Hochschule RheinMain
Studiengang Medieninformatik
Sommersemester 2018
Prof. Dr. Wolfgang Weitz

Betriebssysteme und Rechnerarchitektur

Übungsblatt 1

präsiebzehnter April 2018

In diesem Praktikum verwenden wir jetzt und für immer Linux, nur Linux und nichts als Linux. Außerdem verwenden wir ANSI-C zum Programmieren - daher schreiben wir uns heute zum Einstieg ein paar nützliche Hilfsfunktionen zum Aufwärmen. Alle Übungsaufgaben sind entweder lehrreich oder können als Teil des Semesterprojekts eingesetzt werden oder am besten beides. Das Semesterprojekt ist keine Gruppenarbeit, sondern von jedem eigenständig zu erarbeiten. Code von anderen zu übernehmen (abgewandelt oder nicht) gilt als schändlicher Täuschungsversuch, projektrelevanten Code zum Abschreiben bereitzustellen entsprechend als Beihilfe. Die Aufgaben eines Übungsblatts sind bis zum Erscheinen des nächsten zu bearbeiten, die Ergebnisse müssen jederzeit auf Nachfrage gezeigt und erläutert werden können.

Aufgabe 1:
Im Dialog zu bleiben ist wichtig, das gilt auch für Konfigurationsdialoge. Eine Dialogbeschreibung sei bei uns ein Array von Einträgen des Typs `DialogRec`, siehe zugehörige Header-Datei „`dialog.h`“ im read.MI. Das Arrayende wird durch einen Eintrag markiert, dessen „`command`“-Wert den Leerstring enthält.Hier ein Beispiel:

```
DialogRec dialog[] = {
 /* command,      param, state, nextstate, validator */
 { "BEGIN",       "",    0,     1,         validate_noparam },
 { "kuehlschrank","",    1,     1,         validate_onoff },
 { "fernseher",   "",    1,     1,         validate_onoff },
 { "toaster",     "",    1,     1 },
 { "END",         "",    1,     2,         validate_noparam },
 { "" }
};
```

Diese Dialogbeschreibung gibt an, dass im Zustand „0“ (Anfangszustand) nur der Befehl „`BEGIN`“ möglich ist, der keinen Default-Parameterwert hat und bei Erfolg in den Folgezustand 1 führt. Jedem Kommando kann ein Funktionszeiger auf eine Validierungsfunktion für deren Parameterwert übergeben werden. Wenn dieser Funtionszeiger `NULL` ist, wird nicht validiert (wie beim *Toaster* im Beispiel).

Eine Folge konkreter Dialogeingaben für diesen Dialog könnte also wie folgt aussehen (richtig / falsch)

```
BEGIN                  | kuehlschrank off     -- geht erst in State #1, wir sind noch bei State 0
toaster huepf          | BEGIN MAL            -- BEGIN nur ohne Param. (per validator_noparam)
fernseher on           | fernseher an         -- für fernseher nur „on“ und „off“ (validator_onoff)
fernseher off          | kuehlschrank         -- kuehlschrank braucht Parameter (validator_onoff)
kuehlschrank on        | hopfenspender on     -- in dialog undefiniertes Kommando
END
```

Bitte implementieren Sie im C-Modul „`dialog.c`“ die Funktion

```
DialogRec *findDialogRec(char *command, DialogRec dialogspec[])
```

die in einem Dialog-Array (wie dem oben) den Eintrag findet, dessen '`command`'-String ohne Berücksichtigung von Groß-/Kleinschreibung den Anfang (Präfix) des übergebenen '`command`'-Parameters bildet. Im Erfolgsfall soll ein Zeiger auf dieses `DialogRec` zurückgegeben werden, ansonsten `NULL`.

Für das obige Beispiel würde `findDialogRec(„toaster forever“, dialog)` also einen Zeiger auf den vierten Eintrag (den mit „toaster“) liefern, ebenso wie `findDialogRec(„toast forever“, dialog)` den `NULL`-Zeiger zurückgibt.

Aufgabe 2:
Nun erschaffen Sie bitte in „dialog.c“ eine weitere Funktion

```
ProlResult processLine(char line[LINEMAX], int state, DialogRec dialogspec[]);
```

die einen String line nimmt, feststellt, ob dieser (unter Außerachtlassung von Groß-/Kleinschreibung) mit einem in dialogspec beschriebenen Kommando *beginnt* und (falls das so ist) den auf das Kommando folgenden Reststring in die param-Komponente des zugehörigen DialogRec in dialogspec übernimmt, wobei Leerzeichen am Anfang und Return ('\r') bzw. Linefeed ('\n') Zeichen am Ende des Parameters wegzulassen sind. Es wird eine ProlResult-Struct (die Struct selbst - kein Zeiger darauf) als Ergebnis zurückgeliefert, die wie folgt aufgebaut ist:

```
typedef struct pr {
   char failed;           /* 0 für ok, >0 für Fehler */
   DialogRec *dialogrec;  /* Bezugs-DialogRec */
   char info[GRMSGMAX];   /* Erläuternder Info-String */
} ProlResult;
```

Wenn processLine() kein DialogRec in dialogspec finden kann, dessen Kommando den Anfang der line bildet, wird die Komponente failed auf wahr gesetzt, dialogrec auf NULL und info enthält eine aufrichtig bedauernde Nachricht an den Aufrufer.

Wenn processLine() zwar ein passendes DialogRec findet, der aktuelle (übergebene) Systemzustand state jedoch nicht mit dem im DialogRec geforderten state übereinstimmt, gibt es analog eine Fehler-Rückgabe, dieses Mal jedoch mit einem Verweis auf das im dialogspec gefundene passende DialogRec in der Rückgabe-Komponente dialogrec, denn das könnte den Aufrufer ja interessieren.

Nehmen wir nun an, es wurde ein zur line passendes DialogRec gefunden und wir sind im richtigen Systemzustand. Das ist ja schon einmal nicht schlecht. Dann füllen wir den param-Teil des DialogRec wie anfang beschrieben mit dem Parameter-Teil aus der übergebenen line.

Wenn Sie sich die Vereinbarung von DialogRec in dialog.h ansehen, fällt Ihnen eine zusätzliche Komponente is_valid in's Auge, mit welcher der Inhalt dieses DialogRec als gültig belegt markiert wird. Dass irgendwas in param-String des DialogRec steht, heißt ja noch nicht, dass das auch sinnvoll ist. Dies zu prüfen ist auch Aufgabe von processLine(). Wann kann also processLine() eine anhand der übergebenen line gefundenes DialogRec als gültig markieren? Hier kommt die Komponente validator in DialogRec in's Spiel, der man einen Funktionszeiger auf eine Validatorfunktion übergeben kann. Eine solche Validatorfunktion hat folgende Gestalt:

```
int validator(DialogRec *)
```

Sie liefert den Wahrheitswert „wahr“, wenn das als Parameter übergebene DialogRec als in Ordnung ist, ansonsten „falsch“ (also 0). In der Eingangsbeispiellistingdialogzeile zu „fernseher“ finden Sie beispielsweise als Validierungsfunktion angegeben validate_onoff. Diese Funktion soll überprüfen, ob als Parameter zum „fernseher“ entweder „on“ oder „off“ übergeben wurde, und könnte demnach wie folgt aussehen:

```
int validate_onoff(DialogRec *d) {
   return !strcmp(d->param, "on") || !strcmp(d->param, "off"); }
```

Zurück zu processLine(): Wenn processLine() also zur übergebenen line ein passendes DialogRec in der dialogspec gefunden hat und eine Validatorfunktion angegeben ist, bestimmt diese den Wert der is_valid-Komponente in der DialogRec. Wenn es keine Validatorfunktion gibt, wird is_valid auf „wahr“ gesetzt, wenn die Verarbeitung ansonsten ohne Fehler war.

Nur wenn ein DialogRec zu line gefunden und auf is_valid gesetzt werden konnte, ist auch in der Rückgabe-Struct von processLine() das Feld failed auf „falsch“ (0) zu setzen, sonst nicht.

Hinweis: Sie können (und sollten) Ihre Funktionen testen, indem Sie in einem separaten C-Modul eine main()-Funktion anlegen, die einen Beispieldialog bereitstellt (z.B. zunächst einmal den von oben) und Ihre Funktionen auf diesen Dialog loslässt (im Sinne eines „do it yourself“-Unit-Testing). Decken Sie bitte auch Fehlerfälle ab und halten Sie stets ein passendes Makefile bereit, um sich Compilierung, Aufräumen etc. zu erleichtern.