Übungen zur Vorlesung
Betriebssysteme I
Studiengang Informatik
WS 09/10

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

Ausgabe: 15.11.2009
Abgabe: 26.11.2009

Übungszettel 2

Aufgabe 1: Variabler Ringpuffer im Shared Memory

In der Vorlesung wurde Euch das Konzept des Ringpuffers vorgestellt. Dieses ermöglicht einem Prozess von ihm produzierte Daten in den Ringpuffer 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 programmiert werden, die Ringpuffer im Shared Memory bereitstellt und Datenpakete variabler Länge unterstützt. Wie in der Vorlesung beschrieben, sollen in dem zu implementierenden Ringpuffer nicht die Datenpakete selbst, sondern lediglich Verweise auf diese gespeichert werden. Damit die Datenpakete für alle an der Kommunikation teilnehmenden Prozesse zugreifbar sind, werden diese in einem Datenbereich mit in dem Shared Memory gespeichert.

In dem Ringpuffer selbst sollen nun Elemente des Typs

```
typedef struct SHMRBElement {
  size_t       length;
  unsigned int dataIdx;
} SHMRBElement_t;
```

gespeichert werden. `length` ist dabei variable Länge eines Datenpakets und `dataIdx` dessen Startindex innerhalb des Datenbereichs.

Zusätzlich definieren wir uns

```
#define SHMRB_BUFFER_SIZE 8
#define SHMRB_DATA_SIZE 1024
```

wobei `SHMRB_BUFFER_SIZE` die Anzahl der Elemente in dem Ringpuffer und `SHMRB_DATA_SIZE` die Größe des Datenbereichs festlegt.

Zur Verwaltung eines Ringpuffers mitsamt seiner Nutzdaten wird die Struktur `SHMRBBuffer` im Shared Memory gespeichert:

```
typedef struct SHMRBBuffer {
  SHMRBElement_t buffer[SHMRB_BUFFER_SIZE];
  char           data  [SHMRB_DATA_SIZE];
  unsigned int   ri;
  unsigned int   wi;
  unsigned int   dri;
  unsigned int   dwi;
} SHMRBBuffer_t;
typedef SHMRBBuffer_t * SHMRBHandle_t;
```

`data` ist der Datenbereich und `buffer` stellt den eigentlichen Ringpuffer dar. Des weiteren enthält die Strukur noch Lese- und Schreibindizes auf den Ringpuffer und auf den Datenbereich.

Der Datentyp `SHMRBHandle_t` dient zur Referenzierung eines Ringpuffers bei der Verwendung der Bibliotheksfunktionen, die nun im Folgenden beschrieben werden.

Erstellen des Ringpuffers

Mit der Funktion `shmrb_create()` wird ein neuer Ringpuffer in einem Shared Memory Bereich angelegt. `shm_key` ist der Shared Memory Key für `shmget()` und `handle` ist ein Ausgabeparameter, über den der Ringpuffer für nachfolgende Zugriffe referenziert werden kann.

Iim Fehlerfall wird `-1` und im Erfolgsfall wird `0` zurück gegeben.

```
int shmrb_create(key_t shm_key,
 SHMRBHandle_t *handle);
```

Anfordern des Ringpuffers

Mit `shmrb_get()` wird ein bereits existierender Ringpuffer in dem Shared Memory mit dem Key `key` in Form von `handle` zur Verfügung gestellt.

`rb_get()` gibt `-2` zurück, wenn ein Ringpuffer mit dem Key nicht existiert. Bei sonstigen Fehlern wird `-1` und im Erfolgsfall wird `0` zurück gegeben.

```
int shmrb_get(key_t shm_key,
      SHMRBHandle_t *handle);
```

Schreiben in den Ringpuffers

`shmrb_write()` schreibt ein Datenpaket `item` der Länge `len` in den Ringpuffer referenziert durch `handle`.

Ist das zu schreibende Item größer als der Datenbereich, schlägt der Aufruf mit `-2` fehl. Sollte in dem Ringpuffer oder dem Datenbereich zur Zeit des Aufrufs nicht genügend Platz sein, wird `-3` zurück gegeben. Ist am Ende des Datenbereichs nicht mehr genügend Platz sein, um das gesamte Datenpaket am Stück zu speichern, schlägt der Aufruf so lange fehl, bis am Anfang des Datenbereichs ausreichend Platz ist.

Bei sonstigen Fehlern wird `-1` und im Erfolgsfall wird `0` zurück gegeben.

```
int shmrb_write(SHMRBHandle_t handle,
const char *item,
size_t len);
```

Lesen von dem Ringpuffers

`shmrb_read()` liest vom Ringpuffer referenziert durch `handle` ein Datenpaket `item` mit der Maximallänge `*len`. Die tatsächliche Größe der Nachricht ist nach dem Aufruf in `*len` hinterlegt.

`rb_read()` gibt `-2`, wenn das Datenpaket nicht in `item` passt und `-3`, wenn der Puffer gerade leer ist, zurück. Bei sonstigen Fehlern wird `-1` und im Erfolgsfall wird `0` zurück gegeben.

```
int shmrb_read(SHMRBHandle_t handle,
       char *item,
       size_t *len);
```

Aufgabe 2: Testen des Ringpuffers

Zum Testen des Ringpuffers sollen zwei Programme implementiert werden, die über einen Ringpuffer im Shared Memory Daten austauschen:

- Das erste Programm hat die Rolle des Producers und liest kontinuierlich Text von der Standardeingabe und schreibt diesen dann zeilenweise in den Ringpuffer. Ist in dem Puffer derzeit nicht genügend Platz, wird das Schreiben in einer Schleife so lange versucht, bis es schließlich gelingt. Ist die gelesene Zeile jedoch länger, als die Größe des Datenbereichs, wird die Zeile verworfen. Der Producer Prozess hat die Aufgabe den Ringpuffer anzulegen. Er beendet sich sobald beim Lesen von der Standardeingabe das Dateiende erreicht ist.

- Das zweite Programm hat die Rolle des Consumers und liest diesen Text aus dem Ringpuffer und gibt sie dann auf der Standardausgabe aus.

Testet die Programme mit einer Auswahl von Texten und weist nach, dass die Programme korrekt zu funktionieren scheinen.

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.