Aufgabe 5: palim (22.0 Punkte)

Schreiben Sie ein Programm `palim` (**Pa**rallel **Li**ne **M**atcher), welches in regulären Dateien die Anzahl der Zeilen, die eine bestimmte Zeichenkette (*string*) enthalten, ausgibt.

Das Programm wird wie folgt aufgerufen:
`palim <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 Ausführung soll das Programm folgende Informationen in einer Zeile ausgeben:

- Anzahl der Zeilen, die Treffer enthalten (*lineHits*)
- Anzahl der durchsuchten Zeilen (*lines*)
- Anzahl der Dateien mit mindestens einem Treffer (*fileHits*)
- Anzahl der durchsuchten Dateien (*files*)
- Anzahl der durchsuchten Verzeichnisse (*dirs*)
- Anzahl der aktuell aktiven grep-Threads (*activeGrepThreads*)

Die Ausgabe soll folgendes Format besitzen:
`lineHits/lines` `lines,` *`fileHits/files`* `files,` *`dirs`* `directories,` *`activeGrepThreads`* `active threads`

Die Ausgabe der Statistiken soll kontinuierlich durch den Hauptthread (*main()*) aktualisiert werden sobald sich eine Information ändert. Zwischenzeitlich wartet der Hauptthread **passiv** auf Änderungen. Hierzu ist er 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.

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 Aktualisierungen der Statistiken. Sobald sich alle Threads beendet haben übernimmt `main()` 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` (**opendir(3)**, **readdir(3)**, **closedir(3)**) 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 (**stat(2)** / **lstat(2)**):
  - 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`.
- Die vier `process*()` Funktionen aktualisieren außerdem jeweils die entsprechenden Statistiken.

Hinweise zur Aufgabe:

- Beachten Sie, dass **korrekte Synchronisation** eines der zentralen Ziele der Aufgabe ist!
- Sie dürfen davon ausgehen, dass sich `readdir()` mit verschiedenen Verzeichnis-Streams (`DIR *`) *thread-safe* verhält.
- 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`).
- Der Cursor kann durch Ausgabe des Zeichens `\r` an den Anfang der Zeile zurückgesetzt werden.
- Die Ausgabe einer Zeichenkette kann mittels **fflush(3)** erzwungen werden.
- 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.
- Eine Referenzimplementierung finden Sie im zip-Archiv.
- Zum Testen kann palim mit den Verzeichnisbäumen `/usr/share/doc/*` aufgerufen werden.

Hinweise zur Abgabe:

Bearbeitung:	Zweiergruppen
Abzugebende Dateien:	`palim.c` (22 Punkte)
Abgabezeitpunkte nach Tafelübungsgruppe:	T01-T02: 05.07.2023, 17:30 Uhr T03-T06: 08.07.2023, 17:30 Uhr T07-T09: 09.07.2023, 17:30 Uhr