Aufgabe 3: pforward

Entwerfen und programmieren Sie ein Programm **pforward**, welches TCP-Verbindungen an einem lokalen Port entgegennimmt und die Daten an einen beliebigen Port eines beliebigen Rechners weiterleitet.

Lösen Sie die Aufgabe in folgenden Schritten:

a) Eine bidirektionale Verbindung weiterleiten

Ihr Programm soll an einem frei wählbaren Port Verbindungen entgegenehmen (**socket(2), bind(2), listen(2), accept(2)**). Die lokale Portnummer, der Rechnername und die Portnummer des Zielrechners werden dem Programm als Argumente übergeben. Nach der Verbindungsannahme soll das Programm eine Verbindung zum Zielrechner aufbauen (**gethostbyname(3)**, **connect(2)**) und die Daten weiterleiten. Benutzen Sie zum Weiterleiten der Daten die zunächst vorgegebene Funktion

int forward(int fd1, int fd2).

Diese Funktion liest von beiden Filedeskriptoren Daten ein und gibt die Daten am jeweils anderen Filedeskriptor wieder aus. Sie finden die benötigte Header-Datei (**forward.h**) und die Objekt-Datei (**forward-x86_32-i4.o**) im Verzeichnis */proj/i4sos/pub/aufgabe3*. Erstellen Sie ein passendes Makefile, welches auch die Standardtargets **all** und **clean** enthält. Das mit dem vorgegebenen **forward** generierte Programm soll **pforward_i4** heissen.

b) Die Funktion “forward”

Implementieren Sie die Funktion **forward** nun selbst mit Hilfe von POSIX-Threads (**pthread_create(3)**). Die Funktion soll in einer eigenen Datei forward_pthreads.c gespeichert werden. Erweitern Sie Ihr Makefile entsprechend um ein Programm **pforward_pthreads** zu erzeugen.

c) Mehrere Verbindungen weiterleiten

Erweitern Sie das Programm nun auf die Verarbeitung von mehreren Verbindungen. Für jede entgegengenommene Verbindung soll ein Sohnprozess erzeugt werden (**fork(2)**), welcher die Daten mit Hilfe der **forward**-Funktion weiterleitet. Der Vaterprozess soll gleich wieder neue Verbindungen entgegennehmen können. Richten Sie einen Signalhandler (**sigaction(2)**) ein, welcher entstehende Zombie-Prozesse aufräumt.

d) Alternative Lösungen

Die Funktion **forward** kann auch ohne Threads implementiert werden. Es gibt noch zwei weitere Möglichkeiten. Zum einen könnte man für die jeweilige Übertragungsrichtung einen eigenen Prozess erzeugen (anstelle von Threads), zum anderen bietet die Unix-Systemschnittstelle einen spezialisierten Systemcall (**poll(2)**) zum Überwachen von mehreren Filedeskriptoren.

Implementieren Sie zum Vergleich die Funktion **forward** mit diesem Systemcall. Die Funktion soll in einer eigenen Datei (**forward_poll.c**) abgegeben werden und die gleiche Schnittstelle wie die pthread-Lösung (**forward.h**) verwenden. Passen Sie Ihr Makefile so an, dass nun auch noch ein Programm **pforward_poll** erzeugt wird. Die drei in dieser Aufgabe erstellten Varianten des **pforward** Programms sollen alle vom **all** Target erzeugt werden.

Bei Fragen zur Aufgabenstellung bitte Mail an *i4sos@informatik.uni-erlangen.de*

e) Diskussion

Nennen Sie (**doc/forward.txt**) jeweils Ihnen bekannte Vor- und Nachteile aller drei möglichen Implementierungen und begründen Sie, welche Implementierung Sie bevorzugen würden.

Hinweise zur Lösung dieser Aufgabe:

- Die **pforward_*** Targets in Ihrem Makefile sollen sich lediglich darin unterscheiden, dass mit dem jeweils korrekten forward-Modul gebunden wird.
- Zum Testen können Sie Ihr Programm wie folgt starten:
  pforward <freigewählter port> wwwproxy.informatik.uni-erlangen.de 8080
  und im Webbrowser den lokalen Rechner mit der freigewählten Portnummer als Proxy eintragen.
- Sockets sind nicht Bestandteil des POSIX-Standards. Deshalb müssen Sie Ihr Programm mit der Option `-D_XOPEN_SOURCE=500` übersetzen.
- Die pthread-Funktionen sind in einer speziellen Funktionsbibliothek (libpthread) zusammengefasst, die Sie beim Compilieren bzw. Binden Ihres Programms mit angeben müssen (Option `-lpthread`).