Institut für Informatik
Arbeitsgruppe Software Engineering
Barry Linnert

Betriebssysteme WiSe 2016/17· Übungsblatt 4
Bearbeitungszeit: 08.12.2016 – 05.01.2017 um 09:59 Uhr

Bereiten Sie Ihre **Lösungen** grundsätzlich so vor, dass Sie diese in der Übung Ihren Kommilitonen in geeigneter Form **zeigen** und **diskutieren** können. Geben Sie bitte stets Ihre verwendeten **Quellen** an.

Die Abgabe erfolgt elektronisch über das KVV.

Aufgabe 4-1 (Nachrichten unterschiedlicher Länge) – 5 Punkte

Im Allgemeinen sind an Kommunikationsobjekten Nachrichten unterschiedlicher Länge zugelassen.

1. Welche Auswirkungen hat dies auf die Daten des Kommunikationsobjekts? Bei welchen Varianten der Nachrichtenübergabe hat dies weitere Auswirkungen, und bei welchen hat dies sonst keine weiteren Auswirkungen?

2. Welche Probleme können aus der Sicht der Kommunikationspartner auftreten, wenn Nachrichten unterschiedlicher Länge zugelassen sind? Schlagen Sie geeignete Lösungen vor!

Aufgabe 4-2 (Kontextwechsel und präemptives Multitasking) – 15 Punkte

In dieser Aufgabe soll Ihr Betriebssystem um die Fähigkeit des präemptiven Multitaskings erweitert werden, um so mehrere Threads quasi-parallel ausführen zu können. Der System-Timer dient dabei als Zeitgeber und gibt das Intervall vor, in dem der Kern den gerade ausgeführten Thread automatisch wechselt.

Um dies zu erreichen, sind folgende Teile zu erledigen (nicht unbedingt in dieser Reihenfolge):

1. Definieren und implementieren Sie eine Thread-Verwaltung, die ein Minimum von 16 Threads unterstützt. Insbesondere soll es eine Funktion geben, die einen neuen Thread startet.
2. Entscheiden Sie sich für eine Variante, einen Kontextwechsel durchzuführen, und setzen Sie diese an geeigneter Stelle in Ihrem Code um.
3. Implementieren Sie einen Scheduler, der regelmäßig einen Kontextwechsel anstößt. Es soll kein Thread verhungern und die verfügbare CPU-Zeit gleichmäßig zwischen den Threads aufgeteilt werden (z. B. eine Round-Robin-Variante). Machen Sie dabei die Zeitscheibenlänge (im Quellcode) konfigurierbar.
4. Überlegen Sie sich dabei wie Sie einen Thread sinnvoll beenden, und wie Sie mit dem Fall umgehen, dass es keine Threads zum Ausführen gibt.

Zur Demonstration erzeugen wir in dieser Aufgabe gänzlich unprofessionell aus der DBGU-Interrupt-Behandlung heraus neue Threads:

5. Erzeugen Sie nach dem Empfang eines Zeichens noch innerhalb der Interrupt-Behandlung einen neuen Thread. (Entweder geben Sie dem Thread das Zeichen gleich als Parameter mit, oder Sie puffern das Zeichen wie gehabt und der Thread holt sich das Zeichen als erste Handlung ab.)

6. Jeder erzeugte Thread soll das empfangende Zeichen wiederholt (aber nicht endlos) mit kleinen Pausen ausgeben und sich anschließend beenden.
7. Geben Sie bei jedem Timer-Interrupt ein Ausrufezeichen aus, und bei jedem Kontextwechsel einen Zeilenumbruch.

Damit ergibt sich eine Ausgabe, die der folgenden ähneln sollte (je nach Tastendrücken, Zeitscheibenlänge, Anzahl der Wiederholungen und Länge der Pausen):

```
!!!
AA!AAA!AAA!AA
!!!!!!!
BBB!BBB!
CCC!
BBB!
CCC!
B
CC!CC
!!!!!!
D!DDD!
EEE!
FFF!
DDD!
EEE!
FFF!
DDD!
EEE!
FFF!
E
F
```

Bei kurzen Zeitscheiben und langen Pausen gibt es Leerzeilen nur mit Ausrufezeichen. Bei kurzen Pausen und langen Zeitscheiben wird ein Thread nie unterbrochen, bevor er fertig ist.

Zusatzaufgabe (Dynamischer Timer) – 4 Punkte

Implementieren Sie:

- keine zu kurzen Zeitscheiben
- keine überflüssigen Timer-Interrupts
- gezieltes Aufwachen (nächste Aufgabe muss u. a. sleep() implementiert werden)
- Stromsparmodus bei Idle