Aufgabe 5: paffin (16.0 Punkte)

Schreiben Sie ein Programm `paffin` (**Pa**rallel **Fi**le Fi**n**der), welches in regulären Dateien nach Zeilen, die eine bestimmte Zeichenkette (*string*) enthalten, sucht und den entsprechenden Pfad zur Datei, die Zeilennummer und die Zeile ausgibt.

Das Programm wird wie folgt aufgerufen:
`paffin <string> <max-grep-threads> <trees...>`

Die als Parameter übergebenen Verzeichnisbäume (*trees*) werden jeweils in einem eigenen Thread rekursiv durchsucht (*crawl-Thread*). Für jede gefundene reguläre Datei wird wiederum ein eigener Thread (*grep-Thread*) erzeugt (**pthread_create(3)**), welcher ihren Inhalt zeilenweise durchsucht (**fopen(3)**, **fgets(3)**, **strstr(3)**). Die Anzahl der aktiven grep-Threads wird durch einen Parameter (*max-grep-threads*) limitiert. Falls bereits so viele Threads aktiv sind, wird vor dem Erzeugen eines weiteren Threads **passiv** gewartet, bis ein Thread terminiert.

Während der Suche erfolgt noch **keine** Ausgabe. Gefundene Zeilen werden in einer (gegebenen) Listenimplementierung zwischengespeichert (`enqueue()`). Die Listenimplementierung darf **nicht** nebenläufig genutzt werden und muss entsprechend geeignet mit anderen Threads synchronisiert werden. Während der Suche wartet der Hauptthread **passiv**, bis die Suche beendet ist. Hierzu ist er ebenfalls in geeigneter Weise mit den anderen Threads zu synchronisieren.

Bei Zugriffen auf globale Datenstrukturen muss auf korrekte Synchronisation geachtet werden. Langsame Operationen (z.B. **printf(3)**) dürfen dabei nicht in kritischen Abschnitten ausgeführt werden.

Nachdem die Suche abgeschlossen ist (alle *crawl-* und *grep*-Threads haben sich beendet), entnimmt der Hauptthread alle gefundenen Elemente der Liste (`dequeue()`) und gibt sie aus.

Struktur Ihrer Abgabe:

Nutzen Sie zur Bearbeitung der Aufgabe folgende, im Skeleton vorgegebene, Struktur:

- Die Hauptfunktion `main()` initialisiert die notwendigen Datenstrukturen, startet einen *crawl-Thread* pro übergebenem Verzeichnisbaum und wartet anschließend passiv auf das Ende der Suche. Sobald sich alle Threads beendet haben, übernimmt `main()` die Ausgabe der gefundenen Zeilen und das Aufräumen/Freigeben der verbleibenden Ressourcen.
- Die Funktion `void* processTree(void* path)` ist der Einstiegspunkt für die *crawl-Threads* und ruft die Funktion `processDir()` zum Durchsuchen von `path` auf.
- Die Funktion `void* processDir(char* path)` iteriert über die einzelnen Elemente im übergebenen Pfad `path` und ruft für jedes gefundene Element die Funktion `processEntry()` auf.
- Die Funktion `void* processEntry(char* path, struct dirent* entry)` prüft ob es sich beim übergebenen Element `entry` um eine reguläre Datei oder um ein Verzeichnis handelt:
  - Für Verzeichnisse wird die Funktion `processDir()` zum rekursiven Abstieg aufgerufen.
  - Für reguläre Dateien wird jeweils ein eigener Thread (*grep-Thread*) erzeugt, der die Funktion `processFile()` zum Durchsuchen der Datei aufruft. Sollten bereits `max-grep-threads` Threads laufen, dann wird die Erzeugung eines neuen Threads verzögert.
  - Einträge, die weder reguläre Datei noch Verzeichnis sind, werden ignoriert.
- Die Funktion `void* processFile(void* path)` ist der Einstiegspunkt für die *grep-Threads* und durchsucht die übergebene Datei `path` zeilenweise nach dem per Befehlszeile übergebenen `string` und fügt passende Zeilen in die Liste ein.

Hinweise zur Aufgabe:

- Beachten Sie, dass **korrekte Synchronisation** eines der zentralen Ziele der Aufgabe ist!
- Ihre Lösung muss der vorgegebenen Programmstruktur (siehe Programmskelett und Beschreibung oben) entsprechen, es steht Ihnen aber frei Hilfsfunktionen zu implementieren.
- Verwenden Sie für die Synchronisation das vorgegebene Semaphoren-Modul (`sem.o`, `sem.h`).
- Verwenden Sie als Listenimplementierung das vorgegebene Modul (`list.o`, `list.h`).
- Alle erzeugten Threads sollen im *detached-state* (**pthread_detach(3)**) ausgeführt werden.
- Zum Übersetzen des Programmes ist das zusätzliche Compiler Flag `-pthread` notwendig. Ihr Programm muss also mit den folgenden Flags kompilieren:
  `-std=c11 -pedantic -Wall -Werror -D_XOPEN_SOURCE=700 -pthread`
  Diese Flags werden zur Bewertung herangezogen.

- Gehen Sie (ohne weitere Prüfung) davon aus, dass die Dateien keine Zeilen enthalten, die länger als 4096 (`MAX_LINE`) Zeichen (exklusive `\n` und `\0`) sind.
- Zum Testen kann `paffin` mit den Verzeichnisbäumen `/usr/share/doc/*` aufgerufen werden.
- Neben der `dequeue()`-Methode stellt die Listenimplementierung zusätzlich die Funktion `dequeue_fancy()` zur Verfügung. Dabei wird der zurückgegebene String für menschliche Benutzer ansprechender aufbereitet. Die Funktion und Parameter werden genauer in der Datei `list.h` beschrieben. Die Verwendung von `dequeue_fancy()` für diese Aufgabe ist **optional**.
- Eine Referenzimplementierung finden Sie im zip-Archiv.

Hinweise zur Abgabe:

Bearbeitung:	Zweiergruppen
Abgabezeitpunkt:	**27.01.2023**, 17:30 Uhr
Erforderliche Dateien:	`paffin.c` (16 Punkte)