Aufgabe 7:printdir (14 Punkte, Abgabe bis Do., 10.07.08 16:00)

Diese Aufgabe ist die letzte Übungsaufgabe und kann in 2er-Gruppen bearbeitet werden.

Entwickeln Sie ein Programm **printdir**, mit dem Sie — ähnlich wie mit dem UNIX-Kommando **ls(1)** — den Inhalt von Verzeichnissen (Katalogen / Directories) anzeigen können. Die Namen der Verzeichnisse werden auf der Kommandozeile übergeben. Wird kein Verzeichnis übergeben, so soll Ihr Programm das aktuelle Verzeichnis (**"."**) ausgeben. Gehen Sie beim Entwurf des Programms in folgenden Arbeitschritten vor:

a) Schreiben Sie nun eine Funktion

```
void printDir(const char *dirname),
```

welche zunächst nur den Verzeichnisnamen gefolgt von einem Doppelpunkt in einer eigenen Zeile ausgibt. Der Name des Verzeichnisses wird hierbei als einziger Parameter übergeben. Diese Funktion soll Ihr Programm für jedes anzuzeigende Verzeichnis aufrufen.

b) Erweitern Sie `printDir()` nun so, dass durch einen Tabulator eingerückt die Dateigrößen und Namen (durch einen Tabulator getrennt) aller Verzeichniseinträge in jeweils einer Zeile auf die Standardausgabe ausgegeben werden. Die Dateigröße können Sie mit der Funktion **stat(2)** ermitteln, hierzu müssen Sie vorher den kompletten Pfadnamen zum Verzeichniseintrag konstruieren. Wie beim UNIX-ls sollen dabei Dateien, deren Name mit einem **'.'** beginnt, nicht angezeigt werden, da es sich hierbei um versteckte Dateien handelt. Sollte ein Fehler beim Anzeigen eines Verzeichnisses oder Verzeichniseintrags auftreten, soll `printDir()` eine entsprechende Fehlermeldung liefern und mit der Ausgabe der übrigen Verzeichnisse bzw. Verzeichniseinträge fortfahren.

c) Erweitern Sie Ihr Programm nun so, dass die einzelnen Verzeichniseinträge nach Dateigröße sortiert ausgegeben werden. Verwenden Sie hierzu die Funktion **qsort(3)**. Sie können in dieser Teilaufgabe zunächst davon ausgehen, dass ein Verzeichnis nicht mehr als 1000 Einträge enthält. Bauen Sie sich als zu sortierende Datenstruktur ein Array aus Zeigern auf. Jeder dieser Zeiger zeigt auf eine Struktur, die die Größe und den Namen eines Verzeichniseintrags enthält. Vergessen Sie nicht, dass die von **readdir(3)** zurückgelieferten Zeiger bei erneutem Aufruf von **readdir(3)** ihre Gültigkeit verlieren. Daher ist es notwendig, jeden Datei-/Verzeichnisnamen nach dem **readdir(3)**-Aufruf zu kopieren. Den hierfür notwendigen Speicher sowie den Speicher für die Strukturen müssen Sie mit **malloc(3)** allokieren. Die Größe des Strings können Sie mit **strlen(3)** ermitteln, denken Sie aber an das zusätzlich notwendige Byte für das abschließende **'\0'**-Zeichen. Vergessen Sie nicht, allen von ihrem Programm allokierten Speicher auch wieder freizugeben.
Testen Sie Ihr Programm nun aber auch mit Verzeichnissen, die mehr als 1000 Einträge enthalten, wie z.B. dem Verzeichnis **/usr/bin**. Ihr Programm sollte sich im Idealfall mit einer Fehlermeldung beenden, keinesfalls aber abstürzen.

d) Ergänzen Sie ihr Programm nun so, dass es Verzeichnisse beliebiger Länge ausgeben kann. Hierzu müssen Sie das Array mit den **char*** nach Bedarf vergrössern. Verwenden Sie hierzu die Funktion **realloc(3).**

Hinweise: Sie finden im Verzeichnis **/proj/i4gdi/pub/aufgabe7** für jede Teilaufgabe eine Beispiellösung, deren Ausgabe Sie mit Ihrer eigenen Lösung vergleichen können. Für die Abgabe ist nur die endgültige Lösung notwendig, die Sie in einer Datei **printdir.c** in Ihrem Projektverzeichnis ablegen sollen. Achten Sie auf aussagekräftige Fehlermeldungen, die alle auf dem Standardfehlerkanal ausgegeben werden sollen.

Essentielle Funktionen: **opendir(3)**, **readdir(3), closedir(3), malloc(3), free(3), realloc(3), strlen(3), qsort(3), stat(2), strncpy(3), strncat(3)**