3 Hausaufgaben

Auf diesem Aufgabenblatt wollen wir Ihre Shell um zwei Funktionen ergänzen, die sich mit Interprozesskommunikation und grob verwandten Konzepten beschäftigen.

3.1 Umleitung der Ein-/Ausgabe (7 Punkte)

Wir nähern uns der Interprozesskommunikation mittels Pipes in einem ersten Schritt an, indem wir uns mit der Umleitung der Standardein- und -ausgabe beschäftigen. In einer Shell werden hierzu die Symbole `'<'` und `'>'` verwendet. Diese sollen Sie nun in der Kommandozeile Ihrer Shell erkennen. Achten Sie darauf, dass jeder der beiden Parameter nur maximal einmal vorkommen darf, denn sonst wäre nicht klar, in welche Datei geschrieben bzw. welche Datei gelesen werden sollte.

Wichtig: Sie **können** hierzu Ihren Parser vom Blatt 06 erweitern. Sie **können aber auch** einen vereinfachten neuen Parser schreiben, der keine Sonderzeichen (`'"$\`) unterstützen muss. Dieser muss lediglich einen Eingabestring in durch ein oder mehrere Leerzeichen getrennte Parameter transformieren. Egal ob alt oder neu, verwenden Sie hierfür wieder die Datei parser.c.

- `< Dateiname` liest statt von der Standardeingabe aus der Datei mit dem Namen "Dateiname".
- `> Dateiname` schreibt statt auf die Standardausgabe in die Datei mit dem Namen "Dateiname".

Das Umleiten der Standardein- und/oder -ausgabe muss der von der Shell mit `fork()` erzeugte Kindprozess umsetzen, denn Sie wollen ja nicht die Ein-/Ausgabe der Shell ändern, sondern nur die des jeweiligen Kommandos. Nach dem Umsetzen können Sie dann das Kommando mit `execve()` ausführen.

Die Standardeingabe hat den Dateideskriptor 0, die Standardausgabe den Deskriptor 1 (und die Ausgabe für Fehlermeldungen, wie sie etwa von `perror()` ausgegeben werden hat den Deskriptor 2). Alle Dateideskriptoren werden in einer Tabelle verwaltet, und 0, 1, 2 usw. sind Indizes in dieser Tabelle. Wird eine neue Datei geöffnet oder eine Pipe erzeugt, so wird der niedrigste freie Index verwendet. Wenn Sie also erst die Standardeingabe schließen (`close(0)`) und danach eine Datei öffnen wird für diese neu geöffnete Datei der Deskriptor 0 (wieder-)verwendet. Auf diese Weise können Sie gezielt die Ein-/Ausgabe umlenken. Sie können eine Datei auch erst öffnen und anschließend mit dem Systemaufruf `dup()` diesen Dateideskriptor duplizieren, nachdem Sie Standardein- und/oder Standardausgabe geschlossen haben – auch dann wird der niedrigste freie Eintrag in der Deskriptortabelle verwendet.

Hinweise: Sie können Ihre Kommandozeileninterpretation in zwei Stufen realisieren. Die erste trennt den Eingabestring in mehrere Parameter auf, inklusive ">" und "<" (und dann "|" in der nächsten Aufgabe) und liefert alle diese Parameter in einer Liste zurück, wie bei der letzten Aufgabe auch. Die zweite Stufe arbeitet dann nur noch auf der Liste und sucht nach den Sonderzeichen für die Umleitung der Ein-/Ausgabe, entfernt diese und die zugehörigen Parameter aus der Liste (und stellt auch fest, wenn ein Parameter fehlt bzw. ein das gleiche Sonderzeichen zwei Mal auftritt) führt dann die entsprechende Sonderbehandlung durch.

Sie brauchen, wie oben schon erwähnt, das Symbol "\" hier nicht zu berücksichtigen und müssen sich insbesondere **keine** Gedanken machen, wie Sie in bei einer zweitstufigen Interpretation der Eingabestrings etwa die Zeichenfolgen "\|". "\<" pder "\>" korrekt behandeln.

Beispiel:

```
$ ls > files.txt
$ cat files.txt
Makefile
files.txt
list.c
list.h
mysh
mysh.c
parser.c
$ wc < files.txt
       7       7      54
```

Quellen: parser.c mysh.c list.c list.h

Executable: mysh

3.2 Pipes für die Kommandozeile Ihrer Shell (3 Punkte)

Sie haben in der Vorlesung den System-Call `pipe()` kennengelernt. In dieser Aufgabe sollen Sie die Funktion solcher Pipes in Ihre Shell einbauen. Die grundsätzliche Vorgehensweise ist dabei ganz ähnlich wie bei der Umleitung von Standardein- und -ausgabe, nur dass Sie jetzt nicht aus einer bzw. in eine statische Datei umleiten, sondern in eine Pipe unter Verwendung des entsprechenden Kommandos `pipe()`. Erinnern Sie sich daran, dass die Kommunikation in den Pipes in einer Shell strikt unidirektional ist.

Erweitern Sie hierzu Ihre Shell aus der vorigen Aufgabe. Für die Realisierung einer Pipe verwenden wir das Pipe-Symbol ("|"). Sie müssen nicht mehr als ein Pipe-Symbol auf der Kommandozeile unterstützen. Ihre Shell soll auch weiterhin die Umleitung von Ein- und Ausgabe unterstützen.

```
$ cat mysh.c parser.c | grep -n else
```

In obigem Beispiel werden zwei Kommandos erzeugt:

- `cat mysh.c parser.c` gibt die Inhalte der beiden Dateien `mysh.c` und `parser.c` nahtlos nacheinander auf der Standardausgabe aus.
- `grep else` durchsucht den von der Standardeingabe gelesenen Daten nach Zeilen, die das Suchwort "else" enthalten und gibt nur noch diese Zeile aus. Die Option "-n" sorgt dafür, dass die laufende Zeilennummer vorangestellt wird.

Die Shell muss also für jedes Auftreten einer Pipe in der Kommandozeile jeweils einen weiteren Prozess starten. Vor dem Erzeugen der Kindprozesse muss der Elternprozess jeweils dafür sorgen, dass deren Standardein- und -ausgabe entsprechend in Serie durch vom Elternprozess erzeugte Pipes miteinander verbunden sind.

Hinweise: In dieser Aufgabe müssen Sie beim Auftreten einer Pipe zwei Kindprozesse erzeugen, einer für den Teil links und einen für den Teil rechts von der Pipe. (Im allgemeinen Fall wären es $n+1$ Prozesse bei n Pipes.) Für jeden müssen Sie seinen Teil der Liste in ein Array umwandeln, wie Sie es auch auf dem letzten Aufgabenblatt bereits getan haben (dort allerdings nur für einen Prozess).

Das Erzeugen der Pipe muss im Shell-Prozess geschehen, bevor das erste Kommando erzeugt wird, denn die Deskriptoren dieser Pipe können nur mittels `fork()` an die beiden Kind-Prozesse weitergegeben werden. Der Shell-Prozess darf die Pipe erst schließen, wenn beide Kindprozesse erzeugt worden sind. Dann muss(!) er das aber auch tun!

Denken Sie daran, dass ein Prozess das Ende einer Eingabe erst dann erkennt, wenn alle möglichen Quellen für die Standardeingabe geschlossen worden sind. Sie müssen also dafür sorgen, dass Sie alle nicht benötigten Verweise auf den Deskriptor schlieẽen.

Achten Sie darauf, dass ein Prozess nur entweder aus einer Pipe lesen kann oder aus einer Datei mittels Eingabeumleitung. Ebenso kann ein Prozess seine Standardausgabe nur entweder über eine Pipe dem Folgeprozess zur Verfügung stellen oder die Ausgabe in eine Datei umleiten lassen.

Wenn ein Kindprozess sein Kommando nicht ausführen kann, sollte er die anderen Kindprozesse per Signal beenden (SIGKILL). Starten Sie Ihre Kindprozesse entsprechend der Kommandozeile von links nach rechts. Vermerken Sie die Prozess-IDs in einer Datenstruktur des Shell-Prozesses, die dann ja jeweils an alle Kindprozesse vererbt wird. Wenn Sie den n-ten Prozess starten und etwas schiefgeht, können Sie einfach die Datenstruktur bis $n-1$ durchgehen und die anderen Prozesse beenden.

Quellen: parser.c mysh.c list.c list.h

Executable: mysh

Ein Beispiel (natürlich hängt die Ausgabe in diesem Fall von Ihrem Code ab). Lassen Sie sich nicht durch die Zeilennummern irritieren, denn die Musterlösung ist allgemeiner gehalten und kann beispielsweise mehr als zwei Pipes pro Kommandozeile ausführen.

```
$ ./mysh
$ cat mysh.c parser.c | grep -n else
53:    } else
106:                else
114:        } else if (!strcmp (le->data, "<") || !strcmp (le->data, ">")) {
122:                    else {
128:                } else {
132:                    else {
140:            } else {
147:        } else {
180:            } else {
230:            } else {
480:        } else {
556:            else {
$ exit
```