Aufgabe 10:

02.07.2007

jbuffer (12 Punkte) Bearbeitung in Zweier-Gruppen

Programmieren Sie ein Modul **`jbuffer`** (**j**itter **buffer**), das einen Datenstrom von einem Filedeskriptor einliest, die Daten puffert und anschließend auf einem anderen Filedeskriptor zeitversetzt wieder ausgibt. Solche Mechanismen werden beispielsweise eingesetzt, um Schwankungen in der Ankunftsrate von Audiodaten (Jitter) auszugleichen und eine gleichmäßige Wiedergabe zu ermöglichen.

a) Ringpuffer

Ihr Modul soll eine Funktion mit folgender Schnittstelle bereitstellen:

```
void jbuffer(int fd1, int fd2, int bufsize, int bufdelay);
```

Die Funktion sorgt dafür, dass ein Bytestrom von Filedeskriptor **`fd1`** in einen Puffer der Größe **`bufsize`** eingelesen wird und die Daten auf dem Filedeskriptor **`fd2`** ausgegeben werden, sobald der Puffer erstmals den Füllstand **`bufdelay`** erreicht hat. Die Funktion **`jbuffer()`** soll nach der Initialisierung zwei Threads (**pthread_create(3)**) erzeugen, welche die Aufgabe nebenläufig erledigen.

Der erste Thread liest die Daten zeichenweise von **`fd1`** und schreibt sie in den Puffer. Der zweite Thread wartet zunächst bis der Füllstand **`bufdelay`** erreicht ist und beginnt dann mit der Ausgabe auf **`fd2`**. Der Puffer soll als Ringpuffer betrieben werden. Synchronisieren Sie den Puffer mit Hilfe von Semaphoren (siehe Teilaufgabe b). Zur Synchronisation der beiden Threads beim Puffer-Zugriff (Anfangs-Synchronisation, voller Puffer, leerer Puffer, Endsituation) eignen sich am besten zählende Semaphoren, für einen einfachen gegenseitigen Ausschluss auf gemeinsame Daten reichen ggf. auch Mutex-Variablen.

Thread 1 terminiert nachdem auf **`fd1`** End-of-File erreicht wurde. Thread 2 gibt danach noch alle im Puffer befindlichen Daten aus und terminiert anschließend ebenfalls. Die Funktion **`jbuffer()`** gibt nach dem Ende beider Threads (**pthread_join(3)**) alle belegten Ressourcen frei und kehrt zurück.

b) Semaphoren (vgl. auch Vorl., Folien C 11-29 ff.)

Programmieren Sie selbst auf der Basis von Mutex- (**pthread_mutex_init(3)**) und Condition-Variablen (**pthread_cond_init(3)**) zählende Semaphoren. Die Schnittstelle des Moduls und passende Funktionsrümpfe befinden sich in */proj/i4sos/pub/aufgabe10*. Kopieren Sie die Dateien (*i4sem.c, i4sem.h*) in Ihr Arbeitsverzeichnis und implementieren Sie die Funktionen.

c) Makefile

Erstellen Sie ein zu der Aufgabe passendes Makefile, welches die Objekt-Dateien (jbuffer.o, etc.) nur bei Bedarf übersetzt und das Testprogramm *jbuftest* aus */proj/i4sos/pub/aufgabe10/jbuftest.c* erzeugt.

d) Dokumentation

Beschreiben Sie in einer Datei **jbuffer.txt** welche Koordinierungsprobleme in der Aufgabe auftreten und wie Sie diese jeweils mit welchen Hilfsmitteln synchronisieren.

Hinweise:

- Das Programm *jbuftest* liest Daten von der Datei **`/proj/i4sos/pub/aufgabe10/input`** und schreibt auf die Datei **`./output`**. Überprüfen Sie mit **diff(1)**, dass die beiden Dateien identisch sind.
- Die pthread-Funktionen sind in einer speziallen Funktionsbibliothek (libpthread) zusammengefasst, die Sie beim Kompilieren bzw. Binden Ihres Programms mit angeben müssen (Option **`-lpthread`**).
- Im Verzeichnis */proj/i4sos/pub/aufgabe10/* finden Sie eine Datei *unbufio.c,* die Funktionen zum ungepufferten Lesen und Schreiben von Zeichen enthält und von Ihnen ebenfalls verwendet werden darf.

Abgabe: bis spätestens Donnerstag, 12.07.2007, 12:00 Uhr