Institut für Informatik

der Ludwig-Maximilians-Universität München

Systempraktikum – Wintersemester 2010/2011

Prof. Dr. Dieter Kranzlmüller
Dr. Nils gentschen Felde, Christof Klausecker, Johannes Watzl

Blatt 6— Vertiefung & Projekt II: Semaphoren, Named Pipes, Protokolle, Nutzerinteraktion

Abgabedatum theor. Aufgaben	Abgabedatum prakt. Aufgaben	Deadline Projektaufgaben
—	—	27.12.

Projekt-Aufgaben (Blatt 6)

Hinweise

- **Achtung: Beginnen Sie zeitnah mit der Implementierung der Projektaufgaben! Unterschätzen Sie nicht den Umfang der Aufgaben auf diesem Blatt!**
- Folgende Dateien werden auf der Praktikumswebseite in einem .tgz-Archiv zur Verfügung gestellt:
  - `sysprakSourcesWS0910/util/util_setFdNonblock.c`
  - `sysprakSourcesWS0910/util/util_sem_defines.h`
  - `sysprakSourcesWS0910/consoler/Makefile`
  - `sysprakSourcesWS0910/consoler/consoler.c`
  - `sysprakSourcesWS0910/consoler/consoler.h`
  - `sysprakSourcesWS0910/protocol/protocol.c`
  - `sysprakSourcesWS0910/protocol/protocol.h`
  - `sysprakSourcesWS0910/protocol/server_protocol.c`
  - `sysprakSourcesWS0910/protocol/server_protocol.h`

Aufgabe PROJ-6-1

[Modul **`consoler`**]

Das Modul `consoler` wird Ihnen auf der Praktikumsseite komplett zur Verfügung gestellt. Bitte laden Sie es herunter und binden Sie es in die Versionsverwaltung Ihres Projektes ein. Der Consoler wird nur im Client verwendet.

Um den Consoler in ein Programm einzubinden, erstellen Sie zunächst zwei namenlose Pipes mittels `pipe()`, führen Sie dann den Befehl `fork()` aus und starten Sie im Kindprozess die Funktion `consoler()` mit den korrekten Seiten der beiden Pipes. Achten Sie darauf, dass der geforkte Prozess **nicht** nach der Rückkehr der

[0]Stand: 15. Dezember 2010

Funktion `consoler` in den Code für den Client hineinläuft. Schließen Sie außerdem die jeweils nicht benötigten Enden der Pipes im Consoler-Fork wie auch im Programm selbst sobald möglich.

Hinweise

- Dieses Modul dient dazu, eine sehr einfache asynchrone Nutzerinteraktion zu ermöglichen. Asynchron in diesem Zusammenhang bedeutet, dass jederzeit Meldungen auftreten können, die trotz einer vorhandenen, unvollständigen Nutzereingabe sofort ausgegeben werden sollen (zum Beispiel "Download abgeschlossen"). Deshalb muss nach der Ausgabe die Kommandozeile mit der, möglicherweise vorhandenen, unvollständigen Nutzereingabe wiederhergestellt werden. Im normalen Terminalmodus, der zeilenweise funktioniert (d.h. Eingaben werden erst nach Drücken der Return-Taste auf `STDIN_FILENO` verfügbar) ist dies schwierig zu realisieren. Deswegen wird Ihnen das Modul Consoler zur Verfügung gestellt. Es läuft, ähnlich dem Logger, als eigener Prozess. Zuerst wird das Terminal in den Raw-Modus versetzt, dann eine Schleife gestartet, die einzelne Zeichen von der Konsole liest und diese in einem Buffer zwischenspeichert. Ähnlich wie bei der zeilenweisen Konsole, wird nach Drücken der Return-Taste die getippte Zeile über eine Pipe an den Hauptprozess geschickt. Ausgaben emfängt der Consoler ebenfalls über eine Pipe, aus der er mithilfe der Tokenizer-Funktion `getTokenFromStreamBuffer()` Zeilen ausliest, diese dann auf dem Terminal ausgibt und danach die Eingabe wiederherstellt. Im Hauptprozess sind danach einfach die Filedeskriptoren der zwei Pipe-Enden anstatt `STDIN_FILENO` und `STDOUT_FILENO` zu verwenden.
- Sie sollten darauf achten, dass der Consoler vor dem Haupt-Client beendet wird, da sonst ihre Konsole nicht in den Ursprungszustand zurückversetzt werden könnte (und damit unbrauchbar wird). Dazu empfiehlt es sich, den Rückgabewert von `fork()` zu speichern, und nach dem Schließen aller offenen Pipe-Enden im Hauptclient, mittels `waitpid()` auf den Prozess zu warten.

Aufgabe PROJ-6-2

[Modul **`client`**]

Legen Sie das Modul für den Client an, erstellen Sie im Hauptmakefile ein Target für den Client und erstellen Sie im neuen Modul eine `client.c`-Datei mit einer `main()`-Funktion, in der Sie eine (client-spezifische) Konfigurationsdatei einlesen. Richten Sie in der `main()`-Funktion zuerst den Logger (mit der Log-Datei aus ihrer `struct config conf`) und dann den Consoler (siehe vorige Aufgabe) ein. Treten Sie danach in eine Endlosschleife ein, in der Sie einfach die vom Consoler über die Pipe gelieferten Eingaben dem Logger übergeben und senden Sie dann eine Meldung mit der Anzahl der getippten Zeichen an den Consoler zurück, so dass sie in der Konsole erscheint.

Aufgabe PROJ-6-3

[Modul **`protocol`**]

Das Modul `protocol` ist eines der Kernbestandteile des Projekts. Es wird eingesetzt, um die Anweisungen einer Kontrollverbindung zu erkennen und daraufhin eine Aktion auszuführen. Das Kernmodul `protocol.c` enthält selbst kein Protokoll, sondern die Funktionen zum Zerlegen des Datenstroms, zum Erkennen eines Kommandos aus dem Datenstrom mit Hilfe der `struct protocol` und zum schließlichen Ausführen einer Aktion. Das Kernmodul wird Ihnen vorgegeben, laden Sie es herunter und binden Sie es ins Projekt ein. Gehen Sie den Code aufmerksam durch und machen Sie sich mit den Strukturen und den Funktionen des Moduls vertraut.

Hinweise

- `typedef int(*action) (struct actionParameters *ap, union additionalActionParameters *aap);`
  Diese Typdefinition aus der `protocol.h` definiert einen Typ `action`, der einen Zeiger auf eine Funktion

darstellt. Die Funktionssignatur ist dabei: `int <FUNKTIONSNAME>(struct actionParameters *ap, union additionalActionParameters *aap);`. Durch diese Verzeigerung der Aktions-Funktionen eines Protokolls, kann eine einfache Struktur erstellt werden, die ein Schlüsselwort, welches die Aktion auslösen soll, mit ihrer Aktions-Funktion verknüpft.

- `struct action`
  Diese Struktur beschreibt genau eine Aktion eines Protokolls. Sie besteht aus einem Schlüsselwort für eine Aktion (`actionName`), einer kurzen Beschreibung (`description`) und dem Zeiger auf die auszuführende Funktion.

- `struct protocol`
  Diese Struktur beschreibt ein Protokoll und besteht neben einer Standardaktion (`defaultAction`) aus einem Array aller Aktionen die das Protokoll umsetzt (`actions`). Da `actions` ein Array mit 32 Einträgen ist, muss für Protokolle mit weniger Aktionen zusätzlich das Feld `actionCount` eingeführt werdem, welches die Anzahl der Aktionen angibt.

- `struct actionParameters`
  Diese Struktur enthält alle für das Protokoll (bzw. dessen Aktionen) wichtigen Ressourcen, wie Puffer und Kommunikations-Filedeskriptor (`comfd`). Außerdem enthält sie noch den Filedeskriptor des Loggers (`logfd`), eine Semaphoren-ID (`semid`) und den Filedeskriptor über den Signale empfangen werden (`sigfd`).

- `struct serverActionParameters`
  Diese Struktur enthält weitere Server-Ressourcen, wie das Shared Memory für die Dateiliste.

- `struct clientActionParameters`
  Diese Struktur enthält weitere Client-Ressourcen, wie die zum Consoler gehende Ausgabe-Pipe `outfd`, das Suchresultate-Array `results` und ein Array welches die Client-Kindprozesse speichert (`cpa`).

- `union additionalActionParameters`
  Diese Union vereinigt `struct serverActionParameters` und `struct clientActionParameters` um eine einheitliche Struktur für Aktionen verwenden zu können.

- `int processIncomingData()`
  Diese Funktion wird aufgerufen, wenn auf dem Kommunikations-Filedeskriptor neue Daten anliegen. Sie liest einmal aus dem Filedeskriptor und verwendet den Tokenizer um eine oder mehrere Zeilen zu extrahieren. Daraufhin wird die Funktion `processCommand()` aufgerufen, die sich um die Abarbeitung der neu empfangenen Zeile kümmert.

- `int processCommand()`
  Diese Funktion erwartet im Puffer `ap->comline` ein Kommando (entspricht einer Zeile). Dieses zerlegt sie mittels `getTokenFromBuffer()` weiter und benutzt `ap->comword` zum Ablegen des Wortes. Schließlich verwendet sie die Funktion `validateToken()` und führt die zurückgegebene Aktion aus.

- `int validateToken()`
  Diese Funktion vergleicht den Inhalt des Buffers `ap->comword` mit allen Schlüsselwörtern einer Protokollstruktur. Wird eine Übereinstimmung gefunden, so wird der Zeiger auf diese Aktion zurückgegeben; wird keine gefunden, so liefert die Funktion den Zeiger auf die `defaultAction` des Protokolls zurück.

- `int reply()`
  Diese Funktion dient dem Server zum Senden von Antworten an den Client. Die Antwort wird aus einem 3 stelligen Zahlencode (ähnlich den http-Codezahlen) und der eigentlichen Nachricht zusammengesetzt. Außerdem wird die Nachricht im Loglevel Debug geloggt.

- Rückgabewerte
  Die Aktions-Funktionen haben folgende Rückgabewerte: 1 - Erfolg (weitermachen); 0 - Kommunikation beendet (`comfd` geschlossen, beenden); -1 Fehler (beenden); -2 geforkter Kindprozess kehrt mit Erfolg zurück (beenden ohne endgültig aufzuräumen); -3 geforkter Prozess kehrt mit Fehler zurück (beenden ohne endgültig aufzuräumen). Wenn ein geforkter Prozess zurückkehrt, so sollen IPC-Objekte wie Shared Memory oder Semaphoren-ID's nicht gelöscht werden, da das Programm weiterläuft und andere Prozesse diese noch benötigen. Nur der Hauptserver oder Hauptclient darf diese ganz zum Schluss entfernen. Die Funktionen des Protokollmoduls geben dieselben Rückgabewerte zurück wie die Actions.

Aufgabe PROJ-6-4

[Modul **protocol**] Server-Protokoll

Laden Sie sich die vorgegebenen Dateien `server_protocol.h` und `server_protocol.c` herunter und fügen Sie diese in das Projekt ein. Vervollständigen Sie anschließend die Aktionen in `server_protocol.c`. Verwenden Sie für Antworten an den Client die Funktion `reply()`.

a. `int statusAction(struct actionParameters *ap, union additionalActionParameters *aap);`
   Diese Aktion gibt den Status des Servers zurück. Geben Sie im Moment noch einfach eine beliebige Meldung an den Client zurück - später soll zum Beispiel zurückgegeben werden, wie viel Platz noch in der Dateiliste des Servers ist.

b. `int helpAction(struct actionParameters *ap, union additionalActionParameters *aap);`
   Diese Aktion soll durch die Protokoll-Struktur laufen und zu jedem Schlüsselwort die Beschreibung (`description`) an den Client senden.

Sie können sich bei der Implementierung an den folgenden, komplett vorgegebenen Aktionen orientieren:

- `int unknownCommandAction(struct actionParameters *ap, union additionalActionParameters *aap);`
  Die Standardaktion des Protokolls ist, dass der Befehl nicht verstanden wurde. Dies wird dem Client mitgeteilt.
- `int quitAction(struct actionParameters *ap, union additionalActionParameters *aap);`
  Diese Aktion beendet die Hauptschleife im Server, indem eine 0 zurückgegeben wird - derselbe Wert wird auch von `processIncomingData()` zurückgegeben, wenn `read()` 0 zurückgibt. In diesem Fall soll die Hauptschleife im Server abgebrochen werden (siehe nächste Aufgabe).

Aufgabe PROJ-6-5

[Modul **server**] Server Hauptschleife

Erweitern Sie den Server um eine Hauptschleife. Erzeugen und füllen Sie die beiden Strukturen `struct actionParameters` und `struct serverActionParameters`. Um die Shared Memories, Signale und den Inhalt von `serverActionParameters` brauchen Sie sich noch nicht zu kümmern, legen Sie aber die Puffer an und setzen Sie `logfd`. Benutzen Sie außerdem die vorgegebenen `APRES_*`-Werte, um in der Variable `usedres` ein Bitmuster der belegten Ressourcen anzulegen. Sie können dieses "Setup" auch in eine Funktion des Server-Moduls ausgliedern, um die `main()`-Funktion übersichtlicher zu halten.

Legen Sie anschließend zwei Named Pipes (`mkfifo()`) an, und weisen Sie in ihrer `struct actionParameters` eine Leseseite für die Client-zu-Server Kommunikation `c2s` zu, eine Schreibseite für die Server-zu-Client Kommunikation `s2c`.

Treten Sie nach dem Öffnen der Named Pipes in eine Endlosschleife ein, in der Sie mit den Funktionen aus dem Protokollmodul (`processIncomingData()`) auf der Named Pipe eingehende Daten bearbeiten.

Aufgabe PROJ-6-6

[Modul **client**] Client verbindet sich zum Server, Poll

In dieser Aufgabe sollen Sie die Funktion `poll()` einsetzen. Diese Funktion dient dazu, mehrere Filedeskriptoren gleichzeitig auf bestimmte Ereignisse (hier vor allem `POLLIN`) zu überwachen und dabei zu blockieren, bis ein Ereignis eintritt. Ist ein Ereignis eingetreten, kann man durch einfache bitweise Verknüpfung feststellen, welcher

Filedeskriptor welches Ereignis ausgelöst hat und darauf reagieren. Machen Sie sich selbständig genauer mit der Funktion `poll()` in der man-Page vertraut.

Erweitern Sie die Hauptschleife Ihres Clients wie folgt:

1. Öffnen Sie vor der Schleife die vom Server angelegten Named Pipes und weisen sie jeweils die Lese- und Schreibseiten so zu, dass eine Kommunikation mit dem Server in beide Richtungen möglich ist.
2. Sie haben jetzt zwei Filedeskriptoren auf denen Daten eingehen können, dies ist zum einen die Pipe mit den Benutzereingaben vom Consoler und zum anderen die Named Pipe mit Serverantworten. Legen Sie ein Array der Struktur `struct pollfds` für diese beiden Filedeskriptoren an.
3. Benutzen Sie nun in der Client-Hauptschleife den `poll()`-Aufruf mit dieser Struktur um festzustellen auf welchem der beiden Filedeskriptoren Daten eingehen.
4. Statt, wie in der vorigen Aufgabe, die eingetippten Kommandos zu loggen, senden Sie sie an den Server weiter.
5. Empfangen Sie Daten vom Server, so geben Sie diese einfach über den Consoler auf der Konsole aus.

Hinweise

- Dieses Vorgehen ermöglicht es, den Client asynchron von Serverantworten zu halten. Das heißt der Benutzer muß nicht warten, bis der Server die Anfrage bearbeitet hat und eine Antwort schickt, bevor er weitertippen kann. Da Suchanfragen an den Server unter Umständen länger dauern können und da in einem Peer-to-peer Programm viele Operationen ohne den Server ausgeführt werden können, ist dies sinnvoll.
- Filedeskriptoren sind bei ihrer Erstellung blockierend. Das heißt, Systemaufrufe (z.B. `read()`) auf ihnen blockieren das Programm bis Daten vorliegen oder ein Fehler oder eine Unterbrechung auftritt. Deswegen sollte man bei der Verwendung von `poll()` nichtblockierende Filedeskriptoren verwenden, um zu vermeiden, dass ein anderer Systemaufruf auf einem einzelnen Filedeskriptor blockiert, während auf einem anderen schon ein neues Ereignis vorliegt. Dabei kann der Fehler `EAGAIN` auftreten, der gesondert behandelt werden muss. Um dies zu erreichen, kann man die Funktion `fcntl()` verwenden. Eine kleine Wrapper-Funktion finden Sie im auf der Praktikumswebsite zur Verfügung gestellten Code.

Aufgabe PROJ-6-7

[Module **`client`, `server`, `util`, `logger`**] Semaphoren

Da auf dem nächsten Blatt Server und Client in mehrere Prozesse aufgespalten werden, müssen die gemeinsam genutzten, kritischen Bereiche jetzt mit Semaphoren geschützt werden. Legen Sie dazu in den Setup-Bereichen von Server und Client je eine Semaphorengruppe mittels der Funktion `semget()` an. In beiden Programmen benötigt man je eine Semaphore für die Logger-Pipe. Im Server benötigt man zusätzlich eine Semaphore für das Shared Memory, welches die Dateiliste hält. Der Client benötigt zwei weitere Semaphoren, eine für die Consoler-Pipe, eine für das Shared Memory mit den Resultaten einer Suche.

Schreiben Sie außerdem im Modul `util` drei Funktionen:

a. `int semWait(int semid, int semnum);`
Benutzt die Bibliotheksfunktion `semop()` um die Semaphore `semnum` des Sets `semid` um 1 zu verringern. Gibt den Rückgabewert von `semop()` zurück.

b. `int semSignal(int semid, int semnum);`
Benutzt die Bibliotheksfunktion `semop()` um die Semaphore `semnum` des Sets `semid` um 1 zu erhöhen. Gibt den Rückgabewert von `semop()` zurück.

c. `int semVal(int semid, int semnum);`
Benutzt die Bibliotheksfunktion `semctl()` um den Wert Semaphore `semnum` des Sets `semid` zurückzugeben. Diese Funktion dient nur zum Debuggen.

Für diese Funktionen können Sie folgende Konstanten verwenden, um die Semaphoren der Sets anzusprechen:

```
/* Semaphore defines */
/* General */
#define SEM_LOGGER 0
/* Server */
#define SEM_FILELIST 1
/* Client */
#define SEM_CONSOLER 1
#define SEM_RESULTS 2
```

Setzen Sie nun diese Funktionen in den Funktionen `logmsg()` (Modul `logger`) und `consolemsg()` (Modul `consoler`) ein, um die Schreibseite der jeweiligen Pipes zu schützen. Halten Sie den geschützten Bereich möglichst klein.