Prof. Dr. Franz Schweiggert Institut für Angewandte Informationsverarbeitung 20. Mai 2009
Michael Mattes Blatt 5

Systemnahe Software (II) im Sommersemester 2009

Abgabetermin: 27. Mai 2009

16 Sicherheit und das SUID-Bit (7 Punkte)

Schreiben Sie ein Programm `pubcat.c`, welches als Argument einen Dateinamen erhält, diesen als relativen Pfad zum Unterverzeichnis `pub/` Ihres Heimatverzeichnisses interpretiert und die entsprechende Datei ausgibt. *Hinweis:* Sie dürfen Ihr Heimatverzeichnis direkt in den Code schreiben und müssen es nicht aus dem Environment beziehen. Das nennt man umgangssprachlich auch "fest verdrahten".

Wenn Sie das Programm selbst ausführen, sollten also folgende Befehle das selbe Ergebnis liefern:

```
$ cat ${HOME}/pub/unter/datei.txt
$ pubcat unter/datei.txt
```

Angenommen, die Datei wäre von jedem ausführbar und besäße das SetUID-Bit (zu setzen mit `chmod +s pubcat` bzw. `chmod 04xxx pubcat`). Welches Sicherheitsproblem stellt dies (mindestens!) dar? Nennen Sie zwei Möglichkeiten, dieses Problem zu beheben.

tinysh erweitern

In den folgenden Aufgaben werden wir uns mit einer einfachen Shell-Implementierung, der `tinysh`, befassen. In jeder Aufgabe geht es darum, einen bestimmten Bereich der Shell zu verbessern bzw. zu erweitern.

17 Builtin-Schnittstelle (7 Punkte)

Schaffen Sie in der tinysh eine Schnittstelle, die es ermöglicht, einfache Shell-Builtins einzubauen. Diese sollen bevorzugt vor eventuell gleichnamigen Befehlen ausgeführt werden.

Der guten Erweiterbarkeit halber soll jedes Builtin der Shell über eine einheitliche Struktur bekanntgegeben werden. Diese Struktur enthält den Kommandonamen des Builtins sowie einen Zeiger auf die auszuführende Funktion und hat folgende Form:

```
struct builtin {
        char *cmdname;
        int (*function)(strlist* tokens);
}
```

Die tinysh muss überprüfen, ob das erste Token der Zeile mit dem `cmdname` eines Builtins übereinstimmt und dieses dann ggf. ausführen.

Als Beispiel für die Verwendung dieser Struktur können Sie das Builtin `exit` betrachten:

```
int exit_builtin(strlist* tokens) {
        exit(0);
        return 0; // wird nicht erreicht, ist aber syntaktisch korrekt
}

struct builtin exitbuiltin;
exitbuiltin.cmdname = "exit";
exitbuiltin.function = &exit_builtin;

// Alternativ geht es auch so direkt:
struct builtin exitbuiltin = {
        "exit",
        &exit_builtin
};
```

Noch ein Tipp: Falls Sie diese Aufgabe nicht schaffen, können Sie die nachfolgenden Aufgaben trotzdem lösen, indem Sie sich an obigem Beispiel orientieren.

18 Builtin echo (3 Punkte)

Schreiben Sie ein Builtin `echo`, welches die angegebenen Argumente wieder auf der Standardausgabe ausgibt. *Vorsicht:* Bei dieser Aufgabe müssen Sie sicherstellen, dass auch wirklich Ihr Builtin ausgeführt wird und nicht `/bin/echo` - testen Sie dies ggf. durch das Einfügen einer Zusatzausgabe in Ihrer Implementation. Das gleiche gilt für die folgenden beiden Aufgaben.

19 Builtin env (3 Punkte)

Schreiben Sie ein Builtin `env`, welches alle Umgebungsvariablen auf der Standardausgabe ausgibt.

20 Builtin setenv (3 Punkte)

Schreiben Sie ein Builtin `setenv`, welches eine Umgebungsvariable zum Environment hinzufügt. Sie können auf die Prüfung doppelt vorhandener Umgebungsvariablen verzichten. Ein Beispiel hierzu wäre:

```
%% env
A=1
%% setenv B=2
%% env
A=1
B=2
%% setenv A=3
%% env
A=1
B=2
A=3
%%
```

21 Parsererweiterung (7 Punkte)

Erweitern Sie den Parser der tinysh so, dass nicht nur Leerzeichen, sondern auch Tabulatorzeichen (Tabs) als Trennzeichen zwischen den Tokens stehen können.

Viel Erfolg!