrag abgearbeitet. Nach der Abarbeitung wird eine Antwort an den Client gesendet. Dies wird ermöglicht, da die PID des Client dem Server bekannt ist und die Pipe an den Client den Namen *fifo.Client_PID* hat.

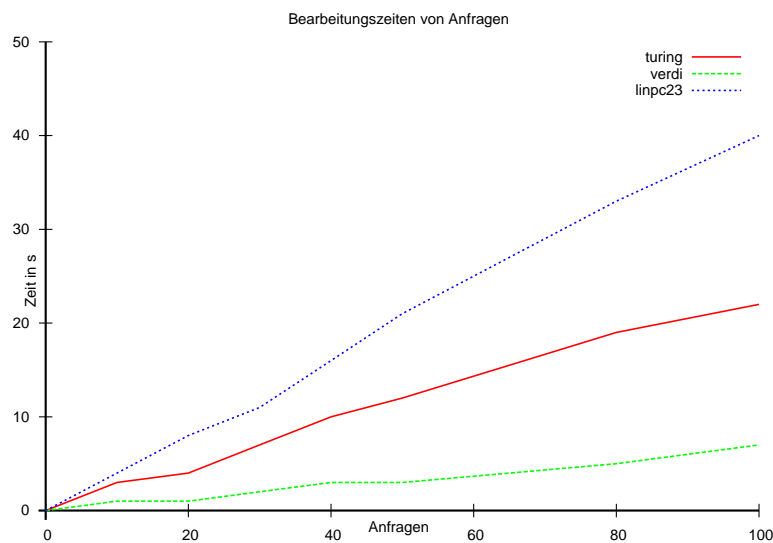
Für die Funktionsweise des Client reicht es wenn man sich den Quellcode ansieht. Im Großen und Ganzen gibt der Benutzer seine Daten ein, diese werden dann an den entsprechenden Server gesendet und entsprechend der Antwort des Server wird weiter gearbeitet. Es sollte erwähnt werden, dass der Client solange schläft bis etwas in seiner Pipe steht, d.h. ein wartender Client belastet nicht das System.

5 Einschätzung der Lösung

5.1 Laufzeiten

Wie man sieht, steigen die Laufzeiten⁵ auf allen 3 Testsystemen linear an. Die Laufzeiten brechen auch bei größeren Problemgrößen nicht ein. Es kann sein, wenn die Problemgröße zu groß ist und das System so viele Daten nicht im Hauptspeicher halten kann, dass dann die Laufzeiten sehr stark anwachsen. Das habe ich aber nicht getestet.

⁵die Problemgröße ist die Anzahl der Clients, die auf den Server zugreifen



Anfragen	linpc23	turing	verdi
10	4 s	3 s	1 s
20	8 s	4 s	1 s
30	11 s	7 s	2 s
40	16 s	10 s	3 s
50	21 s	12 s	3 s
80	33 s	19 s	5 s
100	40 s	22 s	7 s
500	3:22 min	2:06 min	31 s
1000	6:29 min	4:19 min	1:11 min

5.2 Verbesserungsmöglichkeiten

Zur Verbesserung der Laufzeiten kann man mehrere Server starten, sodass man mehrere Anfragen parallel bearbeiten kann. Das ist aber nur auf Mehrprozessorsystemen sinnvoll. Beim Transaktionsserver muss der wechselseitigen Ausschluss, auf die Datei mit den Konten, sicher gestellt werden. Bei der Authentifikation ist das nicht unbedingt erforderlich, da hier nur ein *lesender* Zugriff auf die PIN-Datei statt findet.

Bei der vorgestellten Lösung müssen sich Server und Client auf dem *selben* Rechner befinden. Das ist nicht sehr praxisnah. Es ist also zu überlegen die Kommunikation zwischen Client und Server mittels einer geeigneten Netzwerkschnittstelle zu lösen.