OPERATING SYSTEMS BEISPIEL 3

Aufgabenstellung – auth

In dieser Aufgabe sollen Sie eine passwort-geschützte Datenbank entwickeln. Die Implementierung soll aus zwei Programmen bestehen: einem Server, der die Datenbank verwaltet und Anfragen über deren Inhalt bearbeitet, und einem Client, mit welchem der Benutzer Informationen der Datenbank vom Server setzen oder abfragen kann. Die Kommunikation zwischen den Prozessen soll mittels Shared Memory realisiert werden und die Synchronisation über Semaphore erfolgen.

Der Server speichert folgende die Benutzerdaten - Name des Benutzers (`username`), Passwort (`password`) und ein Geheimnis (`secret`) - in einer Datenbank. Der Client soll dem Benutzer ein Interface bieten, mit dem der Benutzer diese Informationen vom Server setzen und abfragen kann.

Anleitung

Die Kommunikation zwischen den Clients und dem Server soll mittels einem einzigen Shared Memory Object erfolgen (**nicht** einem pro Client). Es darf auch nicht die gesamte Datenbank im Shared Memory geladen sein, sondern nur die Information, die der Server mit einem einzigen Client austauscht. Allerdings muss eine beliebige Anzahl von Clients gleichzeitig und unabhängig voneinander mit dem Server kommunizieren können. Insbesondere darf das Warten auf den Input eines Clients nicht andere Clients blockieren.

Server und Client sollen die Freigabe des Shared Memory Objects und der Semaphore koordinieren. Spätestens wenn der Server und alle Clients terminiert haben, müssen alle Semaphore und Shared Memory Objects freigegeben sein.

Server

```
USAGE: auth-server [-l database]
```

Der Server legt zu Beginn die benötigten Resourcen an. Anschließend lädt der Server die angegebene Datenbank in einer geeigneten Datenstruktur in den Speicher. Wird diese Option beim Start nicht angegeben, so verwenden Sie bitte eine leere Datenbank als Ausgangspunkt. Die Datenbank dient dem Server im Weiteren als Informationsquelle für das Bearbeiten der Client-Anfragen. Gibt es ein Problem beim Einlesen der Datenbank beenden Sie den Server mit einer Fehlermeldung.

Nach der Initialisierung, bearbeitet der Server Anfragen von Clients. Der Server legt den vom Client gewünschten Eintrag an oder sucht den entsprechenden Eintrag in der Datenbank und sendet dem Client eine Antwort mit der gewünschten Information.

Der Server soll bei *jeder Terminierung* seine Datenbank in eine Datei names `auth-server.db.csv` schreiben. Falls die Datei bereits existiert, soll diese einfach überschrieben werden.

Client

```
USAGE: auth-client { -r | -l } username password
```

Der Client wird entweder mit Option `-r` oder mit Option `-l` ausgeführt. Diese Optionen entsprechen den folgenden Ausführungsmodi:

Register Mit der Option `-r` kann ein neuer Benutzer registriert werden. Falls der Benutzer (`username`) schon existiert, soll der Server dies dem Client entsprechend kommunizieren. Der Client soll in diesem Fall nach Ausgabe einer entsprechenden Fehlermeldung terminieren (`EXIT_FAILURE`). Existiert noch kein Benutzer mit dem angegebenen Namen (`username`) in der Datenbank des Servers, so soll der Benutzer angelegt werden und dies dem Client mitteilen. Der Client gibt eine entsprechende Erfolgsmeldung aus und terminiert (`EXIT_SUCCESS`).

Login Mit der Option `-l` kann sich ein bereits registrierter Benutzer einloggen. Der Server soll überprüfen, ob der gegebene Benutzer im System vorhanden ist oder nicht. Falls der Benutzer ungültig ist, soll der Server dies dem Client kommunizieren. Der Client soll nach Ausgabe einer entsprechenden Fehlermeldung terminieren (`EXIT_FAILURE`). Im Erfolgsfall soll der Client eine entsprechende Erfolgsmeldung ausgeben und auf weitere Befehle des Benutzers warten.

Nach einem erfolgreichen Login soll der Benutzer über die Standardeingabe wiederholt einen der folgenden drei Befehle eingeben können:

write secret Der Benutzer soll eine Nachricht auf die Standardeingabe schreiben können, die als `secret` am Server gespeichert und mit dem eingeloggten Benutzer assoziiert wird.

read secret Der Client soll das entsprechende Geheimnis ausgeben.

logout Der Server loggt den Benutzer aus und der Client terminiert (`EXIT_SUCCESS`).

Der Client erfragt den gewünschten Befehl bis sich der Benutzer ausloggt. Gestalten Sie am Client ein einfaches Benutzerinterface, etwa:

```
Commands:
  1) write secret
  2) read secret
  3) logout
Please select a command (1-3):
```

Die Inputs (Befehlsnummer und Geheimnis) sollen von der Standardeingabe gelesen werden.

Um sicher zu gehen, dass der Client auch wirklich eingeloggt ist, soll bei erfolgreichem Login eine zufällige Session-ID vom Server erzeugt, und an den Client gesendet werden. Diese ID muss bei jeder Kommunikation vom Client an den Server gesendet werden, um zu verifizieren, dass es sich um den authentifizierten Client handelt. Sollte ein Client mit falscher Session-ID versuchen, zum Beispiel auf das *secret* eines anderen Clients zuzugreifen, so soll der Server eine Fehlermeldung ausgeben und dem Client signalisieren, dass die Session ungültig ist. Beim Logout soll die Session-ID zerstört werden.

Datenformat

Datenbank

Die Datenbank soll als CSV-Datei gestaltet werden. Jede Zeile beinhaltet ein Geheimnis `secret` eines Benutzers `username`, das mit einem Passwort `password` geschützt ist.

```
username;password;secret
```

Achten Sie darauf, keine unnötigen Leerzeichen zwischen den Semikolons einzufügen, und insbesondere darauf, beim Speichern des secrets keinen Zeilenumbruch (`'\n'`) mitzuspeichern, wodurch leere Zeilen in der Datenbank entstehen würden.

Sie dürfen für alle Felder (`username`, `password`, `secret`) eine maximale Länge definieren um Ihnen die Arbeit zu erleichtern.

Beispiel

```
Theodor;ilovemilka;I'm the best!
Anton;cforever;
Emil;osueisgreat;Something you would like to know, but I won't say ANYBODY!
```

Der zweite Benutzer namens `Anton` hat in diesem Beispiel kein Geheimnis gespeichert.

Shared Memory

Es soll eine geeignete Struktur definiert werden, die Felder für die Anfrage, als auch für die Antwort enthält.

Richtlinien

Beachten Sie unbedingt auch die *Richtlinien für die Erstellung von C-Programmen* („Coding Guidelines“) auf TUWEL, sowie die folgenden allgemeinen Hinweise zur Beispielgruppe 3.

Entwickeln Sie die zu implementierenden Prozesse als getrennte Programme, die über POSIX Shared Memory (siehe `shm_overview(7)`) Daten austauschen. Synchronisieren Sie den Zugriff auf das Shared Memory mit POSIX Semaphoren (siehe `sem_overview(7)`).

Es sind Named Semaphores zu verwenden (`sem_open(3)`, `sem_close(3)` `sem_unlink(3)`).

Verwenden Sie bei der Benennung der Ressourcen Ihre Matrikelnummmer als Präfix!

Shared Memory Objekte und Semaphore werden nach Beendigung des Programms nicht automatisch gelöscht. Sie müssen daher im Programm explizit entfernt werden (siehe unten).

Sollten durch einen Programmfehler bzw. Absturz Shared Memory oder Semaphore nicht gelöscht werden, können diese im Verzeichnis `/dev/shm/` mit den üblichen Kommandos `ls` und `rm` gelistet bzw. gelöscht werden. Sie erkennen die von Ihnen angelegten Ressourcen am Namen (siehe oben) bzw. an der Ownership der zugehörigen Files.

Aufrufreihenfolge. Sollte bei einer Client/Server-Struktur (der Server übernimmt das Anlegen und Freigeben gemeinsamer Ressourcen) der Client noch keine der vom Server anzulegenden gemeinsamen Ressourcen vorfinden, so hat der Client mit einer Fehlermeldung zu terminieren. Ob eine Client/Server-Struktur Sinn macht, ist aus der Beispielangabe abzuleiten. Bei der Verwendung einer Server/Client Struktur, ist, wenn nicht anders angegeben, davon auszugehen, dass mehrere Clients gleichzeitig gestartet werden können. Der Server muß nach der fehlerfreien Terminierung eines Clients noch funktionsfähig bleiben. Es kann jedoch davon ausgegangen werden, dass der Server nur einmal gestartet wird. Wenn sie keine Client/Server Struktur verwenden, kann davon ausgegangen werden, dass jeder Prozess nur einmal gestartet wird und das in einer beliebigen Reihenfolge.

Signalbehandlung. Eine Signalbehandlung ist in dieser Beispielgruppe für alle Programme (z.B. Client und Server) durchzuführen (`SIGINT`, `SIGTERM`).

Ressourcenverwendung. Nutzen Sie ein Shared Memory fixer Größe in Ihrer Lösung. Nutzen Sie so wenige Semaphoren wie nötig; achten Sie jedoch auf den korrekt synchronisierten Zugriff – Ihrem Beispiel entsprechend – auf das Shared Memory. Falls Sie Fragen zur Synchronisation haben, wenden Sie sich bitte an unsere Tutoren in der regulär betreuten Übungszeit.

Terminierung. Nach einer fehlerfreien Programmabarbeitung soll eine synchronisierte Terminierung der Prozesse erfolgen. D.h., achten Sie beim Entfernen aller angelegten Ressourcen auf die Synchronisation zwischen den laufenden Prozessen. Sorgen Sie dafür, dass auch im Fehlerfall alle angelegten Ressourcen gelöscht werden.