SPiC-Aufgabe #7: printdir

(12 Punkte, keine Gruppen)

Entwickeln Sie ein Programm `printdir`, das - ähnlich wie das UNIX-Kommando `ls(1)` - den Inhalt verschiedener Verzeichnisse ausgeben kann.

Es wird empfohlen beim Entwurf des Programms in folgenden Arbeitsschritten vorzugehen:

(a) Schreiben Sie zunächst ein Programm, welches alle Einträge des aktuellen Verzeichnisses (`'.'`) als je einen Eintrag pro Zeile ausgibt. Einträge, deren Name mit einem `'.'` beginnen, sollen nicht angezeigt werden (versteckte Dateien). (`opendir(3)`, `readdir(3)`, `closedir(3)`)

(b) Erweitern Sie das Programm nun so, dass vor dem Dateinamen auch die Dateigröße ausgegeben wird. Name und Größe sollen durch einen Tabulator (`'\t'`) getrennt werden. Am Ende der Ausgabe soll die Gesamtzahl der ausgegebenen Einträge, sowie deren Gesamtgröße ausgeben werden. Ignorieren Sie für die Ermittlung der Gesamtanzahl und -größe alle Einträge, bei denen es sich nicht um eine reguläre Datei handelt. (`lstat(2)`)

(c) Werten Sie nun die Parameter `argv` aus. Alle übergebenen Parameter sollen als Verzeichnispfade interpretiert und wie in (a) und (b) beschrieben ausgegeben werden. Die Ausgabe eines Verzeichnisses soll mit `'<Verzeichnisname>:\n'` beginnen. Wird kein Parameter übergeben, soll das aktuelle Verzeichnis ausgegeben werden.

Hinweise:

- Ihr Programm muss nur Pfade und Dateinamen bis zu einer Gesamtlänge von 1024 Zeichen[1] behandeln können. Achten Sie bei zu langen Pfaden und Dateinamen auch hier auf eine entsprechende Fehlermeldung.
- Die Funktionen zur Behandlung von Zeichenketten aus `string.h` sind für diese Aufgabe hilfreich.
- Begründen Sie die Verwendung von allen `volatile` Variablen. Wenn für mehrere Variablen die selbe Begründung gilt, dürfen Sie diese gemeinsam begründen.
- Im Verzeichnis `/proj/i4spic/<login>/pub/aufgabe7/` befindet sich die Datei `printdir`, welche eine Beispielimplementierung enthält.
- Achten Sie auf aussagekräftige Fehlermeldungen, die alle auf dem Standardfehlerkanal ausgegeben werden sollen. (`fprintf(stderr,...)(3)` / `perror(3)`)
- Testen Sie Ihr Programm auch mit `valgrind`. Dies kann bei der Suche nach Fehlern helfen. *suppressed Errors* können ignoriert werden. Weitergehende Fehlermeldungen erhalten Sie, wenn Sie `valgrind` mit den Flags `--leak-check=full --show-reachable=yes` aufrufen und das zu analysierende Binary mit Debug-Symbolen bauen.
- Ihr Programm muss mit dem folgendem Aufruf übersetzen:
  `gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o printdir printdir.c`
  Diese Konfiguration wird zur Bewertung herangezogen.
- Funktionen der `libc`, für die wir keine Fehlerbehandlung erwarten, sind online in der Linux `libc`-Doku entsprechend markiert.
- Sie können ein `Makefile` schreiben, das eine Anleitung für den Bau des Programms mit dem Tool `make` enthält. Hierfür legen Sie eine Datei `Makefile` in Ihrem Aufgabenordner (`aufgabe7/`) an. In die erste Zeile schreiben Sie:
  `CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3`
  Der Bau erfolgt im Terminal durch `make printdir` oder dem `make` Button in der SPiC-IDE.

[1] Alternativ kann auch `PATH_MAX` aus der `limits.h` verwendet werden.

Beispielausgabe

```
$ cd /proj/i4spic/<login>/pub/aufgabe7
$ ./printdir test/first_path test/second_path
test/first_path:
157     file2.txt
127     file1.txt
4096    test_dir
2 Dateien; 284 Bytes
test/second_path:
115     fileB.txt
4096    dir2
116     fileA.txt
4096    dir1
2 Dateien; 231 Bytes
```

Sie können Ihre Implementierung für das selbe Verzeichnis aufrufen und sollten eine identische Ausgabe erhalten:

```
$ cd /proj/i4spic/<login>/aufgabe7/
$ ./printdir /proj/i4spic/<login>/pub/aufgabe7/test/first_path \
             /proj/i4spic/<login>/pub/aufgabe7/test/second_path
[...]
```

Dies alleine ersetzt jedoch kein ausgiebiges Testen mit eigenen Verzeichnisstrukturen.

Abgabezeitpunkt

alle Gruppen	17.01.2021	18:00:00