Institut für Informatik

der Ludwig-Maximilians-Universität München

Systempraktikum – Wintersemester 2009/2010

Prof. Dr. Dieter Kranzlmüller
Dr. Thomas Schaaf, Dr. Nils gentschen Felde

Blatt 7— Vertiefung & Projekt IV: Dateiübertragung

Abgabedatum theor. Aufgaben	Abgabedatum prakt. Aufgaben	Deadline Projektaufgaben
—	—	22.01.

Projekt-Aufgaben (Blatt 7)

Aufgabe PROJ-7-1

[Module **`protocol`, `client`**] Stdin-Protokoll

Bisher sendet der Client alle Benutzereingaben einfach an den Server weiter. Allerdings gibt es auch Funktionen im Client, die keine Serverinteraktion erfordern. Da es sich hier um ein Peer-to-Peer Filesharing-System handelt, läuft der Download nur zwischen zwei Clients ab, ohne dass der Server dazu nötig ist. Deshalb wird jetzt das letzte Protokoll eingeführt: das `stdin-protocol`. Es dient dazu, die Benutzereingaben zu überprüfen und Kommandos ohne Serverineraktion direkt auszuführen. Alle sonstigen Eingaben sollen per `defaultAction` einfach an den Server weitergeleitet werden. So können am Server leicht neue Funktionalitäten eingebaut werden, ohne dass der Client geändert werden müsste.

a. Erstellen Sie im `protocol`-Modul die Dateien `stdin_protocol.c` und `stdin_protocol.h` und passen Sie das `Makefile` an. Verändern Sie dann den Client so, dass Eingaben, die am Consoler eingehen, durch das `stdin-protocol` geleitet werden, statt direkt an den Server zu gehen. Die folgenden Teilaufgaben sind die Aktionen des `stdin_protocol`.

b. `int passOnAction(struct actionParameters *ap, union additionalActionParameters *aap);`
Dies ist die Standaraktion `defaultAction` im Rahmen der Spezifikation des Protokolls. Die Eingabe wird einfach an den Server weitergschickt. Beachten Sie, dass Sie die Eingabe unter Umständen rekonstruieren müssen.

c. `int stdin_resultsAction(struct actionParameters *ap, union additionalActionParameters *aap);`
Diese Funktion gibt die aktuellen Resultate einer Suche aus dem `results`-Array aus.

d. `int stdin_exitAction(struct actionParameters *ap, union additionalActionParameters *aap);`
Diese Routine schließt das Kommunikations-Socket von Client-Seite aus, ohne dem Server ein `QUIT`-Kommando zu senden.

[0] Stand: 4. Januar 2010

e. `int stdin_helpAction(struct actionParameters *ap, union additionalActionParameters *aap);`
Diese Funktion zeigt, die Hilfetexte aller spezifizierten Aktionen des Protokolls an und sendet danach das `HELP`-Kommando an den Server, um auch dessen Hilfetexte auszugeben.

f. `int stdin_downloadAction(struct actionParameters *ap, union additionalActionParameters *aap);`
...wird durch Eingabe von `DOWNLOAD <nr>` aufgerufen. Das Argument `<nr>` bezeichnet dabei die Nummer des Array-Elements der Resultate (Erinnerung: diese wird bei Resultatausgabe mit ausgegeben). Diese Funktion brauchen Sie noch nicht in dieser Aufgabe implementieren, sie wird später mit Hilfe der Funktion `advFileCopy()` implementiert.

Aufgabe PROJ-7-2

[Module **`client`, `protocol`, `connection`**] Passives Socket im Client, handleUpload

Damit sich ein Client zu einem anderen verbinden kann, um von diesem eine Datei direkt herunterzuladen, muss auch jeder Client über ein passives Socket verfügen, auf dem er Verbindungen von anderen Clients entgegennehmen kann.

a. Legen Sie im Setup-Bereich des Clients ein neues passives Socket an.

b. Legen Sie im Server-Protokoll eine neue Aktion `int portAction(struct actionParameters *ap, union additionalActionParameters *aap);` an. Diese Aktion erwartet als einziges Argument eine Zahl, die dem Port des passiven Sockets des Clients entspricht. Wandeln Sie das Argument-Token in eine Zahl um, und weisen Sie sie dem noch unbenutzten Feld `ap->comport` zu.

c. Verwenden Sie jetzt das Feld `ap->comport`, um in der Funktion `recvFileList()` im `connection`-Modul neben der IP des Clients auch seinen Port zu setzen.

d. Senden Sie nun das Port-Kommando automatisch vom Client an den Server, sobald der Client sich verbunden und sein passives Socket angelegt hat. Achten Sie darauf, dass das Port-Kommando immer vor dem ersten Filelist-Kommando ausgeführt werden muss. Es empfiehlt sich, auch das Filelist-Kommando automatisch auszulösen.

e. Fügen Sie den neuen Filedeskriptor des passiven Sockets in ihren `poll()`-Mechanismus in der Hauptschleife des Clients ein.

f. Wie auch im Server soll bei `POLLIN` auf dem passiven Socket zuerst die neue Verbindung akzeptiert werden (`accept`) und dann ein neuer Prozess für diese Verbindung erzeugt werden (`fork()`). Rufen Sie im neuen Prozess die Funktion `handleUpload()` auf (siehe nächste Teilaufgabe), und achten Sie darauf, dass der neue Prozess danach mit den Rückgabewerten -2 oder -3 zurückkehrt und so in der Hauptschleife aufgeräumt wird.

g. Schreiben Sie im `connection`-Modul eine neue Funktion `int handleUpload(int upfd, int confd, struct actionParameters *ap)`. Diese Funktion soll zuerst eine Zeile, die den Dateinamen der angeforderten Datei enthält, aus der akzeptierten Verbindung (Parameter `int upfd`) lesen. Verwenden Sie dazu die `tokenizer`-Funktion `getTokenFromStream()`. Öffnen Sie die Datei, und rufen Sie dann die Funktion `advFileCopy()` auf (siehe folgende Aufgaben). Der Parameter `int confd` enthält den Filedeskriptor des Consolers, damit Sie Statusmeldungen ausgeben können.

Hinweise

- Die Dateiübertragung wird hier möglichst einfach gestaltet, um den Implementierungsaufwand gering zu halten. Die Übertragung läuft folgendermaßen ab:

    1. Der anfordernde Client benutzt die Verbindungsdaten (IP und Port) eines Suchresultats, um eine Verbindung zu dem passiven Socket des Clients aufzubauen, auf dem die gewünschte Datei liegt.

2. Der bereitstellende Client nimmt die Verbindung an und wartet in der Funktion `handleUpload()` auf den Namen der angeforderten Datei.
3. Sobald die Verbindung aufgebaut ist, schickt der anfordernde Client den Dateinamen in das Socket. Direkt danach betritt er die Funktion `advFileCopy()`, um die Datei zu empfangen.
4. Nachdem der bereitstellende Client den Dateinamen empfangen hat, öffnet er die angeforderte Datei und startet ebenfalls die Funktion `advFileCopy()`, um die Datei zu senden.
5. Bei der Dateiübertragung selbst wird der Inhalt der Datei einfach in das Socket kopiert.
6. Das Ende der Dateiübertragung wird durch das Schließen des Sockets durch den bereitstellenden Client angezeigt. Um sicherzugehen, dass kein Fehler aufgetreten ist, muss nun noch die Zahl der empfangenen Bytes mit der Dateigröße aus dem Suchresultat verglichen werden.

Aufgabe PROJ-7-3

[Modul **`connection`**] advFileCopy

Die Funktion `advFileCopy()` dient dazu, Daten zwischen zwei Filedeskriptoren zu kopieren. Da Sockets wie auch Dateien über Filedeskriptoren angesteuert werden können, eignet sie sich sowohl zum Senden wie auch zum Empfangen von Dateien. Die Funktion `advFileCopy()` ist dafür ausgelegt, in einem eigenen Prozess ausgeführt zu werden. So kann sie die zusätzliche Funktionalität bereitstellen, bei einem eingehenden `SIGUSR1`-Signal den aktuellen Fortschritt des Kopiervorgangs über den Consoler auszugeben.

a. Legen Sie die Funktion `advFileCopy()` im `connection`-Modul an. Ihre Signatur ist:
```
int advFileCopy(int destfd, int srcfd, unsigned long size, const char *name, int semid, int logfd, int sigfd, int confd);
```
Die Parameter haben folgende Bedeutung:
   - `int destfd`: der Filedeskriptor, in den die eingehenden Daten geschrieben werden sollen
   - `int srcfd`: der Filedeskriptor, aus dem Daten gelesen werden sollen
   - `unsigned long size`: die Dateigröße. Sie dient zur Berechnung des Fortschritts des Kopiervorgangs und am Ende der Übertragung zum Feststellen von Fehlern.
   - `const char *name`: der Dateiname. Er wird nur für Ausgaben durch den Consoler und den Logger benutzt.
   - `int semid`: die Semaphoren-ID. Sie ist nötig für die Funktionen `logmsg()` und `consolemsg()`.
   - `int logfd`: der Filedeskriptor des Loggers
   - `int sigfd`: ein Filedeskriptor für die Signale `SIGINT`, `SIGQUIT` und `SIGUSR1`
   - `int confd`: der Ausgabe-Filedeskriptor des Consolers

b. Verwenden Sie eine Schleife mit `poll()`-Mechanismus auf die Filedeskriptoren `sigfd` und `srcfd`.

c. Bei `POLLIN` auf `srcfd` lesen Sie einmal Daten und schreiben Sie diese dann nach `destfd`. Verwenden Sie dazu die bekannten Puffer und Ein-/Ausgabefunktionen aus dem `util`-Modul.

d. Bei `POLLIN` auf `sigfd` brechen Sie für die Signale `SIGINT` und `SIGQUIT` die Dateiübertragung ab. Für das Signal `SIGUSR1` geben Sie über den Consoler den Fortschritt der Dateiübertragung in Prozent an.

e. Nachdem das Lesen abgeschlossen ist (`readToBuf()` liefert 0 zurück), vergleichen Sie die Zahl der gelesenen Bytes mit dem Parameter `size`. Ein Fehler liegt vor, wenn die Werte nicht übereinstimmen.

f. Achten Sie wie immer auf ausgiebige Fehlerbehandlung. Die Funktion `advFileCopy()` soll im Erfolgsfall 1, im Fehlerfall -1 zurückgeben. Die Filedeskriptoren sollen nicht geschlossen werden, dies sollen die aufrufenden Funktionen übernehmen.

Aufgabe PROJ-7-4

[Module **client**, **protocol**] Dateiübertragung

In dieser Aufgabe soll nun die soeben geschriebene Funktion `advFileCopy()` integriert werden, so dass die Dateiübertragung funktioniert. Die Aufgaben 7-1 und 7-2 erwähnen bereits die Stellen, an der die Funktion aufgerufen werden soll.

a. Für den anfordernden Client müssen Sie nun im Stdin-Protokoll die Aktion `downloadAction()` implementieren. Führen Sie zuerst den nötigen Fork durch. Verwenden Sie den Parameter des Download-Kommandos (Index des `results`-Arrays), um den entsprechenden Eintrag im Suchresultat-Array zu finden. Öffnen Sie eine neue Datei zum Schreiben, den Dateinamen finden Sie im Suchresultat. Verbinden Sie sich daraufhin mittels `connectSocket()` zu dem im Suchresultat angegebenen Client. Fordern Sie, sobald die Verbindung steht, die Datei an, indem Sie den Dateinamen als eine Zeile in das Socket schreiben. Erzeugen Sie anschließend noch einen neuen Signal-Filedeskriptor, und rufen Sie dann die Funktion `advFileCopy()` auf. Geben Sie die benutzten Ressourcen wieder frei, und achten Sie darauf mit korrekten Rückgabewerten aus der Aktion zurückzukehren.

b. Integrieren Sie die Funktion `advFileCopy()` in die Funktion `handleUpload()` (siehe auch Aufgabe 7-2). Da Sie hier schon beim Akzeptieren der Verbindung einen `fork()` ausgeführt haben, brauchen Sie nur noch einen neuen Signal-Filedeskriptor erzeugen bzw. anzupassen. Rufen Sie danach `advFileCopy()` auf.

Hinweise

- In beiden Fällen wird die Funktion `advFileCopy()` in einem eigenen Prozess ausgeführt. Die Dateiübertragung ist naturgemäß ein möglicherweise lange dauernder Vorgang, der die Interaktion im Client nicht behindern soll und deswegen in einen eigenen Prozess ausgelagert werden muss.
- Die Signal-Filedeskriptoren (Funktion `getSigfd()`) der Prozesse sollen für die von `advFileCopy()` bearbeiteten Signale eingerichtet werden.

Aufgabe PROJ-7-5

[Module **client**, **util**] Kindprozess-Array

In dieser Aufgabe soll ein Array erstellt und gefüllt werden, das Buch führt über alle erstellten Kindprozesse des Client-Prozesses. Dafür vorgesehen ist das Feld `struct array *cpa` der `struct clientActionParameters`. Dieses Array muss nicht in einem Shared Memory liegen, da es ausschließlich vom Client-Hauptprozess zugegriffen werden soll und alle Forks ebenfalls in diesem Prozess ausgeführt werden. Binden Sie den Code der Datei `childprocesses_util.h` von der Systempraktikums-Homepage in Ihre `util.h` ein.

Erstellen Sie jetzt die drei Funktionen (sie arbeiten alle auf einem Array aus `childProcess`-Strukturen), und verwenden Sie diese anschließend im Projekt:

a. `struct array *addChildProcess(struct array *cpa, unsigned char type, pid_t pid);`
... fügt dem Array `cpa` einen neuen Kindprozess der PID `pid` und des Typs `type` hinzu.

b. `int remChildProcess(struct array *cpa, pid_t pid);`
... löscht den Kindprozess mit der PID `pid` aus dem Array.

c. `int sendSignalToChildren(struct array *cpa, unsigned char type, int sig);`
... sendet allen Kindern des Typs `type` das Signal `sig`.

d. Initialisieren Sie das Array, und fügen Sie nach jedem `fork()` im Client einen neuen Kindprozess mittels `addChildProcess()` hinzu. Benutzen Sie `remChildProcess()` in der Signalbehandlung der Client-Hauptschleife nach einem `SIGCHLD`.

Hinweise

- Der Client-Hauptprozess sollte alle seine Kinder „aufräumen“, bevor er sich selbst beendet, da die Konsole sonst eventuell nicht richtig wiederhergestellt wird (wenn der Consoler länger läuft als der Hauptprozess). Um dies sicherzustellen, eignet sich das Kindprozess-Array.

Aufgabe PROJ-7-6

[Module **`protocol`**] showAction

Damit der Benutzer sich über den Fortschritt seiner Down- und Uploads informieren kann, soll er über das Kommando “`SHOW`” alle aktuell aktiven Down- und Uploads anzeigen lassen können. Erstellen Sie dazu im Stdin-Protokoll die Aktion `int stdin_showAction(struct actionParameters *ap, union additionalActionParameters *aap);`. Senden Sie darin das Signal `SIGUSR1` an alle Down- und Uploadprozesse. Verwenden Sie die Funktion `sendSignalToChildren()`.