Übungen zur Vorlesung
Betriebssysteme II
Sommersemester 2009

AG BS
Jan Peleska
Florian Lapschies

**Blatt 2**

# Entwicklung einer neuen Scheduling-Policy

In diesem Übungsblatt soll der Linux-Kernel um eine weitere Scheduling-Policy erweitert werden.
Der zu entwickelnde Scheduler orientiert sich konzeptuell an dem *Universell stark fairen Scheduler* wie ihr ihn aus Betriebssysteme I kennt. Damit dieser idealisierte Scheduler implementierbar wird, müssen wir verständlicherweise auf die Universalität verzichten und nennen den zu entwickelnden Scheduler daher *Almost Universally Fair Scheduler* oder kurz *AUFAIR*.
Das Schedulingverhalten von AUFAIR lässt sich wie folgt beschreiben:

- AUFAIR lässt unbeschränkt viele Prozesse zu.
- Jedem Prozess wird ein Prozessgewicht $z_i$ (=Prioritäten) im Bereich $z_i \in \{0 \dots AUFAIR\_NB\_WEIGHTS - 1\}$ zugeordnet. Wie beim $O(1)$-Scheduler ist 0 die beste Priorität.
- Initial und nach Inanspruchnahme der CPU erhält jeder Prozess ein zufälliges Prozessgewicht aus $z_i \in \{0 \dots AUFAIR\_NB\_WEIGHTS - 1\}$.
- Gibt es keinen rechenbereiten Prozess, wird auch keiner gescheduled.
- Hat kein rechenbereiter Prozess das Prozessgewicht 0, wird ein rechenbereiter Prozess gescheduled, der unter den rechenbereiten Prozessen das niedrigste Prozessgewicht hat. Das Prozessgewicht aller rechenbereiten Prozesse, die nicht selektiert wurden, wird um eins erniedrigt.
- Hat mindestens ein rechenbereiter Prozess das Prozessgewicht 0, wird der rechenbereite Prozess selektiert, der dieses Gewicht als erstes erhalten hat.
- Wenn ein rechnender Prozess die CPU nicht freiwillig abgibt, kann er sie bis zum Ablauf einer konstanten Zeitscheibe behalten. Danach wird zwangsweise ein anderer Prozess selektiert, wenn vorhanden.

## Aufgabe 1: Implementierung des Almost Universally Fair Scheduler (70%)

Implementiert eine neue Scheduling Policy die den oben beschriebenen Scheduling-Algorithmus realisiert.

Erstellt hierzu eine weitere Schedulingklasse mit dem Namen `struct sched_class aufair_sched_class` und definiert die nötigen Methoden. Eure Schedulingklasse soll sich in der Hierarchie der Schedulingklassen zwischen der Klasse `fair_sched_class` und der Klasse `idle_sched_class` einordnen. Wird ein Prozess einer höheren Scheduling-Policy (z.B.: `SCHED_OTHER`) rechenbereit, sollen eventuell laufende AUFAIR-Prozesse die CPU abgeben. Die maximale Zeitscheibe für einen AUFAIR-Prozess beträgt 100 Ticks.
Prozesse sollen mit dem Systemcall `sched_setscheduler()` in eure Scheduling-Policy wechseln können. Definiert hierzu eure Policy als `#define SCHED_AUFAIR 6`.
Ähnlich wie der $O(1)$-Scheduler verwaltet euer AUFAIR-Scheduler für jedes Prozessgewicht eine Liste von rechenbereiten Prozessen mit genau diesem Prozessgewicht. Verwendet hierzu den Datentyp `struct list_head`. Damit zu gegebenem Prozessgewicht die zugehörige Liste von Prozesse geliefert werden kann, soll ein Array mit der Größe `AUFAIR_NB_WEIGHTS` die Listenköpfen enthalten. Damit das Erniedrigen von Prozessgewichten in konstanter Zeit geschieht, sollen nicht nacheinander alle Prozesse von einer Liste in die vorherige eingehängt werden. Stattdessen soll der Scheduler einen Index auf den Arrayeintrag für die Priorität 0 verwalten. Beim Erniedrigen der Prozessgewichte wird dieser entsprechend versetzt. Nun soll auch das Finden eines Prozesses mit dem geringsten Prozessgewicht möglichst effizient sein. Verwaltet hierzu analog zum $O(1)$-Scheduler eine Bitliste, in der Bits genau dann gesetzt sind, wenn mit dem zugehörigen Prozessgewicht Prozesse vorhanden sind. Nutze Funktionen aus `linux/bitops.h` um das niedrigste Prozessgewicht zu finden. Definiere dabei `AUFAIR_NB_WEIGHTS` als `100`.
Pseudo-Zufallszahlen könnt ihr euch mit `random32()` erzeugen lassen.
Eure Kernelerweiterung soll über die Kernelkonfiguration (`make menuconfig` u.ä.) aktivierbar sein. Erweitert hierzu die Datei `init/Kconfig`, so dass wenn Euer Scheduler bei der Konfiguration ausgewählt wurde `CONFIG_SCHED_AUFAIR` definiert ist. Eure Erweiterungen soll nur dann mitkompiliert werden, wenn `CONFIG_SCHED_AUFAIR` gesetzt ist.

## Aufgabe 2: Tests (30%)

Schreibt Tests, die nachweisen, dass euer Scheduler in der Lage ist, keinen, einen, drei und 120 Prozesse zu verwalten, ohne dabei die Fairness zu vernachlässigen. Ruft hierbei gezielt Systemcalls auf, die auch blockieren können. Testet, ob Prozesse, die sich in einer Endlosschreife befinden und ihre Zeitscheibe voll ausnutzen, die CPU schließlich enzogen wird. Überprüft weiterhin, ob Prozesse der Klasse `SCHED_OTHER` weiterhin gescheduled werden, wenn Prozesse der Klasse `AUFAIR` rechenbereit sind.

## Abgabe: Bis Dienstag, 26.05.2009, in der Vorlesung.

**Zur Abgabe gehört ein Diff eurer Kernelmodifikation, alle**

**geänderten und erstellten C-Dateien und eure Dokumentation der Lösung als PDF-Datei. Sowohl bei der schriftlichen Lösung als auch im Source-Code die Namen aller Gruppenmitglieder nicht vergessen!**