Institut für Informatik
Arbeitsgruppe Software Engineering
Barry Linnert

Betriebssysteme WiSe 2016/17· Übungsblatt 7
Bearbeitungszeit: 02.02.2017 – 16.02.2017 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 7-1 (Auswahlstrategien) – 5 Punkte

Auf einem PC läuft eine Anwendung zur Wiedergabe von Musik. Die Musikdateien liegen auf der lokalen Festplatte. Die Wiedergabe wird in unregelmäßigen Momenten unterbrochen, sodass der Musikgenuss sehr eingeschränkt ist. Wie gehen Sie zur Problembehebung vor? Begründen Sie kurz die einzelnen Schritte.

Aufgabe 7-2 (Prozesse) – 15 Punkte

Bisher teilen sich alle Threads einen (logischen) Adressraum und können sich dadurch gegenseitig den Speicher auslesen oder überschreiben. Diese Art, Mehrprogrammbetrieb zu realisieren, ist somit nur sinnvoll, wenn die Anwendungen sich gegenseitig vertrauen und im Wesentlichen fehlerfrei sind. Im Allgemeinen ist diese Annahme jedoch nicht berechtigt.

In dieser Aufgabe geht es nun um den Schutz von Anwendungen voreinander. Dies wird dadurch erreicht, dass jede Anwendung einen eigenen (logischen) Adressraum bekommt, sodass sie nicht länger auf Speicher einer anderen Anwendung zugreifen kann. Beachten Sie, dass einem Adressraum prinzipiell immer noch mehrere Threads zugeordnet werden können.

Derzeit sollte sich der durch eine Anwendung tatsächlich änderbare Speicherbereich auf den Bereich mit den User-Stacks beschränken. In dieser Aufgabe soll ein Datenbereich hinzugefügt und mehrere Adressräume eingeführt werden. In jedem Adressraum sollen sich der Datenbereich und der Stack-Bereich jeweils an der gleichen (logischen) Stelle befinden, jedoch auf unterschiedlichen physischen Speicher verweisen.

Also:

1. Führen Sie das Konzept von Adressräumen ein und schalten Sie Adressräume beim Kontextwechsel mit um. Achten Sie dabei auf korrektes Handling von Cache und TLB.
2. Erweitern Sie den Aufruf zum Erzeugen des Systemruf-Threads dahingehend, dass der neue Thread entweder im Adressraum des aufrufenden Threads oder in einem neuen Adressraum erzeugt wird – je nach Parametern. (Alternativ können dafür auch zwei Systemrufe realisiert werden.)
3. Unterstützt werden sollen wenigstens acht Adressräume/Prozesse. Die Adressräume sollen sich zumindest – so wie oben beschrieben – im Mapping von User-Daten und User-Stack unterscheiden.

Zur Demonstration:

4. Der initiale Prozess wartet wieder auf Benutzereingaben. Für jeden Tastendruck wird ein Prozess (Thread in neuem Adressraum) gestartet, dem das entsprechende Zeichen übergeben wird.
5. Der Prozess legt das Zeichen im Datenbereich ab, initialisiert einen Zähler im Datenbereich mit Null und startet anschließend zwei weitere Threads im eigenen Adressraum.
6. Anschließend gehen alle drei Threads in eine Schleife: Solange Zähler kleiner Limit, erhöhe Zähler, mache Meldung (siehe unten), lege dich einen Moment schlafen.

Die Meldung der Threads soll Aufschluss über die gedrückte Taste (stellvertretend für den Prozess), den Thread innerhalb des Prozesses (entweder als lokale oder globale ID) sowie den Zähler geben.

Wer möchte kann noch für jeden Thread mitzählen wie viele Ausgaben jeder Thread macht, um die Fairness zu prüfen. Auch darf wieder mit aktivem Warten (Großbuchstaben) und passivem Warten (alle anderen Zeichen) experimentiert werden.

Das könnte dann in etwa folgendermaßen aussehen (<taste><id>: <glob.zähler> (<lok.zähler>)):

a1: 1 (1)		A1: 1 (1)		a1: 1 (1)	...
a3: 2 (1)		A1: 2 (2)		a3: 2 (1)	a3: b (4)
a2: 3 (1)		A2: 3 (1)		a2: 3 (1)	a2: c (4)
a1: 4 (2)		A2: 4 (2)		a1: 4 (2)	b1: 7 (3)
a3: 5 (2)		A3: 5 (1)		a3: 5 (2)	b2: 8 (3)
a2: 6 (2)		A3: 6 (2)		a2: 6 (2)	b3: 9 (3)
a1: 7 (3)		A1: 7 (3)		b1: 1 (1)	a1: d (5)
a3: 8 (3)		A1: 8 (4)		b2: 2 (1)	a3: e (5)
a2: 9 (3)	oder	A2: 9 (3)	oder	b3: 3 (1)	a2: f (5)
a1: a (4)		A2: a (4)		a1: 7 (3)	b1: a (4)
a3: b (4)		A3: b (3)		a3: 8 (3)	b2: b (4)
a2: c (4)		A3: c (4)		a2: 9 (3)	b3: c (4)
a1: d (5)		A1: d (5)		b1: 4 (2)	a1: 10 (6)
a3: e (5)		A1: e (6)		b2: 5 (2)	b1: d (5)
a2: f (5)		A2: f (5)		b3: 6 (2)	b2: e (5)
a1: 10 (6)		A2: 10 (6)		a1: a (4)	b3: f (5)
				...	b1: 10 (6)

Hinweise:

- So wie die Anwendung beschrieben ist, gibt es eine Race-Condition bei Erhöhung des Zählers. Diese darf ignoriert werden.
- Deklarieren Sie die Daten im Datenbereich entweder als `volatile` oder bauen Sie an geeigneten Stellen *compiler optimization barriers* ein, um sicherzustellen, dass die Threads bei jedem Schleifendurchlauf an den Speicher gehen. (Vergisst man das, ist ein mögliches Symptom, dass jeder Thread eines Prozesses für sich zählt.)
- Der Datenbereich kann auf eine von drei Arten (mit steigendem Schwierigkeitsgrad) realisiert werden: als statisches Mapping irgendwo im logischen Adressraum und Zeigern darauf, als tatsächliches Datensegment der Anwendung und globalen Variablen, oder als dynamisch angeforderter Heap.
- Sollten Cache oder TLB falsch behandelt werden, könnten (neben dem Auftreten merkwürdiger Abstürze) Threads verschiedener Prozesse gemeinsam zählen oder die Taste des zuerst gestarteten Prozesses verloren gehen.
- Das Abschalten des Caches ist eine Art, den Cache im Rahmen der Aufgabe korrekt zu behandeln.
- Falls es jemanden stört, dass alle Threads eines Prozesses quasi gleichzeitig aufwachen, so kann auch mit verschieden langen Schlafphasen oder einem versetzen Start der einzelnen Threads gearbeitet werden, solange sich alle Threads/Prozesse irgendwie abwechseln. (Das Ziel dieser Demonstration ist es, die unterschiedlichen Gültigkeitsbereiche von Speicher/Variablen aufzuzeigen: systemweit, innerhalb eines Prozesses, innerhalb eines Threads.)