

# OPERATING SYSTEMS BEISPIEL 2

## Aufgabenstellung – dsort

### SYNOPSIS:

```
dsort "command1" "command2"
```

Schreiben Sie ein Programm, das die beiden Kommandos `command1` und `command2` ausführt, deren Ausgaben einliest und in ein gemeinsames Array speichert. Dieses Array wird dann sortiert und an das Unix-Kommando `uniq -d` weitergegeben. Die Ausgabe Ihres Programmes soll also identisch sein mit jener des folgenden Shellskripts<sup>1</sup>:

```
#!/bin/bash

( $1; $2 ) | sort | uniq -d
```

## Anleitung

Das Programm soll für die beiden Kommandos jeweils mittels Pipes die Ausgabe der Kindprozesse zeilenweise einlesen und in ein Array speichern. Die Kindprozesse werden mit *fork(2)* erzeugt und sollen die Bourne-Shell mit dem Parameter `-c` ausführen (also `/bin/bash -c`). Dadurch ist es möglich dem Programm mittels Anführungszeichen ein Shellkommando zu übergeben – die Shell kümmert sich um alles Weitere.

Das vom Programm erzeugte Array mit den Ausgaben der Kommandos soll dann sortiert werden. Danach wird ein neuer Kindprozess gestartet, welcher das Programm `uniq` mit der Option `-d` ausführen soll. Das sortierte Array wird diesem Prozess über die Standardeingabe übergeben (wieder *pipe(2)* verwenden), indem zeilenweise in die Pipe geschrieben wird.

Der `uniq`-Prozess sucht im `-d`-Modus alle mehrmals nacheinander vorkommenden Zeilen heraus und gibt sie auf die Standardausgabe aus. `uniq` ist ein UNIX-Standardprogramm und muss von Ihnen nicht programmiert werden.

Beispiel:

```
$ ./dsort "cat /etc/passwd" "cat /etc/passwd.tmp"
```

Dieser Befehl würde die beiden Dateien `/etc/passwd` und `/etc/passwd.tmp` vergleichen und alle Zeilen, die in beiden Dateien (oder in einer Datei mehrmals) vorhanden sind, auf die Standardausgabe ausgeben.

Beispiel:

```
$ ./dsort "seq 0 3 100" "seq 0 11 100"
```

Dieser Befehl gibt alle gemeinsamen Teiler von 3 und 11, die zwischen 0 und 100 liegen, aus.

---

<sup>1</sup>Anmerkung: `$1` und `$2` stehen für `command1` und `command2`

## Hinweise

Beachten Sie insbesondere folgende Punkte:

- Sie müssen sowohl das Array als auch die Strings des Arrays dynamisch allozieren und auf eine saubere Freigabe achten. Hierfür stehen die Funktionen *malloc(3)*, *realloc(3)*, *free(3)* und bspw. auch *strdup(3)* zur Verfügung.
- Zum Sortieren der Zeilen können Sie *qsort(3)* verwenden.
- Ihr Programm sollte zeilenweise einlesen (z.B. mit *fgets(3)*) bzw. sortieren.
- Sie können davon ausgehen, dass eine Eingabezeile nicht länger als 1023 echte Zeichen ist.
- Die Funktion *popen(3)* darf nicht verwendet werden.
- Achten Sie darauf, dass der Elternprozess sich erst beendet, nachdem alle Kindprozesse terminiert haben.
- Gewisse Nicht-ANSI-Funktionen (wie etwa *strdup(3)*) werden erst sichtbar, wenn mit `-D_XOPEN_SOURCE=500` kompiliert wird.
- Achten Sie unbedingt darauf, dass alle Ressourcen auch wieder freigegeben werden (besonders im Fehlerfall)!

## Richtlinien

Bitte beachten Sie auch die *Richtlinien für die Erstellung von C-Programmen* sowie die *Allgemeinen Hinweise zur Beispielgruppe 2* auf der Übungswebsite.

Insbesondere ist es ab dieser Beispielgruppe notwendig, die Dokumentation in Doxygen zu führen. Es muss zumindest das HTML-Output generierbar sein. Bitte dokumentieren Sie ausnahmslos alle Funktionen (auch `static`-Funktionen; siehe `EXTRACT_STATIC` in der Doxyfile). Eine kurze Einführung haben wir Ihnen auf [http://wiki.vmars.tuwien.ac.at/index.php/Doxygen\\_Primer](http://wiki.vmars.tuwien.ac.at/index.php/Doxygen_Primer) bereitgestellt. Achten Sie weiters darauf, dass nach außen hin sichtbare Funktionen (exportierte Funktionen) in der Header-Datei und lokale (`static`) Funktionen nur in der C-Datei dokumentiert werden. Sie sollten auch Ihre Typen (insbesondere `structs`), Konstanten und globale Variablen dokumentieren.