# Aufgabe 9: recgrep (14 Punkte) Bearbeitung in Zweier-Gruppen 8.6.2010

**Nur für Mathematiker**

Implementieren Sie ein Programm **recgrep**, das ähnlich dem UNIX-Kommando **grep -rn** arbeitet. Das Programm wird wie folgt aufgerufen:

```
recgrep searchString path... [-maxdepth n]
```

Das Programm ist ähnlich parametriert wie das Programm **crawl** aus Aufgabe 6 und soll auch auf dessen Basis implementiert werden. Die Aufgabe ist in vier Module unterteilt, wobei das *recgrep*-Modul (Datei *recgrep.c*), welches auch die main-Funktion enthält, zu einem großen Teil der crawl-Aufgabe entspricht (Sie sollten Ihre crawl-Aufgabe als Basis für dieses Modul verwenden). Für alle vier Module finden Sie eine Binärvorgabe in */proj/i4sp/pub/aufgabe9*. Sie können so die Module getrennt voneinander entwickeln und mit unseren Binärversionen linken, um Ihr eigenes Modul zu testen. Die Schnittstelle der Module *grep*, *bbuffer* und *sem* ist verbindlich und findet sich in den vorgegebenen Header-Dateien sowie auf der Webseite. Die externe Schnittstelle darf nicht verändert werden, um Interoperabilität zu gewährleisten.

### a) Rekursive Suche nach Dateien, die einen Suchstring enthalten

Erzeugen Sie zunächst das *recgrep*-Modul aus Ihrer crawl-Aufgabe. Nach Vorliegen des Korrekturergebnisses sollten Sie alle dort vorhandenen Fehler ausbessern. Die im Vergleich zur crawl vorzunehmenden Änderungen beschränken sich auf den Aufruf der Initialisierungs- und Zerstörungsroutinen des *grep*-Moduls (siehe API-Dokumentation), sowie den Aufruf der *grep_addJob*-Funktion anstelle des Ausgebens für jede gefundene reguläre Datei.

### b) Mehrfädiges grep-Modul

Implementieren Sie das Modul *grep* wie in der Header-Datei beschrieben. Das Modul muss noch weitere Funktionen wie die Arbeitsfunktion der Arbeiterthreads enthalten, diese sollen aber nicht Teil der externen Schnittstelle werden. Das Modul bietet eine Funktion, die das Hinzufügen von Jobs (Dateien, die nach dem Suchmuster durchsucht werden sollen) ermöglicht. Neue Jobs werden in einen Ringpuffer eingetragen (Modul *bbuffer*, Teilaufgabe c)). Eine feste Zahl von Arbeiterthreads wird einmalig zur Initialisierung des Moduls erzeugt und entnimmt dann Jobs aus dem Ringpuffer. Für jeden Job liest ein Arbeiterthread zeilenweise die komplette Datei und durchsucht jede Zeile nach dem Vorkommen des Suchstrings. Für jede passende Zeile wird der Dateiname, die Zeilennummer sowie der Zeileninhalt auf die Standardausgabe ausgegeben. Nach Ende der Bearbeitung schliesst der Thread die Datei und gibt die restlichen Ressourcen des Jobs frei. Danach reiht sich der Thread wieder am Ringpuffer zur Entnahme des nächsten Jobs ein. Implementieren Sie auch eine End-Synchronisation, zu der vor Programmende alle noch ausstehenden Jobs abgearbeitet werden, die Threads sich kontrolliert beenden und alle noch belegten Ressourcen wieder freigegeben werden, bevor sich das Hauptprogramm beendet.

### c) Semaphore

Implementieren Sie zählende Semaphore zur Koordinierung von POSIX-Threads (pthread_mutex_lock(3), pthread_cond_wait(3)) im Modul *sem* (Datei *sem.c*).

### d) Ringpuffer

Implementieren Sie einen Ringpuffer im Modul *bbuffer* (Datei *bbuffer.c*), der für einen einzelnen Produzententhread und mehrere Konsumententhreads konzipiert ist. Verwenden Sie die Semaphorimplementierung aus Teilaufgabe c) zur Synchronisation der Über- und Unterlaufsituationen, so dass Produzent bzw. Konsumenten beim Einfügen in einen vollen bzw. bei der Entnahme aus einem leeren Puffer blockieren.

**Abgabe: bis spätestens Freitag, 9.7.2010, 12:00 Uhr**


Bei Fragen zur Aufgabenstellung bitte Mail an *i4sp@informatik.uni-erlangen.de*