Übungen zur Vorlesung
Betriebssysteme II
Sommersemester 2004

AG BS
Jan Peleska, Elmar Prüße

Blatt 1

Entwicklung einer neuen Scheduling Klasse im linux-Kernel

Führen Sie in der Linux-Datei `sched.c` eine neue Scheduling Klasse `SCHED_BS2` in den Schritten ein, welche durch die folgenden Aufgaben definiert sind. Als Basis verwenden Sie bitte ein Scheduler-Version, wie sie beispielsweise in den 2.4.18 (oder älteren, es geht wohl auch 2.4.20) Kernen vorhanden war: Wir benötigen die Verwendung/Modifikation der Funktion `goodness()`. Diese gibt es nicht mehr beim neuen O(1)-Scheduler. Daher sind die neuen Versionen von `sched.c` nicht für die folgenden Aufgaben geeignet.
SMP-bezogene Änderungen im Scheduler sind nicht erforderlich.

Aufgabe 1: Erweiterung des System Calls `sched_setscheduler(2)`

Erweitern Sie den Code von `setscheduler()` in `sched.c` so, dass eine neue Scheduling Klasse `SCHED_BS2` (= 4) akzeptiert und in die Prozesstabelle (`task_t`, Feld `policy`) zum aufrufenden/betroffenen Prozess eingetragen wird. Zu dieser neuen Klasse gibt es keine Benutzerdefinierbaren Prioritäten (also keine Änderungen in den Systemaufrufen, die sich auf die Priorität beziehen).

Aufgabe 2: Erweiterung der Funktionen `goodness()` und `schedule()` in `sched.c`

Erweitern Sie den Code der Funktionen `goodness()` und `schedule()`, so dass die neue Scheduling Policy folgende Wirkung hat:

(a) Rechenbereite `SCHED_BS2`-Prozesse werden immer gegenüber `SCHED_OTHER` bevorzugt.

(b) `SCHED_RR`, `SCHED_FIFO`-Prozesse werden immer gegenüber `SCHED_BS2`-Prozesse bevorzugt (Implementierung in `goodness()`).

(c) `SCHED_BS2`-Prozesse bekommen in iherem Prozesstabelleneintrag einen nice-Wert im Bereich 500-699.

(d) Wird ein `SCHED_BS2`-Prozess selektiert, erhält er einen zufälligen neuen nice-Wert im Bereich 500..599 (dies wird in `schedule()` implementiert).

(e) Wird ein rechenbereiter `SCHED_BS2`-Prozess NICHT selektiert, wird sein nice-Wert um 1 inkrementiert, aber niemals höher als 699 (Implementierung in `goodness()`). Wovon hängt es ab, dass dieses Verfahren fair ist?

Aufgabe 3: Neuer Systemaufruf `getSchedInfo()`

Implementieren Sie den Systemaufruf
`int getSchedInfo(pid_t pid)`
der zu gegebener Prozess-Id zurückgibt, wie häufig dieser Prozess seit seinem Start die CPU bekommen hat. Erweitern Sie hierzu sched.c um eine globale Datenstruktur, in welcher der Scheduler die Aufruf-Häufigkeiten speichert. Führen Sie `getSchedInfo()` als neuen System-Call mit einer noch nicht vergebenen Aufrufnummer ein. Integrieren Sie `sys_getSchedInfo()` in `sched.c`.

Aufgabe 4: Testprogramm

Schreiben Sie einen Prozess, der wahlweise (über Aufrufparameter gesteuert) als `SCHED_OTHER` oder `SCHED_BS2` Prozess läuft. Jeder Prozess vollzieht eine Schleife i:0..MAX, in welcher er kontinuierlich abfragt, wie oft er schon vom Scheduler selektiert wurde. Nach Terminierung der Schleife wird die Anzahl Selektionen ausgegeben. Testen Sie die Ergebnisse der Aufgaben (1)..(4), indem Sie 5 `SCHED_OTHER` und 5 `SCHED_BS2`-Prozesse parallel starten. Interpretieren Sie das Ergebnis im Vergleich der beiden Scheduling policies.

Abgabe: Bis Montag, 20. Mai 2004, in der Vorlesung.

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