Übungen zur Vorlesung
**Betriebssysteme I**
Studiengang Informatik
WS 2010/2011

AG Betriebssysteme FB3
Prof. Dr. Jan Peleska, Florian Lapschies

Abgabetermin: 2.12.2010

# Übungszettel 2

## Aufgabe 1: Ringpuffer Bibliothek

In der Vorlesung wurde Euch das Konzept des Ringpuffers vorgestellt. Dieses ermöglicht einem Prozess, von ihm produzierte Daten in den Puffer abzulegen, die dann in FIFO-Reihenfolge von einem weiteren Prozess ganz ohne weitere Synchronisationmechanismen ausgelesen werden können.

In diesem Übungszettel soll nun eine C-Bibliothek für Ringpuffer programmiert werden. Wie in der Vorlesung beschrieben, sollen hierbei in einem Ringpuffer nicht die Datenpakete selbst, sondern lediglich Verweise auf diese gespeichert werden.

Die Ringpuffer-Bibliothek soll durch Einbindung einer Header-Datei `ringbuffer.h` nutzbar sein. Einen Ringpuffer definieren wir uns als Datentyp `ringbuffer_t`:

```
typedef struct ringbuffer {
  unsigned int rIdx;
  unsigned int wIdx;
  unsigned int mask;
  void *buffer[0];
} ringbuffer_t;
```

`rIdx` und `wIdx` ist hierbei der Lese- bzw. der Schreibindex im Ringpuffer. Anstatt beim Weitersetzen dieser Indizes die Modulo-Operation zu nutzen, verwenden wir das auf vielen Hardwareplattformen effizientere bitweise UND mit einer Bitmaske `mask`. Damit dies funktioniert muss die Größe des Ringpuffer-Arrays natürlich eine Zweierpotenz sein. Direkt nach diesen Verwaltungsfeldern folgt dann das Ringpuffer-Array `buffer` mit den Verweisen auf die zu speichernden Daten.

### Erstellen eines Ringpuffers

```
ringbuffer_t *ringbuffer_create(unsigned int nb_entries);
```

Mit der Funktion `ringbuffer_create` wird ein Ringpuffer, der mindestens `nb_entries` Einträge speichern kann, erstellt. Um nicht auf die Modulo-Operation angewiesen zu sein, muss die Größe für das Ringpuffer-Array zur nächst-größeren Zweierpotenz hochgerundet werden.

Die Funktion gibt bei Erfolg den neu erstellten Ringpuffer und im Fehlerfall `NULL` zurück.

### Freigeben eines Ringpuffers

```
void ringbuffer_destroy(ringbuffer_t *rb);
```

`ringbuffer_destroy()` gibt einen übergebenen Ringpuffer wieder frei.

### Schreiben in einen Ringpuffer

```
int ringbuffer_put(ringbuffer_t *rb, void *in);
```

Mit der Funktion `ringbuffer_put()` wird ein Datenpaket `in` in den Ringpuffer `rb` gespeichert. Im Ringpuffer wird lediglich ein Verweis auf das Datenpaket gespeichert. Das Datenpaket selbst wird nicht kopiert. Im Erfolgsfall gibt die Operation `0` zurück. Sollte der Ringpuffer voll sein, wird `-2` zurückgegeben. Bei anderen Fehlern wird `-1` zurückgegeben.

### Lesen von einem Ringpuffer

```
int ringbuffer_take(ringbuffer_t *rb, void **out);
```

Die Funktion `ringbuffer_take()` liest aus einem Ringpuffer `rb` das nächste Datenpaket und speichert es nach `out`. Im Erfolgsfall gibt die Operation `0` zurück. Sollte der Ringpuffer leer sein, wird `-2` zurückgegeben. Bei anderen Fehlern wird `-1` zurückgegeben.

## Aufgabe 2: Testen des Ringpuffers

Zum Testen der Ringpuffers-Bibliothek soll ein Programm entwickelt werden, bei dem zwei Threads über einen Ringpuffer kommunizieren. Die Anwendung erstellt hierzu zunächt mit unserer Bibliothek einen Ringpuffer. Danach erzeugt sie mit `pthread_create()` jeweils einen Reader- und einen Writer-Thread und wartet dann mit `pthread_join()` auf deren Terminierung, um daraufhin den Ringpuffer wieder freizugeben.

Das Writer-Thread liest kontinuierlich Text von der Standardeingabe und schreibt diesen zeilenweise in den Ringpuffer. Zum zeilenweisen Einlesen von Text benutzt Ihr `getline()` und lasst diese Funktion den Speicher für die eingelesene Zeile allozieren. Natürlich müsst Ihr diesen auch mit `free()` wieder freigeben. Ist in dem Ringpuffer derzeit nicht genügend Platz zum Speichern der Zeile, wird das Schreiben in einer Schleife so lange versucht, bis es schließlich gelingt. Der Writer-Thread beendet sich, sobald beim Lesen von der Standardeingabe das Dateiende erreicht ist.

Das Reader-Thread liest Text aus dem Ringpuffer und gibt sie dann auf der Standardausgabe aus. Nach einem erfolgreichen Leseversuch versucht der Reader-Thread sofort wieder, einen neuen Puffereintrag zu lesen. Ist der Puffer leer, wird eine Sekunde mittels `sleep()` gewartet, bevor der nächste Versuch erfolgt. Wenn der Ringpuffer keine Daten mehr enthält und sich der Writer-Thread bereits beendet hat, beendet sich auch der Reader-Thread.

Testet die Anwendung durch die Eingabe einer Textdatei über `stdin`, und lenkt die Standardausgabe `stdout` in eine Ausgabedatei um. Prüft das Ergebnis durch Vergleich der Ein- und Ausgabedatei. Dokumentiert Eure Ergebnisse.

### Hinweise

Die Abgabe erfolgt als Ausdruck am Ende der Vorlesung und zusätzlich elektronisch über das Subversion Repository. Die Dokumentation der Aufgabenlösung ist in LaTeX anzufertigen. Bitte vergesst nicht die Namen aller Gruppenmitglieder mitanzugeben.