Praktikum 2: Software-Entwicklung in Linux

Die Lernziele in diesem Praktikum sind

- Entwicklung von Software unter Linux.
- Der Debugger `gdb`.

Hinweise:
Lesen Sie auch die zum Praktikum gehörenden Praktikumsfolien für zusätzliche Hinweise.

Aufgabe 1 (Der Editor `vi`)
Implementieren Sie ein *Hello World*-Programm in C innherhalb eines Terminals. Das ist ein häufiges Szenario wenn Sie auf einem entfernten Server oder eingebetteten Gerät arbeiten. Melden Sie sich dazu mittels `ssh` auf dem SWT-Server an, oder verwenden Sie ein Terminal an Ihrem lokalen Rechner. Benutzen Sie den Editor `vi` und kompilieren Sie das Programm mittels des Kompilers `gcc`. Beide Tools sind in jedem Linux-System verfügbar.

Hinweise:
Sie können für den Rest der Vorlesung einen moderneren Editor verwenden, z.B. *Geany* (`geany`) oder *Visual Studio Code* (`code`).

Aufgabe 2 (Der Debugger `gdb`)
Im Export-Verzeichnis finden Sie einige fehlerhafte Implementierungen von Algorithmen, die Sie bereits aus der OOP-Vorlesung kennen.

- Eine binäre Suche in C++.
- Eine doppelt verkettete Liste in C.

Nutzen Sie den `gdb`, um die Fehler zu lokalisieren.

Aufgabe 3 (Mini-Shell: Kommandoausführung)
Implementieren Sie schrittweise eine eigene Befehlsshell namens `myshell` in C++. Erstellen Sie dazu eine neue Datei `myshell.cpp` als Hauptprogramm.

Ihre Shell soll als Kommandoprompt `"myshell> "` ausgeben. Daraufhin soll eine ganze Zeile mit einem Kommando von der Standardeingabe als String eingelesen werden.

Als Vorbereitung auf folgende Aufgaben soll die Shell, bevor sie mit der Ausführung von Befehlen beginnt, den eingelesenen String in den Befehlsnamen und die Argumente aufsplitten. Bei Eingabe von `exit` gibt die Shell `exit` aus und beendet sich daraufhin.

Führen Sie ansonsten das eingegebene Kommando aus. Die Shell blockiert bis zum Ende der Ausführung und kehrt dann zur Eingabeaufforderung zurück. Verwenden Sie für diese Aufgabe die Systemaufrufe `fork`, `execvp` und `waitpid` (alternativ `wait`, das auf einen beliebigen Kindprozess wartet.)

Testen Sie mit Befehlen wie `ls -l` und `sleep 5` und vergleichen Sie mit der Standard-Shell.

Hinweise:

- Das letzte Element im `execvp`-Parameter `char* argv[]` muss ein Null-Pointer sein.
- Parsen des Strings ist durch Verwendung von `getline` zum einlesen und `stringstream` mit Ein- Ausgabe-Operatoren zum Auslesen der Wörter möglich. Beispielprogramm:

```
#include<string>
#include<sstream>
#include<iostream>

using namespace std;

int main() {
  string str = "10 20";
  stringstream sstr(str);
  int a = 0;
  int b = 0;
  sstr >> a >> b;
  cout << a << " " << b; // 10 20
}
```

Aufgabe 4 (Mini-Shell: Hintergundausführung)
Erweitern Sie die Shell aus der letzten Aufgabe um die Möglichkeit Prozesse im Hintergrund auszuführen. Dabei soll ein Befehl der Form `CMD &` die Ausführung des Befehls `CMD` im Hintergrund starten. Die Shell soll sofort zur Eingabeaufforderung zurückkehren.

Wenn die Ausführung des Befehls abgeschlossen ist und der Benutzer die Eingabetaste drückt, sollte die Shell vor der nächsten Ausgabe des Prompts melden, dass die Ausführung des Befehls beendet ist. Ein Aufruf von `waitpid` mit der Option `WNOHANG` ermöglicht ihnen zu prüfen, ob ein Sohnprozess fertig ist, ohne zu blockieren. Die Shell muss sich natürlich merken, welche Prozesse im Hintergrund gestartet wurden. Wählen Sie dazu eine geeignete Datenstruktur.

```
myshell> sleep 5 &
Gestartet: sleep 5 &
myshell>
Beendet: sleep 5 &
```