Übungen zur Vorlesung
Betriebssysteme II
Sommersemester 2012

AG BS
Jan Peleska
Florian Lapschies

Blatt 1

Integration neuer Systemaufrufe in den Linux-Kernel

In dieser Übungsserie wird die Entwicklung neuer Systemaufrufe, ihre Integration in den Linux-Kernel, sowie die zugehörige Kernelgenerierung geübt.
Dabei handelt es sich um vier Funktionen zur **optimistischen Transaktionsverwaltung**, welche den konkurrierenden Zugriff auf gemeinsam benutzten Speicher unterstützen. Die Aufrufe lauten

```
int bs2_ta_init();
ssize_t bs2_ta_read(void *buf, size_t count);
ssize_t bs2_ta_write(void *buf, size_t count);
ssize_t bs2_ta_unconditional_write(void *buf, size_t count);
```

und haben folgende Bedeutung: Im Kernel wird ein Puffer (unsigned char array) fester Größe angelegt, in welchem die Nutzdaten der Transaktion gespeichert werden.
Mit dem Aufruf `bs2_ta_init()` wird dieser Bereich mit 0 initialisiert.
Mit `bs2_ta_unconditional_write()` werden die ersten `count` Bytes des Puffers überschrieben. Ist der Puffer kleiner als `count` Bytes, wird nur bis zur Pufferlänge geschrieben. Die tatsächlich geschriebene Länge wird im Rückgabeparameter gemeldet. Bei jedem Schreiben wird ein interner Transaktionszähler im Kernel erhöht.
Mit `bs2_ta_read(void *buf, size_t count);` werden die ersten `count` Bytes bzw. der gesamte Puffer gelesen. Die tatsächlich gelesene Länge wird im Rückgabeparameter gemeldet.
Mit `bs2_ta_write()` werden die ersten `count` Bytes bzw. der gesamte Puffers überschrieben, **wenn nach dem letzten Read des Schreibers zwischenzeitlich kein Write durch einen anderen Prozess erfolgt – d. h. der Transaktionszähler nicht erhöht worden – ist**. Andernfalls kehrt der Befehl mit `EAGAIN` zurueck, und der Schreiber muss die Daten zunächst neu lesen.

Aufgabe 1: Interpretation des Verfahrens (10%)

Begründen Sie, warum dieses Verfahren

- eine Form der konsistenten Transaktionsverwaltung darstellt,
- als “optimistisch” bezeichnet werden kann.

Aufgabe 2: Entwicklung der Systemaufrufe (60%)

Erweitern sie den Linux-Kernel 3.3.2 um die o.g. Systemcalls analog zum Beispiel aus dem Tutorium. Implementieren Sie den Kernelcode für die Systemcalls in einer neuen Datei `bs2_transactions.c` im Verzeichnis `linux-3.3.2/kernel`.
Hinweise: Die PID des aufrufenden Prozesses kann über `current` ermittelt werden. `current` ist ein Pointer auf den Prozesstabelleneintrag (Typ `struct task_struct`) des aktuellen Prozesses. Die PID ist in `current->pid` gespeichert.
Sie dürfen voraussetzen, dass nicht mehr als 100 Prozesse an der Transaktionsverwaltung teilnehmen.
Wählen Sie eine Puffergröße von 256 Bytes.
Kommentieren Sie sorgfältig die kritischen Abschnitte der einzelnen Kernelfuntionen. Schützen Sie diese Sektionen durch den Aufruf der Funktion

```
spin_lock(spinlock_t *)
```

Das Ende der Critical Section wird mit

```
spin_unlock(spinlock_t *)
```

markiert. Der erste Aufruf bewirkt, dass die Verarbeitung des Systemaufrufs nicht nach einem Re-Scheduling durch den gleichen oder einen in Konflikt stehenden Systemaufruf eines anderen Prozesses "überholt" werden kann.

Aufgabe 3: Testprogramme (30%)

Um zu testen, dass Ihre Kernelerweiterung vernünftig funktioniert, sollen folgende Userspace-Programme geschrieben werden:

1. Schreiben Sie ein Initialisierungsprogramm, welches den Transaktionspuffer mit 0,...,255 fortlaufend vorbelegt.

2. Schreiben Sie ein in mehrfacher Kopie ausführbares Programm, welches folgendes Transaktionsverhalten implementiert:

   - Der ganze Puffer wird gelesen.
   - Im Puffer wird der erste Wert ungleich Null gesucht.
   - Der Puffer wird im Userspace an dieser Stelle mit Null überschrieben und dann zurück in den Kernel geschrieben.
   - Wenn das Write gelingt, wird der "von der Applikation verbrauchte" Wert in eine Datei geschrieben. Wenn das Write nicht gelingt, erfolgt keine Ausgabe.
   - Nach dem Write wird wieder mit dem Lesen des Puffers fortgefahren.
   - Wenn alle Werte im Puffer Null sind, terminiert das Programm.

   Wenn Ihre Implementierung korrekt ist, kann auch bei paralleler Ausführung mehrerer Programmkopien jeder Wert genau einmal in der Ausgabedatei vorkommen.

   Zur Generierung der Systemcalls-Wrapper zum Einsprung in den Kernel mit Interrupt `0x80` kann das Makro `INLINE_SYSCALL` aus der beiliegenden Datei `sysdep.h` genutzt werden. Hierbei handelt es sich um eine vereinfachte Datei aus der GNU C Library 2.9.

Abgabe: Bis Montag, 07.05.2012, in der Übung.

Sowohl bei der schriftlichen Lösung als auch im Source-Code die Namen aller Gruppenmitglieder nicht vergessen!