Hinweise und Richtlinien für die Rechnerübung zu Betriebssysteme

- Die Aufgaben sollen in Gruppen aus zwei Teilnehmern gelöst werden. Bitte sprechen Sie sich hierfür mit ihren Kommilitonen ab und melden sie sich gemeinsam auf der Internetseite zur Betriebssysteme Vorlesung an:

  `https://www.ibr.cs.tu-bs.de/courses/ws1213/bs/index.html`

- Die **Abgabe der Aufgaben erfolgt durch Präsentation** der eigenen Lösung gegenüber einem Übungsleiter in einer der kleinen Übungen bis zum angegebenen Abgabetermin. Die Programme sollten hierfür auf den Rechnern im CIP Pool Laufen.
- Vorgegebene Dateien werden über Betriebssysteme-Internetseite zugänglich gemacht (s. Oben).
- Hilfestellungen und Vorgaben zu den Übungsaufgaben werden in der Tafelübung gegeben und finden sich außerdem in den Aufgabenbeschreibungen und in den gegebenen Quelltext Dateien.
- Bei der Bearbeitung der Aufgaben kann (und wird) es immer mal wieder zu Unklarheiten bzw. Fragen kommen. Daher:
  - Kommt zu den Tafelübungen!
  - Kommt zu den Rechnerübungen!
  - Fragt die Übungsleiter!
  - Kontaktiert uns per E-Mail!
  - Fragt Kommilitonen!
- Bitte führt keine Änderungen an den gegebenen Quellcode durch. Alle Aufgaben sind so konzipiert, dass entweder nur in markierten Bereichen Ergänzungen vorgenommen werden sollen oder eigene Dateien für den Programmcode angelegt werden müssen.
- Der Code muss lesbar und nachvollziehbar sein. Falls komplizierte Teile vorkommen, sollen diese mit Kommentaren erläutert werden.
- Die Lösungen sollen eigenständig erarbeitet werden. Wer abschreibt lernt nichts.

Bei Problemen mit der Aufgabenstellung wendet euch an die BS Mailing List bs@ibr.cs.tu-bs.de oder direkt an martens@ibr.cs.tu-bs.de.

Aufgabenblatt #R1 zur Rechnerübung Betriebssysteme

Ziel der folgenden Aufgaben ist es den grundlegenden Umgang mit **make**, **C** und der **C-Standardbibliothek** zu üben und zu vertiefen.

1.1 Makefile

Gegeben ist die Datei hello.c. Schreiben Sie eine Makefile, wie in der Tafelübung beschrieben, dass die Targets *all*, *clean* und *hello* bereitstellt.

- *all*: soll das Target hello erzeugen
- *clean*: soll mit Hilfe von **rm(1)** alle `.o` und alle `.exe` Dateien aus dem Arbeitsverzeichnis entfernen
- *hello*: soll den Quellcode in hello.c zum Programm hello.exe kompilieren, verwenden Sie dafür den Compiler **gcc(1)** mit den Parametern: **-std=c99 –pedantic –D_XOPEN_SOURCE=600 -Wall**

1.2 shellutils

Als nächstes sollen Hilfsfunktionen für die Entwicklung einer eigenen Shell implementiert und getestet werden. Gegeben sind hierfür die header Datei shellutils.h mit den kommentierten Funktionsdeklarationen und die Datei shellutils.c mit unvollständigen Funktionsimplementierungen. Folgende Funktionen sollen nun von Ihnen in der shellutils.c implementiert und in einem eigenen kleinen Programm getestet werden:

1.2.1 prompt()

Implementieren Sie die Funktion `prompt()` in der shellutils.c. Diese Funktion soll das aktuelle Arbeitsverzeichnis gefolgt von einem Doppelpunkt (:) ausgeben. Verwenden Sie hierzu die Bibliotheksfunktionen **getcwd(3)** sowie **printf(3)**.
Schreiben Sie außerdem ein Programm `prompt.c`, dass die Funktion `prompt()` testet in dem es die funktion prompt() aus den shellutils aufruft.
Erweitern Sie ihre Makefile um das Target `prompt`, welches aus prompt.c und den shellutils das programm prompt.exe erzeugt. Ergänzen Sie außerdem das *all* Target so, dass beim Aufruf von *make* die Targets *hello* **und** *prompt* ergzeugt werden.

1.2.2 parseCommandLine()

Damit die shell aus einer Kommando Eingabe in Form eines Strings den eigentlichen Programmaufruf von Parameter, Flags und sonstigen Angaben unterscheiden kann benötigt man einen sogenannten Parser.
Gegeben ist hierfür die Funktion `COMMAND *parseCommandLine(char *cmdLine)`. Diese Funktion bekommt das Argument einen char-Pointer auf einen String namens cmdLine übergeben. Abhängig von der Länge des Arguments cmdLine alloziert die Funktion zur Laufzeit den benötigten Speicher für die Struktur `COMMAND` (s. shellutils.h). Anschließend wird die Struktur mit Inhalt gefüllt und ein Pointer auf die Strukur (`cmd`) von der Funktion zurückgegeben.

Leider ist die Funktion parseCommandLine() bisher nicht in der Lage das Argument cmdLine in seine Bestandteile aufzuteilen. Das zu Implementieren ist nun Ihre Aufgabe:

- Erweitern Sie die Funktion parseCommandLine() im markierten Bereich so, dass der String auf den cmdLine zeigt in seine Einzelteile zerlegt wird. D.h. ein Array von char-Pointern (s. char** argvPos) soll im ersten Element auf den Funktionsaufruf und in allen folgenden Elementen auf die einzelnen Parameter zeigen. Verwenden Sie hierzu die Funktion **strtok(3)**. Gehen Sie davon aus, dass die Parameter durch Leerzeichen oder Tabulatoren (\t) voneinander getrennt sind.
  Da parseCommandLine() im Vorfeld genug Speicher für ein Array von 128 char*-Elementen reserviert, es aber unwahrscheinlich ist, dass es so viele Parameter gibt, soll das Array-Element nach dem letzten Parameter auf `NULL` zeigen um das Ende der Parameter zu markieren.

  Beispiel: das Kommando `ls -l -a` soll in folgendes überführt werden:

  ```
  argvPos[0] == ls
  argvPos[1] == -l
  argvPos[2] == -a
  argvPos[3] == (NULL)
  ```

- Schreiben Sie außerdem ein Programm parse.c um die Funktion parseCommandLine() auf Korrektheit zu überprüfen. Das Programm parse.c soll um eine Eingabeaufforderung bitten (verwenden Sie hierzu **fgets(3)**), parseCommandLine() aufrufen und die zurückgegebene Kommando Datenstruktur leserlich ausgeben.

  Beispiel: Ihre Ausgabe sollte für die Eingabe `ls -l -a` etwa wie folgt aussehn:

  ```
  background: 0
  outFile: (null)
  parseError: (null)
  cmdLine: ls -l -a
   argv[0]: ls
   argv[1]: -l
   argv[2]: -a
  ```

- Erweitere Sie ihre Makefile um das Target **parse**, welches aus parse.c und den shellutils das Programm parse.exe erzeugt. Erweitern Sie auch das *all* Target so, dass beim Aufruf von *make* die Targets *hello*, *prompt* und *parse* erzeugt werden.

Abgabe: bis 16.11.2012