Betriebssysteme
bs@ibr.cs.tu-bs.de
Wintersemester 2025/26

Technische Universität Braunschweig

Übungsblatt 3: Dateisysteminteraktionen

Abgabetermin: Donnerstag, 04.12.2025 (23:59 Uhr)

Allgemeine Hinweise zu den BS Übungen

- Die Aufgaben sind alleine zu bearbeiten, allerdings ist Partnerarbeit (maximal zu zweit) möglich. Beide Partner müssen aber in ihrem Git-Repository abgeben und eine `PARTNER`-Datei mit dem Name des Partners im Repository anlegen.
- Jeder Teilnehmende muss dabei in der Lage sein, die gesamte Lösung zu erläutern.
- Zur Versionsverwaltung wird jedem Teilnehmenden ein Git-Repository zur Verfügung gestellt, welches auch zur Abgabe verwendet wird. Der Zugriff auf das Repository erfolgt über das IBR GitLab unter `https://gitlab.ibr.cs.tu-bs.de`.
- Die Abgabe ist spätestens zur Deadline durch ein `git push` zu tätigen. Es sollte sichergestellt werden, dass die Änderungen vorher auch durch ein `git commit` gesichert wurden.
- Eine Aufgabe kann beliebig oft abgegeben werden; es gilt die letzte *rechtzeitige* Abgabe (der letzte rechtzeitige Commit auf dem `main` Branch).
- Die abgegebenen Programme werden auf Ähnlichkeit mit anderen Programmen desselben Semesters und früherer Semester überprüft. Bei starken Übereinstimmungen behalten wir uns ein Ausschluss vom Übungsbetrieb und eine Meldung an den Prüfungsausschuss vor.
- Die Lösungen der Programmieraufgaben müssen im GitLab CI/CD lauffähig sein. Zusätzlich kann das System `x1.ibr.cs.tu-bs.de` zum Testen genutzt werden. Auf diesem können Sie sich per SSH mit ihrem IBR Account anmelden.
- Für die Bearbeitung kann auf einem beliebigen Linux-System gearbeitet werden. Allerdings empfehlen wir das frühzeitige Testen der Abgabe durch ein `git push` und die Inspektion der Test Pipeline im GitLab.
- Die Programmieraufgaben werden automatisch durch das bereitgestellte Testsystem per GitLab CI/CD ausgewertet. Es müssen mindestens $50\,\%$ der verfügbaren Testcase-Punkte erreicht werden.
- Die Testcase-Punkte können kontinuierlich über die Ausgabe des `run-tests` Jobs aus der GitLab CI/CD Pipeline eingesehen werden. Gewertet wird der letzte Pipeline-Status auf dem von uns zur Abgabe angelegten Abgabebranch (z.B. `abgabe1`), überprüfen Sie diesen gegebenenfalls.
- Sollten Sie $50\,\%$ der Punkte erreicht haben werden Sie ggf. für ein Code-Interview ausgewählt. Sollten Sie zum Code-Interview nicht erscheinen oder ihre Lösung nicht ausreichend erklären können wird der Zettel als „ungenügend“ bewertet.
- **KI Richtline**: Der Einsatz von KI-gestützten Tools ist bei der Bearbeitung der Hausaufgaben streng untersagt, da die eigenständige Erarbeitung der Lösungen inhärentes Lernziel ist. Es ist zu beachten, dass wir alle Abgaben als Prüfungsdokumente archivieren und es auch bei einer nachträglichen Prüfung zu einer rückwirkenden Aberkennung der Studienleistung kommen kann.

Aufgabe 1: Traversieren des Dateibaums

In dieser Aufgabe soll das Programm `crawl` implementiert werden. Die Aufgabe besteht dazu aus zwei Modulen. Im ersten Teil wird der Argumentenparser entwickelt, der die Verarbeitung der Eingabeargumente übernimmt. Im zweiten Teil wird die Hauptfunktionalität des `crawl`-Programms implementiert. Das fertige Programm arbeitet ähnlich wie die UNIX-Werkzeuge **find(1)** und **grep(1)** und durchsucht dabei Verzeichnisbäume und Dateiinhalte. Es wird wie folgt aufgerufen:

```
crawl path... [-maxdepth=n] [-name=pattern] [-type={d,f}] [-size=[+|-]n] [-line=string]
```

Das heißt: Das Programm erhält einen oder mehrere Pfadnamen (Datei- oder Verzeichnisnamen) auf der Kommandozeile, optional gefolgt von Parametern und Suchausdrücken. Dazu soll implementiert werden:

(a) **Das `argumentParser`-Modul:** Erstellen Sie ein Modul zum Verarbeiten der Kommandozeilen-Argumente. Das Modul soll aus dem gegebenen `argv` eines C-Programms das aufgerufene Programm, die Optionen im Format `-key=value` und die weiteren gegebenen Argumente herausarbeiten. Die Schnittstelle ist in der Datei `argumentParser.h` vorgegeben und darf nicht verändert werden. Beachten Sie außerdem, dass die Funktionen des Moduls keine allgemeine Fehlerbehandlung zulassen – insbesondere dürfen also keine **malloc(3)**-Aufrufe oder vergleichbare Funktionen innerhalb des Modules verwendet werden.

Der Argumentenparser akzeptiert nur Eingaben, die dem folgenden Format entsprechen:

```
COMMAND [ARGUMENT...] [-option=value...]
```

Das heißt: Der erste Eintrag (der Programmname) ist gefolgt von der Liste der Argumente. Zuletzt können beliebige viele Optionen folgen. Diese beginnen mit einem - gefolgt von ihrem Namen, der mit einem = vom Wert separiert wird. Ein Vertauschen der Reihenfolge wird vom Argumentenparser nicht akzeptiert. Bearbeiten Sie diese Aufgabe in der Datei `argumentParser.c`.

(b) **Das `crawl`-Modul:** Das Programm `crawl` soll gegebene Verzeichnisse und Dateien rekursiv durchsuchen und die gefundenen Einträge auf der Standardausgabe auflisten. Hierbei soll sich das Suchverhalten und die Ausgabe mit Optionen einschränken lassen. Alle nicht ignorierten Einträge werden mit ihrem relativen Pfad auf die Standardausgabe in jeweils einer eigenen Zeile ausgegeben. Bearbeiten Sie diese Aufgabe in der Datei `crawl.c`.

Es wird empfohlen die Implementierung schrittweise in den folgenden Arbeitspaketen zu bearbeiten:

1. **Rekursiver Verzeichnisdurchlauf:** Beginnen Sie damit, dass ihr `crawl` rekursiv alle als Argumente gegebenen Verzeichnisse durchsucht. Verwenden sie hierzu den Argumentenparser aus dem vorherigen Aufgabenteil. Alle Dateien, die weder regulär noch Verzeichnis sind, sollen ignoriert werden, symbolische Verknüpfungen sollen nicht verfolgt werden. Die speziellen Einträge `.` und `..` werden bei der Tiefensuche ignoriert, können aber als Pfade auf der Kommandozeile übergeben werden und dienen dann als Wurzelverzeichnis der Tiefensuche.

2. **Einschränkung der Suchtiefe:** Die Option `-maxdepth=n` schränkt die Suchtiefe ein. Dies bedeutet, dass nur Verzeichnisse bis zu der gegebenen Tiefe $n \geq 0$ durchsucht werden sollen (0: Nur die auf der Kommandozeile übergebenen Pfade selbst werden untersucht, aber nicht deren Inhalt; 1: der Inhalt auf der Kommandozeile übergebener Verzeichnisse wird untersucht, aber nicht der ihrer Unterverzeichnisse, usw.). Ist diese Option nicht angegeben, so soll das Programm bis an die Blätter der Verzeichnishierarchie absteigen.

3. **Einschränkung der Ausgabe:** Falls einschränkende Optionen gegeben sind (siehe folgende Auflistung), so sollen nur diejenigen Einträge ausgegeben werden, auf die alle Einschränkungen zutreffen. Achtung: Die Suchausdrücke schränken nur die Ausgabe von `crawl` ein. Wenn also die Suchausdrücke nicht auf ein gefundenes Verzeichnis passen, die maximale Pfadtiefe aber noch nicht erreicht ist, so wird dieses Verzeichnis dennoch durchsucht.

-name= Mit diesem Ausdruck kann der Name eines Verzeichniseintrages (also nur die letzte Komponente des Pfades) mit einem Shell-Wildcard-Muster verglichen werden. Verwenden Sie für den Vergleich die Funktion **fnmatch(3)**.

-type= Hiermit kann die Ausgabe auf Verzeichnisse (`d`) oder reguläre Dateien (`f`) eingegrenzt werden.

-size= Hiermit kann nach Einträgen mit einer bestimmten Dateigröße (in Bytes) gesucht werden. Wird als Präfix das Zeichen – bzw. + vor der Größenangabe verwendet, so wird nach Einträgen gesucht, die kleiner bzw. größer als die angegebene Größe sind.

-line= Hiermit kann nach regulären Dateien gesucht werden, deren Inhalt den angegebenen (POSIX Extended) regulären Ausdruck enthält. Ist diese Option gegeben, so wird die entsprechende Zeile der Datei ausgegeben. Zu jeder passenden Zeile wird der Dateiname und die entsprechende Zeilennummer ausgegeben. Dateien, die keine passende Zeile enthalten sollen nicht ausgegeben werden. Lesen sie hierzu die Datei Zeilenweise und verwenden sie **regex(3)** zum Prüfen der Bedingung.

Hilfreiche Manual-Pages: **lstat(2)**, **opendir(3)**, **readdir(3)**, **closedir(3)**, **basename(3)**, **fnmatch(3)**, **strtol(3)**, **getline(3)**, **regex(3)**, **hier(7)**.

Weitere Hinweise zur Programmieraufgabe

Die `argumentParser`-Schnittstelle

Folgende Schnittstelle ist durch die Datei `argumentParser.h` vorgegeben:

- `int initArgumentParser(int argc, char* argv[])`: Initialisierungsfunktion. Ihr werden die Anzahl der Kommandozeilenargumente und das Feld mit diesen übergeben. Sie prüft die Validität der übergebenen Einträge und kann Organisationsstrukturen für die weiteren Funktionsaufrufe vorbereiten. Die Übergebenen Felder dürfen hierbei explizit durch den Parser verändert werden. Im Fehlerfall wird -1 zurückgegeben, im Erfolgsfall 0.
- `char *getCommand(void)`: Zugriffsfunktion für den Programmnamen. Diese Funktion liefert einen Zeiger auf den Namen des Programms. Dies ist der erste Eintrag in `argv`.
- `int getNumberOfArguments(void)`: liefert die Anzahl der Argumente, die in der gegebenen `argv`-Struktur gefunden wurden.
- `char *getArgument(int i)`: liefert einen Zeiger auf das Argument mit entsprechendem Index, falls vorhanden. Ansonsten wird `NULL` zurückgegeben.
- `char *getValueForOption(char *optionName)`: liefert einen Zeiger auf den Wert der Option mit gegebenem Namen falls diese vorhanden ist. Ansonsten wird `NULL` zurückgegeben.

Referenzimplementierung & Beispielausgabe

Eine Referenzimplementierung ist unter `/ibr/courses/ws2526/bs/blatt3/crawl` aufrufbar.

```
x1:/ibr/courses/ws2526/bs/blatt3$ ./crawl . -name='*.log'
./testdir/resources/utility/srvLog/2014-03-03.log
./testdir/resources/utility/srvLog/2014-03-02.log
./testdir/resources/utility/ProgrammAufrufe.log
./testdir/resources/Downloads/2014-03-03.log
./testdir/resources/Downloads/2014-03-02.log


x1:/ibr/courses/ws2526/bs/blatt3$ ./crawl . -line=print -name="m*"
./testdir/resources/src/myLength.c:4:    printf("I'm a program with a short name\n");
./testdir/resources/src/minecraft.c:4:    printf("You should not play right now!\n");
```