Aufgabe 4: fdserver

Die folgende Aufgabe entspricht in etwa der Aufgabenstellung aus der SOS1-Klausur aus dem Sommersemester 2005. Die original Aufgabenstellung finden Sie unter
http://www4.informatik.uni-erlangen.de/Lehre/SS05/V_SOS1/Pruefung/klausuren.shtml
Fragen zu Posix-Threads werden in der Rechneruebung noch behandelt.

a) fdserver

Implementieren Sie ein Server-Programm **`fdserver`**, das auf Anfrage den Inhalt von Dateien oder Directories über eine TCP/IP-Verbindung ausliefert. Wir gehen davon aus, dass dieser Server mit einer sehr hohen Anfragerate belastet wird, deshalb sollen die Anfragen nicht in separaten Sohnprozessen, sondern durch bereitstehende Threads behandelt werden. Das **`fdserver`**-Programm erhält als Aufrufparameter die Nummer des Ports, auf dem Verbindungen angenommen werden.
Das Programm soll folgendermaßen arbeiten:

- Zunächst werden vom Haupt-Thread die bereitzuhaltenden Threads gestartet. Sie erledigen ihre Aufgabe in der Funktion **`serve`**. Bedenken Sie, dass jeder Thread seinen eigenen Stack hat (die dort liegenden Variablen sind also Thread-lokal), aber alle Threads ein gemeinsames Datensegment haben.
- Die Zahl der Threads wird im Quellcode über ein Makro **`NTHREAD`** mit Wert 25 festgelegt.
- Anschließend nimmt das Programm Verbindungen über beliebige IP-Adressen auf dem angegebenen Port an. Bis zu 20 Verbindungswünsche sollen anstehen können, bevor Verbindungswünsche abgewiesen werden.
- Wurde eine Verbindung angenommen, wird einer der bereitstehenden Threads mit der Bearbeitung beauftragt. Die Beauftragung erfolgt, indem der Socket-Deskriptor der angenommenen Verbindung in einen Ringpuffer (der Größe **`NTHREAD`**) eingetragen wird.
- Alle Threads warten in der Funktion **`serve`** an diesem Ringuffer. Nachdem ein Socket-Deskriptor eingetragen wurde, wird ein Thread deblockiert, der dann die Verbindung bearbeitet.
- Sollte der Ringpuffer voll sein, werden so lange keine Verbindungen angenommen.
- Die Koordinierung zwischen Haupt-Thread und den `serve`-Threads erfolgt durch zählende Semaphore. Implementieren Sie hierzu ein Semaphor-Modul `sem.c` mit den Funktionen `sem_init`, `P` und `V` (Schnittstellen siehe Datei `sem.h` auf der nächsten Seite).
- Beachten Sie auch, dass zwar nur der Haupt-Thread Aufträge in den Ringpuffer einträgt, aber evtl. mehrere Threads gleichzeitig Aufträge entnehmen wollen.
- Ein Client sendet über die Verbindung die Namen von Dateien oder Directories (komplette Pfandnamen, String mit max. 1024 Byte), deren Inhalt er geliefert bekommen möchte.
- Der **`serve`**-Thread entnimmt den Socketdeskriptor aus dem Ringpuffer, erzeugt einen **`FILE`**-Zeiger daraus, liest den Datei-/Directory-Namen von der Verbindung und ruft die Funktion **`void handle(FILE *socket, char path[])`** zur weiteren Bearbeitung auf. Anschließend wird der Socket geschlossen und der nächste Auftrag erwartet.
- In der **`handle`**-Funktion wird ermittelt, ob **`path`** eine reguläre Datei oder ein Directory ist. Zur weiteren Bearbeitung wird entweder **`fhandle(FILE *socket, char path[])`** oder **`dhandle(FILE *socket, char path[])`** aufgerufen (bei anderen Dateitypen oder Fehlern bei der Auftragsbearbeitung wird eine aussagekräftige Fehlermeldung auf die Socketverbindung ausgegeben und die Bearbeitung dieses Auftrags abgebrochen).
- Die **`fhandle`**-Funktion gibt einfach den Inhalt der regulären Datei auf den Socket aus, die **`dhandle`**-Funktion gibt die Namen aller in dem Directory verzeichneten Dateien (jeweils auf einer Zeile) auf den Socket aus.

Im Verzeichnis */proj/i4sos/pub/aufgabe4* finden Sie ein Gerüst für das beschriebene Programm. In den Kommentaren sind nur die wesentlichen Aufgaben der einzelnen, zu ergänzenden Programmteile beschrieben, um Ihnen eine Leitlinie zu geben.

b) Makefile

Schreiben Sie ein Makefile zum Erzeugen des fdserver-Programms. Ein Aufruf von *"make fdserver"* soll das Programm erzeugen, ein Aufruf von *"make clean*" soll das **fdserver**-Programm und evtl. erzeugte **`.o`**-Dateien entfernen.