AUSH - Laborbericht

Autor/en: Aurelio Buonomo

Klasse: 3IA

Schule: Technologische Fachoberschule "Max Valier" Bozen

Auftragsdatum: 12.04.2021

Abgabedatum: 13.04.2021

Auftraggeber: Karl Lunger; Marco Grazioli

Art des Dokuments: Laborbericht

Pfad:

https://snets-my.sharepoint.com/:w:/g/personal/stu_bnmrla04b16_snets_it/EU8UaOBLcH1Mho0cm3CvJt0BRkeU4_t8uc5rdtt-Q3cN2A?e=bi0lSZ

1. Inhalt

2. Arbeitsauftrag

Entwickle eine kleine Shell, die Programmaufrufe des Benutzers einlesen und durchführen kann. Nach der Durchführung kann der Benutzer weiteren Aufrufe durchführen. Zu berücksichtigen sind übergebene Argumente und evtl. die Angabe eines Pfades für den Aufruf.

3. Theoretischer Hintergrund

I. AUSH

AUSH ist eine Abkürzung für ***Au***relio ***Sh***ell.

II. Shell

Nach Wikipedia:
In der Informatik wird als Shell die Software bezeichnet, mittels derer ein Benutzer mit einem Betriebssystem interagiert – eine Mensch-Maschine-Schnittstelle.

Unter Linux ist die BASH-Shell am bekanntesten und ist die Default-Shell der meisten Distributionen.
Auch Windows hat eine Shell: cmd.exe existiert unter Windows schon lange, doch mittlerweile verwendet man die PowerShell, die benutzerfreundlicher ist und besser für Entwickler geeignet ist.

III. Exec*

Die C-Funktionen der exec*() Familie können verwendet werden, um externe Programme auszuführen. Meistens verwendet man execvp. Diese Funktion übernimmt die Argumente, die bei der Ausführung des Programmes mitgegeben werden sollen als Array von Char Zeigern (exec***v***p), außerdem sucht die Funktion, wenn kein Pfad als Befehl / Argument 0 mitgegeben wird, das auszuführende Programm in den Suchpfaden (PATH Umgebungsvariable) (execv***p***).

IV. Fork

Zu beachten ist, dass die exec* Funktionen nach der Ausführung nicht zu dem aufrufenden Prozess zurückkehren, deshalb sollte man exec* in einem Kindprozess verwenden.
Um einen solchen Kindprozess zu erstellen kann man den Systemaufruf fork() benutzen. Diese Funktion erstellt einen Kindprozess, der mit dem exakten Code weiterläuft, wie der aufrufende Prozess. Die Idee dahinter ist, mit fork() einen Kindprozess zu erstellen, anhand des Rückgabewertes den Code für Elternprozess und Kindprozess zu trennen, im Kindprozess das externe Programm mit execvp() aufzurufen und im Elternprozess auf das Kind zu warten.

V. String.h

Sehr wichtig in diesem Projekt ist auch die <string.h> Headerdatei. Diese enthält viele Funktionen, die einen effizienteren Umgang mit Zeichenketten ermöglichen.

VI. Libreadline

In diesem Projekt habe ich außerdem Libreadline verwendet, eine GNU C-Library, die eine bessere Eingabeaufforderung ermöglicht. Sie stellt folgende Funktionen zur Verfügung:
readline(char*): Mit dieser Funktion kann man eine Eingabe vom Benutzer verlangen. Übergeben kann man den Shell-Prompt als Zeichenkette
add_history(char*): Mit dieser Funktion kann man den übergebenen String dem Verlauf der Shell hinzufügen, der mit „Pfeiltaste nach oben“ aufgerufen werden kann.

4. Analyse

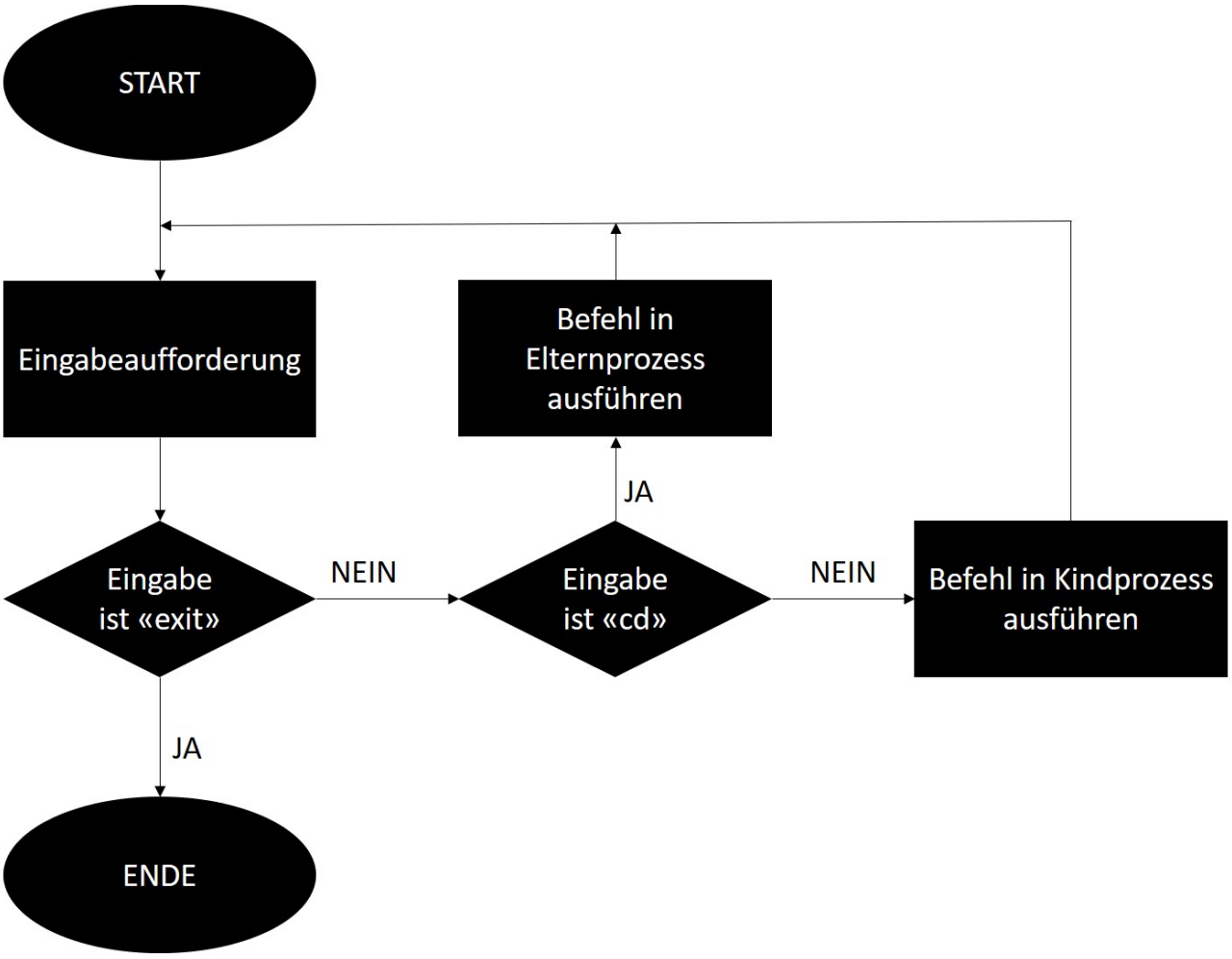
Zu entwickeln ist ein Programm (Shell), das beim Start eine Eingabe vom Benutzer fordert, die Eingabe in die Argumente aufteilt und das Programm ausführt. Die Eingabelänge wird kontrolliert und dem Benutzer wird dann eine Meldung gegeben, wenn die Länge die maximale Länge überschreitet

5. Planung

I. Beschreibung

Das Programm startet. Es wird dann Speicher für die Benutzereingabe angelegt und dann über readline() vom Benutzer verlangt. Der Prompt der Shell wird bei jedem Zyklus neu erstellt. Anschließend wird die Eingabe in Tokens unterteilt, die am Ende die Argumente sind. Außerdem wird eine Variable gesetzt, die später aussagt welche Art von Befehl ausgeführt werden soll. Ein Pipe-Struct wird auch erstellt, wenn eine Pipe entdeckt wird, doch in der aktuellen Version bricht die Shell einfach ab, da Pipes noch in Entwicklung sind. Darauf wird die Flag kontrolliert. Handelt sich es um eine Pipe wird das Programm diese auswerten und ausführen, wird der „exit“ Befehl eingegeben beendet sich die Shell, wird der „ret?“ Befehl eingegeben sollte die Shell den letzten Rückgabewert ausgeben, doch diese Funktionalität ist noch in Entwicklung, wird der „cd“ Befehl eingegeben so wird dieser in dem Elternprozess ausgeführt (kein fork()), wird ein sonstiger Befehl eingegeben so wird dieser mit execvp() in einem Kindprozess ausgeführt. Im Default-Fall wird eine Fehlermeldung ausgegeben, doch der kann unter normalen Umständen nicht auftreten. Dieser Vorgang wird dann wiederholt.

II. Flussdiagramm



6. Implementierung

Siehe Quellcode:

- aush.c:
  Hauptprogramm; Ruft Funktionen in util.c auf, wie Eigabeaufforderung usw.
- util.c:
  Enthält alle Funktionen, die die Shell braucht, wie die Aufteilung der Eingabe in Tokens, die Eingabeaufforderung selbst und mehrere Hilfsfunktionen, um die Arbeit mit Strings zu vereinfachen.

7. Testbericht

I. Testfälle

- Testfall 1: Die Shell wird gestartet
- Testfall 2: Der Benutzer gibt nichts ein und drückt „Enter“
- Testfall 3: Der Benutzer gibt einen ungültigen Befehl ein
- Testfall 4: Der Benutzer gibt einen gültigen Befehl ein
- Testfall 5: Der Benutzer verwendet die Funktionalitäten von Libreadline

II. Verhalten der Shell

Die Shell verhält sich in allen Testfällen wie erwartet. Wird keine Eingabe oder eine ungültige Eingabe vom Benutzer übergeben so ignoriert die Shell diese einfach. Wird ein gültiger Befehl eingegeben, so führt die Shell diesen erfolgreich aus. Die internen Befehle (siehe Handbuch) funktionieren wie sie sollen. Die Funktionen von Libreadline funktionieren ebenfalls mit einer plattformbedingten Ausnahme (siehe Beobachtung und Erkenntnisse). Gibt der Benutzer einen zu langen Befehl ein so wird ihm das ebenfalls mitgeteilt.

8. Beobachtung und Erkenntnisse

I. Schwierigkeiten

Das größte Problem bei der Entwicklung der Shell war die Arbeit mit den Strings. Ich hatte oft Speicherprobleme, die ich mit der Zeit lösen konnte. Eine große Fehlerquelle war das Vergessen der Nullterminierung von handangelegten Strings und vom Args-Array. Sind diese nicht nullterminiert, kann im Array Müll landen.

II. Ungelöster Bug

Ein Bug konnte ich nicht entfernen, denn er stammt wahrscheinlich von Libreadline, dessen Quellcode ich nicht verändern kann. Der Bug führt, bei Inputs, die die Terminalbreite überschreiten, zur Überschreibung der Zeile. Dieser Bug tritt aber nur bei Windows-Terminals (PowerShell, Windows Terminal, …) auf. Ich gehe davon aus, dass der Bug plattformbedingt ist, da unter Linux die Shell einwandfrei funktioniert.

9. Quellverzeichnis / Resourcen

https://de.wikipedia.org/wiki/Shell_(Betriebssystem) (12.4.2021)
https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29 (12.4.2021)
https://en.wikipedia.org/wiki/Cmd.exe (12.4.2021)
https://en.wikipedia.org/wiki/PowerShell (12.4.2021)
https://www.geeksforgeeks.org/exec-family-of-functions-in-c/ (12.4.2021)
https://www.geeksforgeeks.org/fork-system-call/ (12.4.2021)
https://www.tutorialspoint.com/c_standard_library/string_h.htm (12.4.2021)
https://tiswww.case.edu/php/chet/readline/rltop.html (12.4.2021)