Hochschule RheinMain
Studiengang Medieninformatik
Sommersemester 2018
Prof. Dr. Wolfgang Weitz

Betriebssysteme und Rechnerarchitektur

Übungsblatt vier (muss pfingstbedingt für zwei Wochen reichen)

14.05.2018

Aufgabe:
Die behandelten Lowlevel-Dateioperationen haben leider keinen Begriff von (zeilenstrukturierten) Textdateien, sie kennen nur Bytefolgen. Bitte implementieren Sie *ohne* Rückgriff auf die C-Stream-basierten Dateioperationen der C-Standardbibliothek (also ziemlich alles, was mit „f“ anfängt bzw. mit `FILE*` arbeitet) folgende Operationen, die ein zeilenweises Lesen von einem Descriptor ermöglichen.

Sie können **nicht** davon ausgehen, dass der Eingabestrom `seek`()-bar ist (außer in der Funktion `buf_seek`()) - denken Sie z.B. an die Standardeingabe, Filedescriptor 0). Lesen Sie bitte mit `read`() vom Eingabedescriptor in Blöcken von `LINEBUFFERSIZE` Bytes (also *nicht* z.B. Einzelbyte-weise), puffern Sie die gelesenen Daten im `buffer`-Feld des `LineBuffer` und holen Sie z.B. bei `buf_readline`() nach Bedarf Daten aus diesem Buffer.

Das Feld `end` enthält die Anzahl der tatsächlich belegten Bytes in `buffer`, während `here` die Indexposition des nächsten zu verarbeitenden Bytes im buffer ist. Sobald dieser ausgelesen ist (also `here` >= `end`), lesen Sie die nächsten `LINEBUFFERSIZE`-vielen Bytes in `buffer` ein usw.

Die zugehörigen Deklarationen finden Sie in der Header-Datei `linebuffer.h` im read.MI.

- `LineBuffer *buf_new(int descriptor, const char *linesep)`
  erzeugt (dynamisch) eine Struktur `LineBuffer`, um (später) vom übergebenen (File-)Descriptor zeilenweise lesen zu können. Zeilenenden werden durch die Zeichenfolge `linesep` markiert (C-String, Zeilentrenner können mehrere Bytes lang sein). Die Felder der Struct sind in `linebuffer.h` erklärt und sollten alle mit Null initialisiert werden bis auf `descriptor`, `linesep` und `linelen`, die aus den Parametern belegt werden.
- `void buf_dispose(LineBuffer *b);`
  Speicher für mit buf_new() bezogene `LineBuffer`-Struktur freigeben.
- `int buf_readline(LineBuffer *b, char *line, int linemax);`
  Vom im `LineBuffer` gemerkten Descriptor unter Berücksichtigung des verwendeten Zeilenseparators eine Zeile in `line` einlesen (ergibt C-String), max. `linemax` Bytes, **ohne** den Zeilenseparator. Rückgabewert ist -1 bei Eingabeende, ansonsten die Offset-Position des Zeilenanfangs im Eingabe-Bytestrom (nicht im aktuellen Buffer). Verwenden Sie zur Positionsermittlung bitte nicht `lseek`() o.ä., um auch mit Eingabe-Descriptoren arbeiten zu können, die nicht seek-bar sind. Berücksichtigen Sie bitte, dass die Zeilentrenner-Zeichenfolge zufällig auf zwei aufeinanderfolgende `read`()-Blöcke gesplittet sein könnte (Ende eines read() / Rest am Anfang des nächsten).
- `int buf_where(LineBuffer *b);`
  liefert die aktuelle Byte-Offset-Position im Eingabe-Bytestrom, zeilenübergreifend zu ermitteln durch Mitzählen der eingelesenen Bytes (incl. Zeilentrenner), wird bei `buf_new`()

als Null angenommen. Verwenden Sie nicht `lseek()` o.ä., um auch mit Eingabe-Descriptoren arbeiten zu können, die nicht seek-bar sind.

- `int buf_seek(LineBuffer *b, int seekpos);`
  Positioniert (absolut) auf Offset `seekpos` innerhalb des Eingabe-Bytestroms. Dies ist die einzige Funktion in dieser Aufgabe, wo Sie die `lseek()` verwenden dürfen. Rückgabewert ist der Rückgabewert von `lseek()`. Die Funktion setzt `bytesread` in LineBuffer auf `seekpos` sowie `here` und `end` auf Null.

Aufgabe:
Aufbauend auf den Ergebnissen der vorigen Aufgabe basteln wir uns jetzt einige Funktionen, um eine in Abschnitte eingeteilte Textdatei zu indizieren. Die zugehörigen Deklarationen finden Sie in `fileindex.h`.
Die betrachteten Textdateien bestehen aus Abschnitten. Jeder Abschnitt beginnt mit einer Separatorzeile, die mit einer übergebenen Separator-Zeichenfolge *beginnt*, und endet mit einer Leerzeile. Weder Separatorzeile noch abschließende Leerzeile gehören zum eigentlichen Inhalt des Abschnitts, sie begrenzen ihn nur. Leerzeilen, auf die nicht direkt eine Separatorzeile (oder Dateiende) folgt, gehören zum umgebenden Abschnitt, sind also „Inhalt“.

Die `FileIndex`-Struktur enthält eine Liste von `FileIndexEntry`-Strukturen (`entries`) , welche der Reihe nach die Abschnitte der zugrunde liegenden Datei beschreiben. Der Einfachheit halber enthält `FileIndex` auch die Anzahl der Abschnitte (`nEntries`) und deren Gesamtgröße in Bytes (`totalSize`) einschließlich Zeilentrennern. Da die Begrenzungszeilen der Abschnitte nicht zum Inhalt zählen, ist diese kleiner als die Dateigröße.

Ein `FileIndexEntry` enthält die in `fileindex.h` beschreibenen Felder. Bitte implementieren Sie die Funktionen aus `fileindex.h`:

- `FileIndex *fi_new(const char *filepath, const char *separator)`
  indiziert angegebene die Datei `filepath.` Trennzeilen sind solche, die mit der Zeichenkette `separator` (C-String) anfangen. Wie erwähnt gehören Trennzeilen nicht zum Inhalt. Ergebnis ist ein Zeiger auf die erzeugte `FileIndex`-Struktur oder `NULL` bei Fehler.
- `void fi_dispose(FileIndex *fi)`
  gibt Speicher für `FileIndex`-Struktur frei (incl. Liste der `FileIndexEntry`-Knoten).
- `FileIndexEntry *fi_find(FileIndex *fi, int n)`
  liefert einen Zeiger auf den `FileIndexEntry` zum Listenelement Nr n (Zählung beginnt wie oben erwähnt bei Eins!) oder `NULL` bei Fehler.
- `int fi_compactify(FileIndex *fi)`
  löscht alle Abschnitte, deren `del_flag` true ist, aus der Datei (wie bei der Binärdatei-Übung schon praktiziert am besten durch Umkopieren und Umbenennen) und liefert 0 bei Erfolg und !=0 bei Fehler.

Hinweis: Sie können insbesondere die Korrektheit Ihrer `seekpos`- und `size`-Berechnungen sehr einfach testen. Lassen Sie sich in einer `main()` einmal eine Text-Testdatei indizieren und danach die Inhalte Ihrer Indexstruktur ausgeben. Verwenden Sie dann Python, um zu sehen, ob Ihre Positions-/Größenwerte genau den Inhaltsteil des gesuchten Abschnitts in der Textdatei liefern.

Beispiel: Ihr C-Programm sagt, dass eine Eingabedatei „`input.txt`“ einen Abschnitt hat, der bei `seekpos` 710 beginnt und 123 Bytes lang ist (`size`). Check mit Python:

```
print(open(„input.txt“).read()[710:710+123])
```