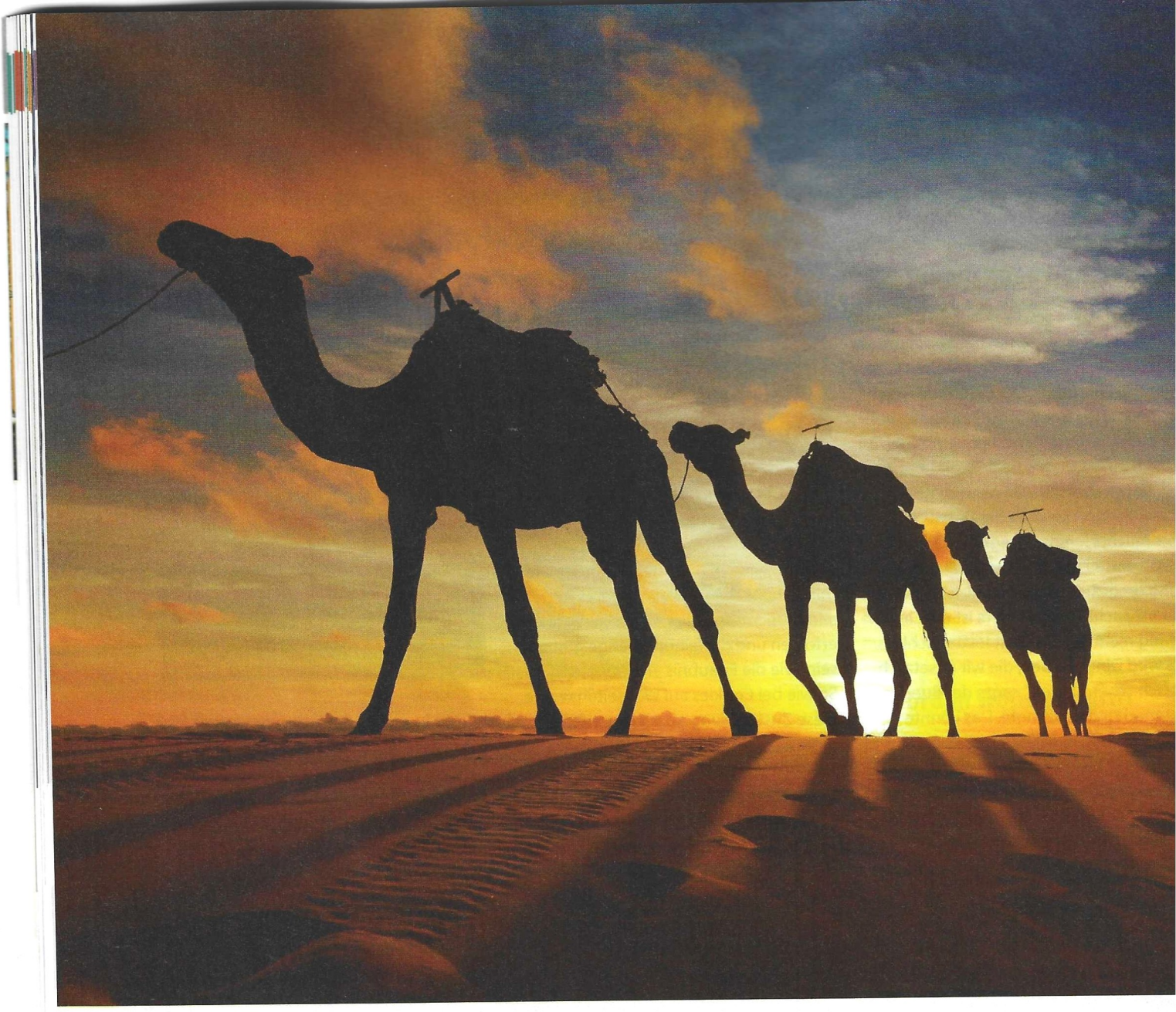


Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 125

Clevere Lastverteilung

Moderne Systemarchitekturen machen das Task-Scheduling immer komplexer. Für einen besonders sparsamen Betrieb sammelt der Kernel dazu Informationen über die Leistung der CPU-Kerne und über den Rechenzeithunger der Tasks.

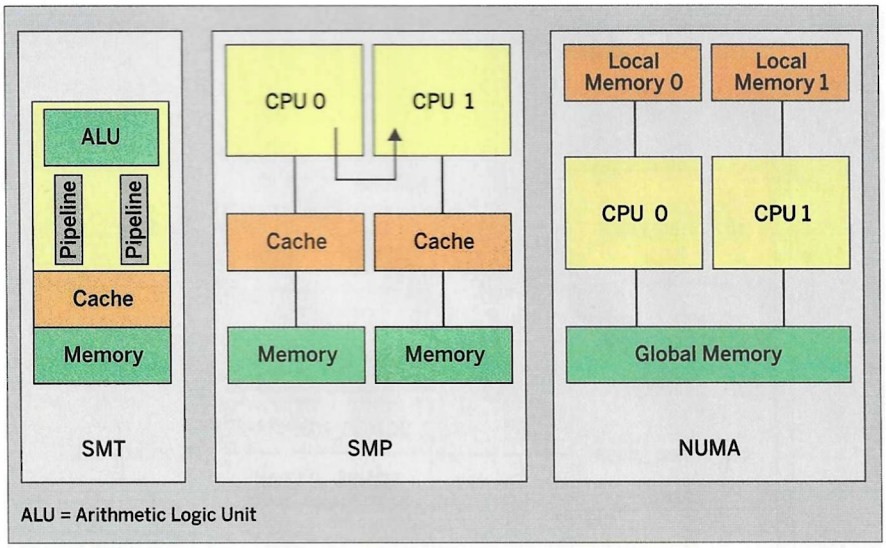
Eva-Katharina Kunst, Jürgen Quade

Ein Betriebssystem ist laut Definition Verwalter von Ressourcen. Es kümmer sich um das Verteilen von Speicher, un die Peripherie und um die Rechenzeit. Gerade Letzteres, also die Verteilung d Ressource CPU, zählt zu den wichtigst Aufgaben. Im Rahmen des Prozessma nagements ist dafür der Task-Schedul zuständig. Früher war er recht übersic lich gestaltet: Er musste lediglich die R sourcen eines einzigen Prozessorkern möglichst gerecht unter allen Rechen prozessen aufteilen. Jeder Prozess erh – unter der Randbedingung, einen ho hen Durchsatz bei überschaubarem A

wand zu gewährleisten – dieselbe Rechenzeit. Die Prozesse konnten freundlich sein und auf Unix-Maschinen per Kommando `nice` von sich aus einen Teil von der eigenen Rechenzeit an weniger freundliche Tasks abtreten.

Heute sieht die Prozessorwelt erheblich komplizierter aus. Aus Singlecore-Rechnern haben sich Multicore-Maschinen entwickelt, dem klassischen Singlecore-Scheduler gesellt sich der Multicore-Scheduler hinzu. Neben der quasi parallelen Verarbeitung gibt es dank Multicore-Hardware eine real parallele Verarbeitung. Zudem entstanden neue Anforderungen: Es geht nicht mehr nur um den Durchsatz, sondern auch um das Einhalten von Zeitbedingungen und um den sparsamen Umgang mit Energie.

Als wäre es noch nicht genug, solch unterschiedliche Anforderungen unter einen Hut zu bekommen, erschweren die unterschiedlichen Hardwarearchitekturen das Optimierungsproblem erheblich. Neben den klassischen SMP-Systemen, bei denen alle Prozessorkerne (Cores) eines Multicore-Systems identisch sind, gibt es heute Hyperthreading (SMT) sowie NUMA- und LITTLE.big-Architekturen (siehe Tabelle Multicore-Architekturen).



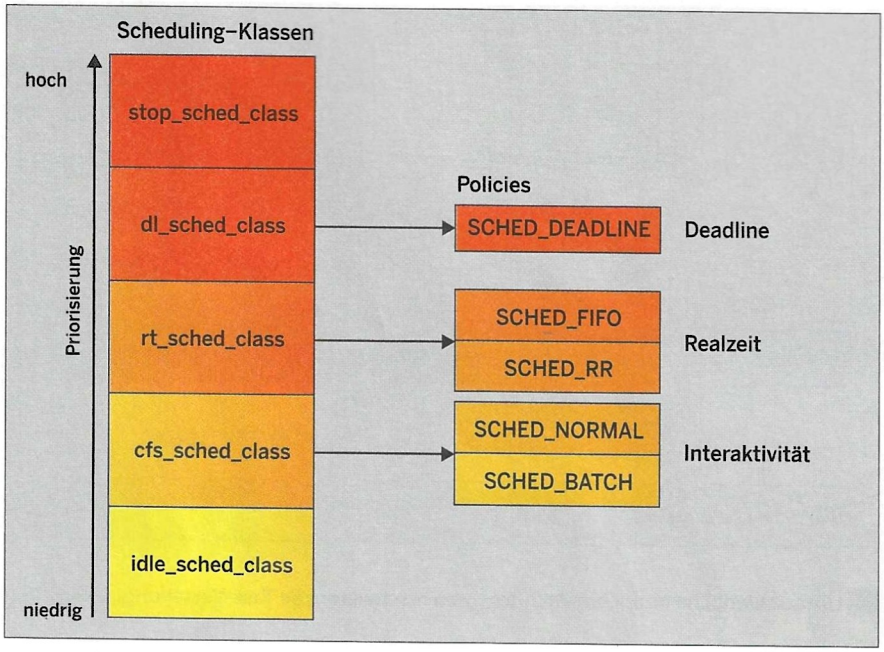
1 Unterschiedliche Multicore-Architekturen erschweren die Task-Verteilung.

In der Realität kommen diese Architekturen kunterbunt gemischt vor. So ist es völlig normal, dass ein NUMA-System mehrere SMP-Prozessoren einsetzt, die wiederum allesamt SMT unterstützen.

Die Krönung allerdings sind moderne Prozessoren, deren Verarbeitungsleistung von der Anwendung abhängt. Zu ihnen zählen beispielsweise CPUs, die besonders gut KI-Aufgaben lösen oder Floating Point beherrschen, aber gegenüber ihren herkömmlichen Schwestern und Brüdern ins Hintertreffen geraten, wenn es um normale Berechnungen respektive Datendurchsatz geht. Das alles verdeutlicht: Betriebssysteme haben bei Verwendung moderner Hardware ein erhebliches Optimierungsproblem zu bewältigen. Da

Multicore-Architekturen

Name	Beschreibung
Symmetric Hyperthreading (SMT)	Hat eine CPU zwei Befehls-Pipelines zur eigentlichen Verarbeitungskomponente (ALU), spricht man von Hyperthreading 1. Aus Sicht des Kernels handelt es sich um zwei Cores, abgesehen von der Befehlsdekodierung gibt es jedoch keine tatsächliche Parallelarbeit. Die Task-Migration kostet hier fast nichts, allerdings bleibt auch der Effekt enttäuschend niedrig.
Symmetric Multiprocessing (SMP)	Bei diesem Klassiker unter den Multicore-Architekturen sind die Prozessorkerne identisch aufgebaut und weisen damit auch dieselben Leistungsdaten auf. Bei der Migration eines Tasks von einem SMP-Kern zum anderen gilt es insbesondere, die CPU-Caches zu leeren. Dafür kann im besten Fall der Task mit voller Geschwindigkeit abgearbeitet werden.
Non-Uniform Memory Architecture (NUMA)	SMP- und SMT-Kerne teilen sich einen gemeinsamen Speicher. Da darauf immer nur eine CPU zugreifen kann, wird dieser Speicher bei zunehmender Anzahl der Kerne schnell zum Flaschenhals. Bei NUMA-Systemen haben die einzelnen Prozessoren jeweils einen eigenen lokalen Speicher, auf den sie mit voller Geschwindigkeit zugreifen können. Die Kosten für eine Task-Migration sind bei dieser Architektur relativ hoch, weil es nicht nur Caches und Pipelines zu leeren gilt, sondern auch Speicherinhalte transferiert werden müssen. Dafür steht dem migrierten Task aber auch ein System zur Verfügung, bei dem er keine weiteren Ressourcen teilen muss.
LITTLE.big	Bei dieser Variante stehen leistungsstarken Cores energieeffiziente, langsamere Kerne zur Seite. Mittlerweile gibt es sogar dreistufige Varianten. So werkeln beim Snapdragon 888 von Qualcomm zum Beispiel neben vier energieeffizienten Cores (Cortex-A55) drei leistungsfähige, mit bis zu 2,4 GHz getaktete Prozessorkerne (Cortex-A78). Zu diesen sieben Kernen gesellt sich dann noch ein Prime genannter Kern (Cortex-X1), der mit bis zu 2,84 GHz getaktet wird und ausgesprochen leistungsstark ist. ARM nennt diese Technologie DynamIQ.



2 **Dank Scheduling-Klassen werkeln unterschiedliche Scheduling-Verfahren parallel.**

freut sich der Betriebssystemkern, wenn er per Konfiguration Hilfestellung bei seiner Aufgabe bekommt (siehe Kasten Runtime-Tuning des Task-Schedulers).

Neugierige Nerds

Wer sich mit Systemarchitekturen beschäftigt, benötigt folglich zwingend Kenntnisse über Scheduler und Scheduling, insbesondere wenn gehobene Ansprüche vorliegen. Nerds interessieren sich vielleicht generell für das spannende Stück Technik, das mittlerweile über viele Jahrzehnte gereift ist.

Auf jedem Prozessorkern läuft ein Singlecore-Scheduler mit seiner eigenen Liste von auf Bearbeitung wartenden Tasks. Diese Liste zu füllen oder ihr wieder Tasks zu entnehmen und auf andere Cores zu verschieben, ist hingegen die Aufgabe des Multicore-Schedulers.

Runtime-Tuning des Task-Schedulers

Über das Kommando `sysctl` lassen sich zur Laufzeit Kernel-Parameter des Task-Schedulers über das Setzen von Variablen parametrieren. Das Kommando `sysctl -A | grep "sched"` liefert die zur Verfügung stehenden Variablen (Listing 1). Mit Root-Rechten ausgestattet, lassen sich die Werte der Variablen per `sysctl Variable=Wert` anpassen. Die Bedeutung der Variablen ergibt sich größtenteils aus den Namen. Anmerkungen zu einigen davon finden sich in der Dokumentation des Linux-Kernels.

Einzelgänger

Linux stellt gleich mehrere Singlecore-Scheduling-Verfahren zur Wahl. Standardmäßig kommt das Completely Fair Scheduling (CFS) zum Einsatz, das für eine weitestgehend gerechte Verteilung der zur Verfügung stehenden Rechenzeit auf die wartenden Tasks sorgt. Im Detail ist CFS recht komplex.

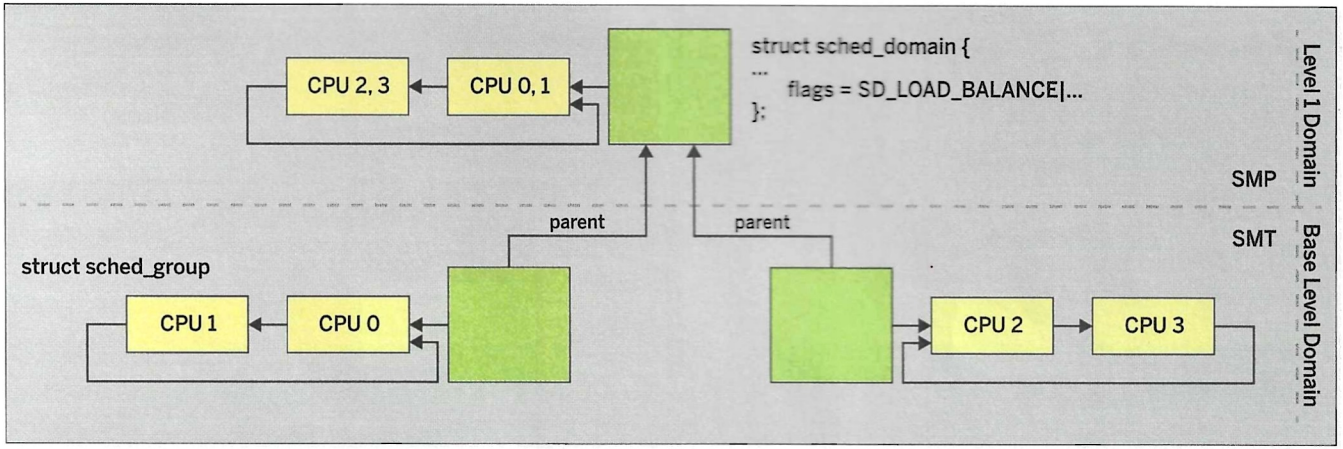
Realzeitsysteme setzen meist auf ein prioritätsgesteuertes Scheduling. Sie teilen Tasks eine Priorität zu, und der Scheduler wählt aus der Liste wartender Tasks denjenigen mit der höchsten Priorität aus. Der ausgewählte Task darf so lange laufen, bis er sich schlafen legt, terminiert oder ein Task mit höherer Priorität lauffähig wird. Gibt es mehrere Tasks mit identischer Priorität, gewährt der Scheduler die Vorfahrt entweder auf Basis eines Zeitscheibenverfahrens (Round Robin) oder per FIFO (First in, First out), also mittels eines Warteschlangenverfahrens. Bei Round Robin lässt sich zudem die Länge der Zeitscheibe festlegen.

Für Echtzeitsysteme eignet sich eher das sogenannte Deadline-Scheduling, Earliest Deadline First (EDF). Es wählt aus der Liste der wartenden Tasks den aus, der als nächster fertig bearbeitet sein muss. EDF ist nachweislich anderen Verfahren überlegen, wenn es darum geht, Zeitanforderungen einzuhalten. Andererseits muss der Programmierer dem Task-Scheduler mitteilen, bis zu welchem Zeitpunkt die Behandlung eines Ereignisses durch den Task abgeschlossen sein muss.

Mit CFS, prioritätengesteuertem Scheduling, FIFO, Round Robin und EDF gibt es genügend Wahlmöglichkeiten. Die gute Nachricht: Alle Verfahren lassen sich mehr oder minder gleichzeitig nutzen. Dazu hat Linus Torvalds Scheduling-Klassen eingeführt, die zu einer Priorisie-

Listing 1: Runtime-Tuning

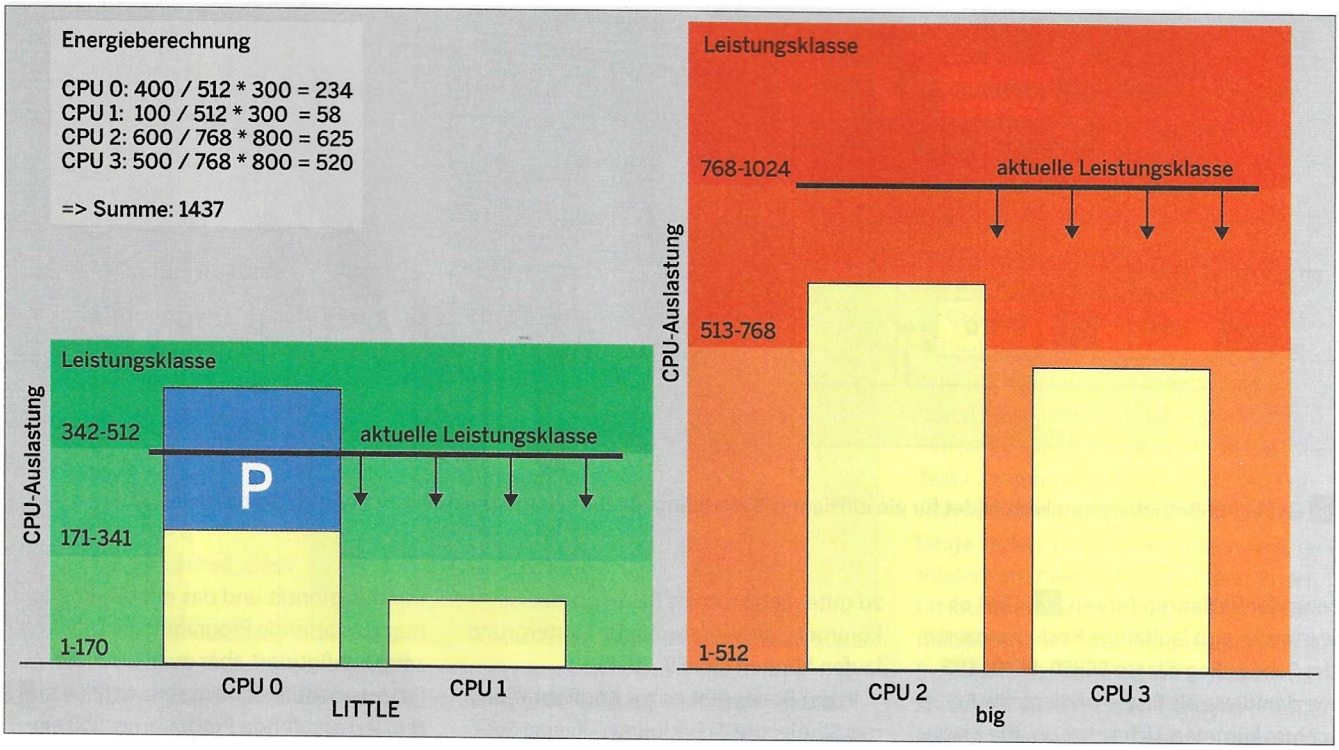
```
# sysctl -A | grep "sched"
kernel.sched_autogroup_enabled = 1
kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_deadline_period_max_us = 4194304
kernel.sched_deadline_period_min_us = 100
kernel.sched_energy_aware = 1
kernel.sched_rr_timeslice_ms = 100
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_schedstats = 0
kernel.sched_util_clamp_max = 1024
kernel.sched_util_clamp_min = 1024
kernel.sched_util_clamp_min_rt_default = 1024
```



3 Der Linux-Betriebssystemkern bildet für ein effizientes Scheduling die Hardwaretopologie in Scheduler-Domains ab.

rung der Verfahren führen 2. Gibt es wartende, also lauffähige Rechenprozesse der Scheduling-Klasse `SCHED_DEADLINE`, werden diese als Erstes bedient. Im Anschluss kümmert sich Linux um die Klasse `SCHED_RT`, also um Tasks mit einer Realzeitpriorität. Darauf folgt das Standard-Scheduling `SCHED_CFS`, bevor sich Linux zu guter Letzt um die Rechenprozesse kümmert, die wunderbar im Hintergrund laufen können (`SCHED_IDLE`).

In der Praxis gibt es zur Konfiguration des Singlecore-Schedulings diverse Systemcalls. Auf die Schnelle geht es aber auch im Terminal mithilfe des Kommandos `chrt`. Im einfachsten Fall gibt man nur die Priorität und das mit dieser Priorität zu startende Programm an. Das Werkzeug konfiguriert aber auch über die Prozessidentifikationsnummer (PID) referenzierte laufende Programme. Soll ein Task per Deadline-Scheduling verwaltet werden, muss man dazu die Laufzeit, die Periode und die Deadline spezifizieren.



4 **Das Energiemodell im Kernel liefert zu jeder CPU Leistungsklassen inklusive des zugehörigen Energiebedarfs.**

Tinder im Kernel

Stehen mehrere Prozessorkerne zur Verfügung, muss sich der Kernel auch um eine sinnvolle Verteilung der Tasks auf die Cores kümmern. Die dazu definierten, übergeordneten Optimierungsziele widersprechen sich dabei teilweise: Fairness, Echtzeitverhalten und Energieeffizienz stehen in Konkurrenz zueinander. Um auf einem Multicore-System einen passenden Match zwischen einem Rechenprozess und einem Prozessorkern zu finden, greift Linux tief in die Trickkiste.

Bereits beim Booten erforscht der Kernel detailliert die zugrunde liegende Hardwaretopologie. Er sammelt Informationen zu den einzelnen Prozessorkernen, deren Leistungsdaten und deren Anbindung untereinander. Bei diesem ersten Schritt geht es darum herauszufinden, welche Cores zur Verfügung stehen, wie leistungsfähig sie sind und wie teuer die Task-Migration kommt, also das Verschieben eines Tasks von einem CPU-Kern auf einen anderen.

Linux löst das Optimierungsproblem, indem es sogenannte Scheduling-Domains bildet. Eine solche Domain setzt sich aus Scheduling-Gruppen zusammen, jede Gruppe besteht aus einem Prozessorkern oder einer anderen Scheduling-Domain. Dadurch entsteht eine Baumstruktur, deren Blätter die einzelnen Cores repräsentieren. Abbildung **3** zeigt beispielhaft die Modellierung eines SMP-Systems, das zwei SMT-Prozessoren mitbringt. Für das Scheduling gibt es damit insgesamt vier CPU-Kerne.

Leistungsgesellschaft

Liegen in einer Scheduling-Domain identische Gruppen vor, balanciert der Multicore-Scheduler die Last innerhalb der Domain aus. Dafür ist die Leistungsorientierte Task-Verteilung zuständig (Capacity Aware Scheduling). Sie gleicht auf der einen Seite die Leistung der eingesetzten CPU-Kerne und auf der anderen Seite die benötigte Rechenleistung der zu verteilenden Tasks miteinander ab.

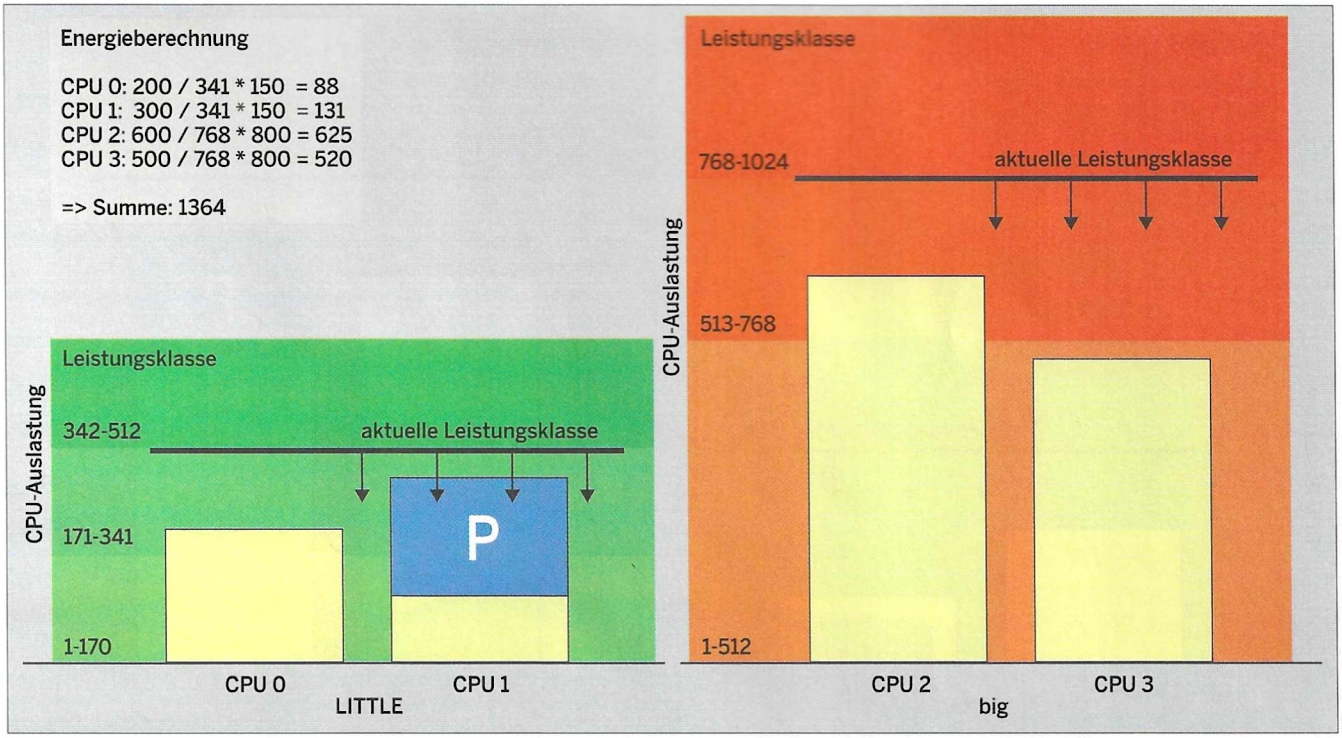
Die benötigte Rechenleistung wird als Auslastungsgrad (Utilization) angegeben. Die Leistung eines Prozessorkerns spezifiziert man vereinfacht über Leistung pro Takt (`work_per_hz(cpu)`) und die maximale Taktfrequenz (`max_freq(cpu)`). Die Multiplikation beider Angaben ergibt die maximale Leistung des CPU-Cores (`capacity(cpu)`).

Ständig busy

Die von einem Task benötigte Rechenleistung hat zunächst nichts mit der Leistung der CPU und der aktuell genutzten Taktfrequenz zu tun. Linux bestimmt den von diesen Parametern unabhängigen Wert eines Tasks (`task_util(p)`) für den

Energy Model

LITTLE		big	
Klasse	Energie	Klasse	Energie
179	50	512	400
341	150	768	800
512	300	1024	1700



5 **Falls Task P auf der CPU 1 läuft, können die energieeffizienten Kerne heruntertakten und verbrauchen insgesamt weniger Energie.**

Fall, dass der Prozess auf dem leistungsfähigsten Prozessorkern bei der höchsten Taktfrequenz abgearbeitet wird.

Genau genommen lässt sich die benötigte Rechenleistung eines Tasks selbst per Glaskugel nicht exakt vorhersagen. Sie schwankt typischerweise und hängt nicht nur von der Implementierung ab, sondern auch stark von den Eingangsdaten. Deshalb muss eine Abschätzung genügen, die das Kernel-Modul PELT (Per-entity Load Tracking) vornimmt und die grob auf dem Auslastungsgrad (`duty_cycle(p)`) basiert, den Linux über das Verhalten des Tasks in der Vergangenheit abschätzt. Tatsächlich liegt der Berechnung ein Polynom zugrunde.

Sobald die Parameter über die Leistungsfähigkeit der Kerne und die Anforderungen der Rechenprozesse vorliegen, kann die Lastverteilungskomponente des Multicore-Schedulers die Arbeit aufnehmen. Das Resultat variiert abhängig vom eingesetzten Basis-Scheduler. Im Fall von CFS kann ein Task dann auf einem Prozessorkern abgearbeitet werden, wenn dieser generell ausreichende Leistung für die Bearbeitung bietet (Fitness-Kriterium). Ein Task, der eine ständig aktive Endlosschleife implementiert und damit alles an Rechenzeit aufsaugt, was er bekommen kann, lässt sich gemäß dieses Kriteriums auf keiner CPU abarbeiten. Er wird als CPU-Bound bezeichnet und verbleibt typischerweise auf dem Kern, auf dem er gerade aktiv ist.

Übrigens kann man entweder per Systemcall `sched_setattr()` oder über Control Groups (per Uclamp) einen minimalen und einen maximalen Auslastungsgrad für einen Task angeben und damit die Auswahl des Prozessorkerns beeinflussen. So bleibt auch der Endlosschleifen-Task unter Kontrolle und lässt sich migrieren. Umgekehrt erlaubt das, das Abarbeiten eines an sich wenig Rechenzeit verbrauchenden Tasks auf einem leistungsfähigen CPU-Kern zu erzwingen.

Strom sparen

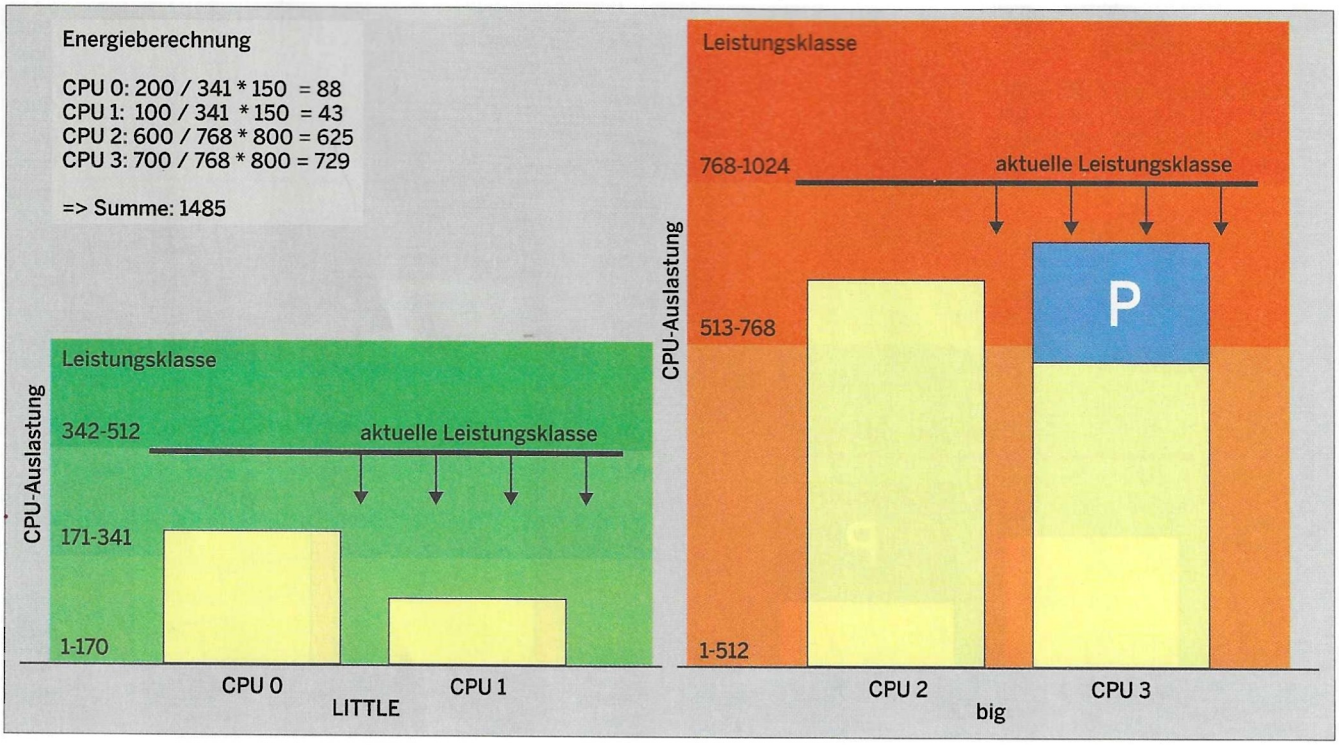
Anders sieht es aus, wenn die Scheduling-Gruppen innerhalb der Domain keine identischen Leistungsdaten aufweisen. Das trifft zum Beispiel auf ein LITTLE.big-System zu, wie man es bei den meisten Mobilgeräten vorfindet. Hier teilen sich sowohl die leistungsstarken als auch die energieeffizienten Prozessoren eine Domain. Bei solchen Systemen steht typischerweise die Energieeffizienz im Vordergrund, und Linux tauscht den normalen Balancing-Algorithmus gegen den Energy Aware Scheduler (EAS) aus.

Er verwendet für eine Migrationsentscheidung neben der Systemtopologie die Power-Management-Eigenschaften sowie den aktuellen Workload der Prozessorkerne, den er wie gehabt via PELT bestimmt. Die Power-Management-Eigenschaften stellt das Kernel-Modul Energy Management (EM) bereit. Es könnte diese Informationen beispielsweise einem Devicetree entnehmen, also einer statischen Konfiguration. Alternativ testet EM eine CPU aus.

Das Energy Model liefert zu jeder CPU die Daten in Form einer Tabelle von aufsteigend sortierten Leistungsenergiezuständen. Dazu teilt es eine CPU in eine Leistungsklasse ein, die es auf einen Wert bis 1024 skaliert. Zu jeder Leistungsklasse liefert EM einen Verbrauchswert.

Verbrauchsmessung

Mithilfe dieser Daten lässt sich der aktuelle Verbrauch einer Task-Verteilung auf Prozessoren berechnen. Soll der Scheduler beispielsweise entscheiden,



6 Bei einer Migration von P auf CPU 3 bliebe die CPU in ihrer Leistungsklasse, doch der schnelle Kern würde mehr Energie verbrauchen.

auf welchem der Prozessorkerne es einen Task P zu platzieren gilt, spielt er die verschiedenen Alternativen durch. Die Dokumentation zum Linux-Kernel zeigt das Vorgehen beispielhaft an einem aus vier Kernen bestehenden LITTLE.big-System. Das dazugehörige EM finden Sie in der Tabelle Energy Model.

Abbildung 4 zeigt die Leistungsenergiedaten der beiden energieeffizienten und der beiden leistungsstarken Cores und außerdem die auf den CPU-Kernen platzierten Tasks. Die beiden energieeffizienten Kerne bilden in diesem System eine sogenannte Power Domain, bei der beide Cores in der jeweils gleichen Leistungsklasse betrieben werden (grün dargestellt). Auch die zwei leistungsstarken Kerne umfassen eine Power Domain (rot).

Die Autoren

Eva-Katharina Kunst ist schon seit den Anfängen von Linux Fan von Open Source. **Jürgen Quade**, Professor an der Hochschule Niederrhein, gibt auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux.

Der jeweilige Verbrauch hängt außer von der Leistungsklasse noch von der tatsächlichen Last ab. Linux berechnet ihn nach der Formel *Energie = Last / Leistungsklasse * Energie_der_Klasse*.

Die zum Betrachtungszeitpunkt durchschnittliche Auslastung der Prozessorkerne (*util_avg*) liegt im Beispiel bei 400 für CPU 0, 100 für CPU 1, 600 für CPU 2 und 500 für CPU 3. Der zu platzierende Task P (blau dargestellt) hat auf der Skala bis 1024 eine Auslastung von 200 und ist zurzeit auf CPU 0 platziert. Die Energieberechnung ergibt für diese Konstellation einen Wert von 1437. Bei einer Migration des Tasks P auf Core 1 ergäben sich Kosten von 1364 5, bei einer Migration auf Core 3 solche von 1485 6. Demzufolge migriert der Scheduler aus Effizienzgründen den Task P auf Core 1.

Bei den vielen zu betrachtenden Faktoren, insbesondere der Anzahl der Performance-Domains, der Prozessorkerne und deren Leistungsklassen ergibt sich schnell ein großer Optimierungsraum. Linux berechnet daher durch Multiplikation der Faktoren einen Komplexitätswert, der 2048 nicht übersteigen darf. Dies ist neben der Asymmetrie der Cores eine Bedingung zum Einsatz von EAS.

Überblick behalten

Bunt gemischte Hardwarearchitekturen, eine Reihe sich widersprechender Optimierungskriterien, unbekannte Leistungsparameter und nicht von Beginn an vorhersagbare Anforderungen an Rechenzeit durch eine nicht vorhersehbare Anzahl von Tasks – all das verlangt dem Linux-Kernel einiges ab.

Um diesen Anforderungen gerecht zu werden, sammelt er alles an Informationen ein, was er bekommen kann, und passt sich dann ganz individuell an die vorhandene Hardware an. Das funktioniert in der Praxis erfreulich gut. Zugleich bietet Linux extrem viele Stellschrauben, um durch optimiertes Scheduling zu noch besseren Ergebnissen zu gelangen. Wer hohe Anforderungen an Energieeffizienz oder Zeitverhalten mitbringt, für den lohnt der Aufwand, sich intensiv mit der Konfiguration der Task-Verwaltung auseinanderzusetzen. (*jlu*)

Weitere Infos und interessante Links
www.lm-online.de/qr/47351