Systemprogrammierung I - Wirtschaftsinformatiker mit SP1-Schein

Nachfolgend finden Sie die zwei Aufgaben, die Sie als Zulassungsvoraussetzung für das Kolloquium am Ende des Semesters lösen und abgeben müssen. Sollten Fragen zu den Aufgabenstellungen und/oder zu den Lösungsmöglichkeiten auftauchen, dann stehen wir Ihnen jederzeit zur Verfügung.

Aufgabe 1:

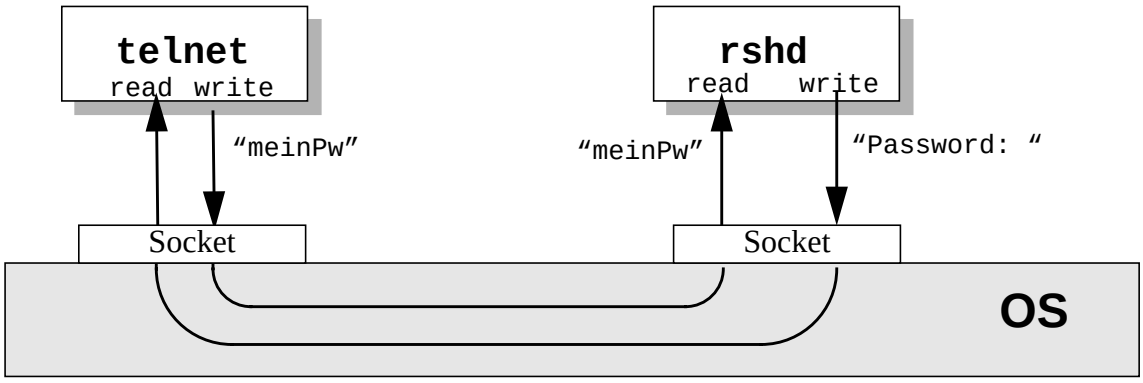
Remoteshell (rshd) (18 Punkte)

Programmieren Sie basierend auf der Musterlösung der Aufgabe 4 vom Wintersemester 1999/2000 (~i4sp/pub/aufgabe4) eine Shell, die ihre Kommandos nicht mehr von der Tastatur erhält, sondern über einen Socket.
Schreiben Sie dazu einen Daemon, welcher von außen z.B. mittels **telnet (1)** kontaktiert werden kann: Dieser Daemon soll für jede Verbindung einen neuen Prozeß starten, in welchem das unveränderte yash-Programm von Aufgabe 4 ausgeführt wird. Gehen Sie folgendermaßen vor:

a) Socket-Verbindung

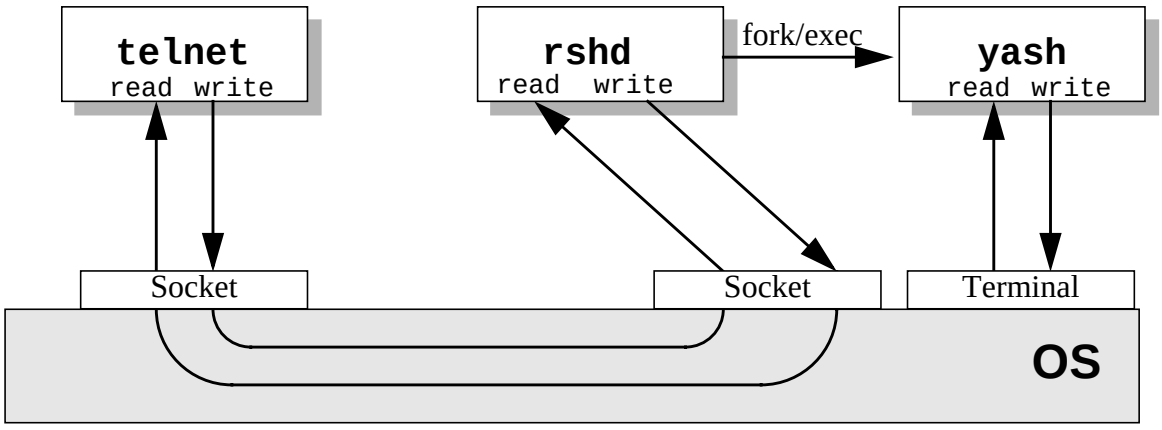
Beim Start des Daemons soll dieser nach einem “Paßwort” fragen. Dieses Passwort, soll dann bei jedem Verbindungsaufbau abgefragt werden. Das Passwort soll maximal 8 Zeichen lang sein.

Erzeugen Sie in Ihrem Daemon einen Streamsocket (**socket(2)**) und binden Sie ihn an alle Adressen des verwendeten Rechners (**bind(2)**). Die Portnummer soll Ihrer UID entsprechen (**id(1)**). Stellen Sie die Anzahl der wartenden Verbindungen auf 5 (**listen(2)**). Nehmen Sie dann Verbindungen über den Socket entgegen (**accept(2)**). Fragen Sie als erstes das beim Daemonstart gesetzte Passwort ab, d.h. schicken Sie der Client-Applikation (telnet) den String “Password:” und werten Sie die nächste Eingabe als Paßwort aus. Unterscheidet sich das Paßwort von dem, daß Sie beim Starten des Daemon angegeben haben, dann soll die Verbindung beendet werden: ***Die Shell läuft unter ihrer UID und kann von JEDEM angesprochen werden!***



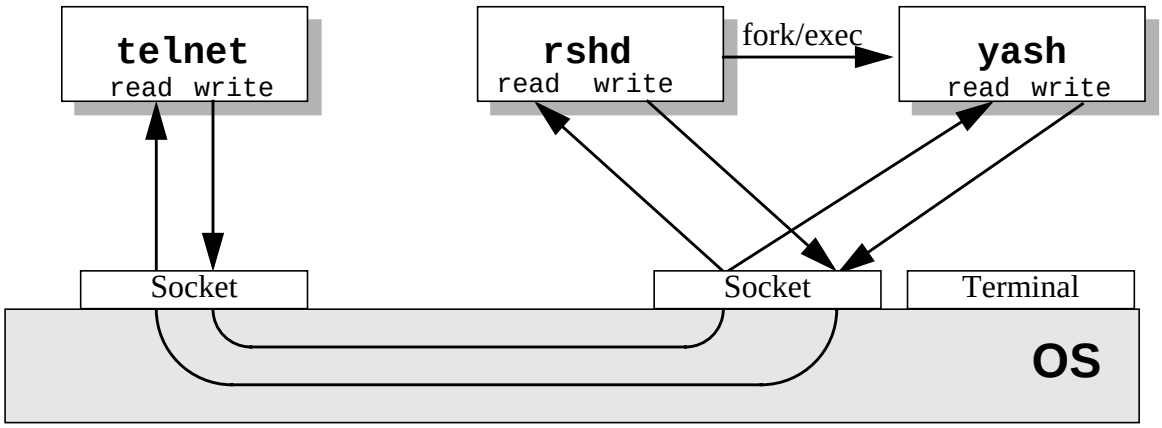
b) Fork der Shell-Prozesse

Erzeugen Sie für jede neue Socket-Verbindung einen neuen Prozeß, der diese Verbindung bearbeitet, während der Elternprozeß auf die nächste Socket-Verbindung wartet. Im Kindprozeß soll das yash-Programm von Aufgabe 4 ausgeführt werden. Die Ein- und Ausgaben der yash sollen noch nicht über den Socket gehen.



c) Ein-/Ausgabeumleitung (3 Punkte)

Leiten Sie die Ein- und Ausgabe des neuen Prozesses (der Shell) auf den Socket um (**dup(2)**). Die Ein- und Ausgaben der vom yash-Prozeß gestarteten Kommandos werden damit ebenfalls über den Socket geschickt. Denken Sie bitte daran, Filedeskriptoren in Elternprozessen zu schliessen, sobald die Filedeskriptoren dort nicht länger benötigt werden.



d) Verwaltung der Shell-Prozesse

Die Musterlösung der yash beendet sich bei Eingabe des Schlüsselwortes “exit”. Der Daemon soll das Beenden der von ihm gestarteten yash-Prozesse auf der Standardausgabe anzeigen.

Verwalten Sie die für die Verbindungsbearbeitung gestarteten yash-Prozesse. Geben Sie beim Starten eines solchen Prozesses auf der Standardausgabe des Servers die Meldung “Started shell with pid <pid>” aus, wobei <pid> für die Prozeß-ID des gestarteten Prozesses steht. Bei Beendigung dieser Prozesse soll der Exitstatus ausgegeben werden: “Exitstatus shell <pid>: <status>”
Verwenden Sie zur Verwaltung der Shell-Prozesse die joblist-Implementierung aus Aufgabe5.

Hinweis zur Lösung dieser Aufgabe:

- Sockets sind nicht Bestandteil des POSIX-Standards. Deshalb müssen Sie Ihr Programm mit dem Define `-D_XOPEN_SOURCE` übersetzen, um dem Compiler mitzuteilen, daß das Programm die im “Single Unix Standard” der “Open Group” enthaltene Funktionalität nutzt.
- Schreiben Sie ein Makefile, mit dem das gesamte Programm erstellt werden kann. Mit “make cle-

an" soll das src-Verzeichnis von unnötigen Objectfiles gesäubert werden.

Dokumentieren Sie Ihre Vorgehensweise in **rshd.doc**!

Aufgabe 2:

Shared-Memory und Semaphore

Programmieren Sie ein Erzeuger-Verbraucher-System, das einzelne Zeichen über einen Ringpuffer überträgt. Der Erzeugerprozeß liest von der Standard-Eingabe und schreibt in den Ringpuffer, der Verbraucherprozeß liest aus dem Ringpuffer und schreibt auf die Standard-Ausgabe. Im einzelnen sollen Erzeuger und Verbraucher die nachfolgende Funktionalität haben.

Erzeuger:

Der Prozeß erzeugt ein Shared-Memory-Segment und die für die Koordinierung benötigten Semaphore. Der Key für diese Datenstrukturen soll mittels ***ftok(3)*** aus dem Directory

`/proj/i4sp/<login>`

gewonnen werden. Am Anfang des Shared-Memory-Segments soll eine Verwaltungsdatenstruktur angelegt werden, die alle Daten enthält, die für die Verwaltung der Kommunikation benötigt werden. Außerdem liegt der Kommunikationspuffer im Shared-Memory. Der Puffer soll als Ringpuffer benutzt werden. Die Größe des Ringpuffers soll über eine Macro-Definition in einer include-Datei festgelegt werden. Testen Sie Ihr Programm mit den Pufferlängen 1024, 12, 2 und 1, für die Abgabe ist Puffergröße 12 einzustellen. Die Schreiboperationen auf dem Ringpuffer müssen so mittels Semaphore koordiniert werden, daß der Ringpuffer nicht überläuft und konkurrierende Zugriffe von Erzeuger und Verbraucher nicht zum Verlust von Daten führen!

Beim Start des Erzeugers überprüft dieser, ob bereits ein Erzeuger existiert und terminiert in diesem Fall. Beim Beenden wartet der Erzeuger, bis alle Daten aus dem Ringpuffer gelesen wurden, terminiert dann der Verbraucher durch das Signal SIGPIPE und löscht sowohl das Shared-Memory Segment als auch die Semaphore.

Verbraucher:

Der Verbraucherprozeß *attached* das Shared-Memory-Segment und überprüft, daß noch kein Verbraucher existiert. Existiert bereits ein Verbraucher, terminiert er mit einer Fehlermeldung. Existiert noch kein Verbraucher, schreibt er seine Prozeß-Id in die Verwaltungsstruktur, beginnt aus dem Ringpuffer zu lesen und die Daten auf dem Standardausgabekanal auszugeben. Am Ende der Datenübertragung wird er vom Erzeuger mit dem Signal SIGPIPE terminiert. Er muß sich nicht weiter um Aufräumarbeiten kümmern.

Es soll möglich sein, den Verbraucher mit dem Signal SIGINT abzubrechen. In diesem Fall meldet sich der Verbraucher ab (er signalisiert mit Prozeß-Id "-1" in der Verwaltungsstruktur, daß kein Verbraucher mehr vorhanden ist!) und terminiert. Danach soll es möglich sein einen neuen Verbraucher zu starten, der mit der Ausgabe fortfährt.

Beachten Sie hierbei, daß SIGINT an beliebiger Stelle eintreffen kann, auch während gerade Daten aus dem Ringpuffer gelesen werden. Die Verwaltungsdaten (z. B. Schreib/Lese-Indizes/Semaphore) dürfen dabei auf keinen Fall in einen inkonsisten Zustand geraten

Hinweise:

Da sowohl Erzeuger als auch Verbraucher Semphore und Shared-Memory benötigen, sind die dafür notwendigen Funktionen in je zwei eigenständige Dateien auszulagern (**sem.c** und **shm.c**). Die Prototypen dafür sollen in der Headerdatei **common.h** abgelegt werden. Erzeuger und Verbraucher sind entsprechend in **erzeuger.c** und **verbraucher.c** zu implementieren. Zur automatischen Übersetzung ist ein Makefile zu erstellen, bei dem durch "**make all"** beide Programme erzeugt werden können.

Die Koordinierung auf dem Shared-Memory (Puffer und Verwaltungsstruktur) muß in der Datei **sem.doc** knapp aber präzise dokumentieren werden.

Außerdem ist der Aufbau der Verwaltungsstruktur zu dokumentieren und zu begründen (warum wurde welche Strukturkomponente eingeführt).