JITTY – ein einfaches, portables, Linux-kompatibles Betriebssystem

Volkmar Sieh, Bernhard Heinloth, Dustin Nguyen,
Wolfgang Schröder-Preikschat

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg
Volkmar.Sieh@fau.de, Dustin.Nguyen@fau.de, Bernhard.Heinloth@fau.de,
wosch@cs.fau.de

Zusammenfassung. Vor einiger Zeit wurde für unsere Betriebssysteme-Vorlesung Beispiel-Code gesucht. Aufgrund der Komplexität schied der Linux-Quelltext aus, System-Aufrufe im Minix-Betriebssystem sind aufgrund der durch den Mikrokern-Ansatz erforderlichen Kommunikation im Kern schwer nachzuverfolgen. Selbst *BSD-Systeme sind für diesen Einsatz noch zu unübersichtlich. Daher wurde am Lehrstuhl das **JITTY**-Projekt ins Leben gerufen. Ziel war ein in C geschriebener, übersichtlicher und leicht verständlicher, portabler, Linux-kompatibler SMP-Betriebssystem-Kern. Die üblichen Anwendungsprogramme einschließlich dem X-Window-System sollten unverändert lauffähig sein.
Um die Komplexität des Quelltextes möglichst gering zu halten, wurde der Fokus bei der Entwicklung auf „sauberen" Code gelegt: Priorisierung von lesbaren gegenüber performanten Code, Verwendung von Standard-Typen unter Einhaltung eines strikten Coding-Standards. Reduzierung von polyglotem Quelltext durch Verwendung von lediglich unvermeidbarem Assembler-Code und Ausschluß von Präprozessor Makros sowie Benutzung sehr weniger `#ifdef`-Blöcke und `#define`s sowie standardisierter `#include`s. Strikte Trennung zwischen Hardware-abhängigem und -unabhängigem Quelltext, inklusive Auslagerung des typischerweise „hässlichen" Codes zum Hochfahren des Rechners in einen extra Bootloader – das eigentliche Betriebssystem startet damit im C-Code mit allen CPUs laufend. Die Synchronisierung der CPUs erfolgt über wenige, einfach zu verstehende Funktionen zum Verwalten von Wartelisten sowie zum eigentlichen Warten. Die Koordinierung der CPUs geschieht ausschließlich mittels Umlaufsperren.
Die Funktionalität des Kerns ist nachvollziehbar auf wenige Module verteilt. Veraltete, nur aus Kompatibilitätsgründen vorhandene Systemaufrufe sind in einem Extra-Modul ausgelagert und rufen indirekt die aktuellen Systemaufrufe auf. Funktionalität, die für einen Betrieb als typischen Einzelplatzrechner nicht unbedingt gebraucht wird (z.B. Quotas, ACLs, NUMA-Support) wird bewusst weggelassen.
Es ergeben sich aktuell nur ca. 47k Zeilen Code für den generischen sowie ca. 8k Zeilen für den Hardware-abhängigen Teil (jeweils inklusiv vieler Kommentare), wobei verschiedene Architekturen unterstützt werden. Dazu kommen ca. 18k Zeilen Code für die Gerätetreiber.
Zur Erhöhung der Sicherheit des Systems werden alle Variablen-Zugriffe zur Compile-Zeit per Thread-Safety-Analyse überprüft. Mit Zeigern wird nur in Gerätetreibern, im Hardware-abhängigen Teil, bei Zugriffen auf Verzeichnisstrukturen in Dateisystemblöcken sowie beim Zerlegen/Zusammbauen von Netzwerk-Nachrichten gerechnet.
Zwar enthält der JITTY-Kern performante Datenstrukturen (Caches, Hash-Listen, usw.), durch den nicht an die Architektur angepassten Code ist die Performanz des Kerns aber sehr viel geringer als z.B. bei Linux. Daher wurden Werkzeuge entwickelt, die automatisch mittels Umlaufsperre geschützte kritische Abschnitte optimieren (z.B. durch verwendung von `atomic`-Instruktionen). Ein Boot-Zeit-Compiler erlaubt die Optimierung des Codes für die aktuelle Hardware beim Systemstart. Geplant ist ein Just-in-Time-Compiler mit Hotspot-Optimierung.

Danksagung. Die Arbeit wird unterstützt durch die Deutsche Forschungsgemeinschaft (DFG) unter den Förderkennzeichen SCHR 603/13-1 („PAX").