

PikeOS - eine Mikrokern-basierte Umgebung für partitionierte Systeme

Robert Kaiser

SYSGO AG,

Am Pfaffenstein 14, D-55270 Klein-Winternheim

rkaiser@sysgo.com

Februar 2004

Zusammenfassung

PikeOS ist eine Mikrokern-basierte Umgebung, in der mehrere Betriebssysteme vollständig gegeneinander geschützt ablaufen können. Die PikeOS- Systemsoftware unterteilt die Ressourcen (Speicher, Rechenzeit und Ein/Ausgabe Geräte) eines Rechensystems anhand einer in XML spezifizierten Konfiguration in verschiedene Untermengen, sogenannte „Partitionen“, auf die jeweils eine Betriebssysteminstanz garantierten, exklusiven Zugriff erhält.

Durch die so geschaffene Entkoppelung der verschiedenen Betriebssysteminstanzen ergeben sich zahlreiche neue Möglichkeiten: sicherheitskritische Systeme können ohne Risiko mit Standardanwendungen unter Linux kombiniert werden, Echtzeit- und Timesharing-Systeme können ohne Kompromisse in einer Maschine vereint werden, Consumer Geräte können gefahrlos ausführbare Inhalte aus dem Web laden, und Server können in mehrere virtuelle Maschinen aufgeteilt werden.

1 Entkoppeln durch Partitionierung

Das Prinzip der Softwarepartitionierung ist in der Flugzeugindustrie seit langem bekannt, und wird zum Beispiel in dem für diesen Bereich maßgeblichen Standard (DO-178B, [1]) beschrieben. Dabei werden die Ressourcen des Rechensystems (Speicher, Ein/Ausgabegeräte, Rechenzeit) anhand einer vorgegebenen Konfiguration auf verschiedene Subsysteme, so genannte „Partitionen“ aufgeteilt. Jeder dieser Partitionen wird der exklusive Zugriff auf ihren „Anteil“ des Rechners garantiert. Jede Partition ist somit vollständig von den anderen Partitionen unabhängig, d.h. ein in ihr ablaufendes Programm kann durch keinen irgendwie gearteten Fehler, der in einer anderen Partition geschieht, beeinträchtigt werden.

Wenn dieses Maß an Entkoppelung zuverlässig sichergestellt werden kann, so spricht nichts dagegen, in den verschiedenen Partitionen Programme mit unterschiedlichen Sicherheitsanforderungen ablaufen zu lassen, d.h. man könnte zum Beispiel ohne weiteres ein Linux-basiertes Navigationssystem und einen auf einem proprietären Kern basierenden Autopiloten gemeinsam in einem Rechner betreiben.

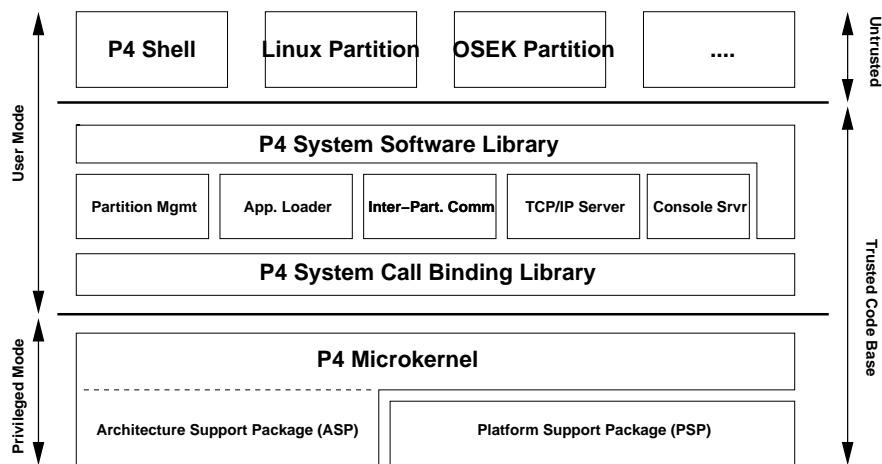


Abbildung 1: PikeOS Systemarchitektur

Natürlich gibt es auch in einem solchen Szenario eine „trusted code base“, d.h. Code, der im Fehlerfall das gesamte System zum Absturz bringen kann: dabei handelt es sich um eben diejenige Softwareschicht, die die Partitionierung bewerkstelligt. Sie muss, um zertifizierbar zu sein, nach den Regeln der einschlägigen Standards entwickelt werden, und sie muss mit wenig Code auskommen, d.h. sie muss sich in ihrer Funktionalität auf das Wesentliche beschränken.

2 Ein Framework für partitionierte Systeme

Das hier vorgestellte PikeOS, ein Entwicklungsvorhaben der SYSGO AG, ist kein neues Betriebssystem im herkömmlichen Sinne, sondern eine Umgebung, in der mehrere Betriebssysteme kontrolliert ablaufen können.

PikeOS basiert auf dem Mikrokern „P4“. Dieser Mikrokern ist dabei das einzige Stück Code im gesamten System, das im privilegierten Modus abläuft. Unter Verwendung der (lediglich sieben) Systemaufrufe des Mikrokerns realisiert eine „System Software“ Schicht das beschriebene Partitionierungsmodell (siehe Abb. 1).

Der Partition Manager übernimmt dabei die Erzeugung und Kontrolle der Partitionen, in denen die verschiedenen Betriebssysteme jeweils im User Mode ablaufen. Er kann einzelne Partitionen starten und zurücksetzen, wobei dieser Vorgang optional auch über die in einer eigenen Partition ablaufende „P4 Shell“ interaktiv ausgelöst werden kann.

Der Application Loader lädt -beim Systemstart oder auf Anforderung- ausführbare Inhalte in die jeweiligen Partitionen. Diese Inhalte können auf verschiedene Weisen in das System gelangen: so ist es zum Beispiel möglich, Software aus einem ROM oder über das Netzwerk zu laden, aber ebenso kann auch eine der bereits aktiven Partitionen als „File Provider“ fungieren, sodass zum Beispiel in einer Linux-Partition mit den standard- Linux-Werkzeugen ein Programm erstellt werden kann, das dann eigenständig in einer anderen Partition abläuft. Zusätzlich stellt die System Software noch verschiedene, teils optionale Dienste

zur Verfügung, wie zum Beispiel eine Schicht für die Kommunikation zwischen den Partitionen, oder für den Zugriff auf gemeinsam genutzte Ressourcen (beispielhaft sind hier TCP/IP und Konsole gezeigt) ermöglichen.

3 Mikrokern P4

P4 ist eine Eigenentwicklung von SYSGO. Er basiert ursprünglich auf den Konzepten des von J. Liedtke 1995 vorgestellten Mikrokerns „L4“ [2], weist aber gegenüber anderen L4-Implementierungen einige Besonderheiten auf. P4 implementiert drei wesentliche Primitive:

- Adressräume
- Aktivitäten (Threads)
- Inter-Prozess Kommunikation (IPC)

Dabei ist der IPC-Mechanismus nicht einfach nur ein Mittel zum Austausch von Daten, sondern er kann auch Rechte (z.B. Zugriffsrechte auf Speicherseiten) zwischen Threads in verschiedenen Adressräumen austauschen. Interrupts und Ausnahmebedingungen werden nicht vom Mikrokern selbst behandelt, sondern er wandelt sie lediglich in IPC-Nachrichten um, die einem registrierten Handler-Thread zugestellt werden.

P4 ist weitgehend in C geschrieben. Intern besteht er aus drei Komponenten:

1. Der generischen Schicht: diese macht etwa 2/3 des gesamten Codes aus.
2. Der architekturenspezifischen Schicht (ASP): Das sind die Teile des Codes, die sich mit den architektur- (nicht aber board-) spezifischen Eigenschaften des Systems befassen.
3. Dem „Platform Support Package“: Dieses befasst sich mit den Board-spezifischen Eigenheiten der Hardware. Es ist als separates Binärmodul ausgeführt und kann auch stand-alone, d.h. ohne den restlichen Mikrokern entwickelt/getestet werden.

Wie aus dieser Aufteilung unmittelbar deutlich wird, wurde P4 auf gute Portierbarkeit ausgelegt. Mittlerweile läuft der Kern neben x86 PCs auch auf verschiedenen ARM-, PowerPC- und MIPS-Systemen.

4 Partitionierung im Detail

4.1 Speicherressourcen

Die Partitionierung von Speicherressourcen wird mit Hilfe der P4-Adressräume umgesetzt: Ein Thread unter P4 ist immer einem Adressraum zugeordnet (dieser kann gleichzeitig bis zu 128 Threads beinhalten). Jeder Thread kann immer nur Speicherseiten aus seinem eigenen Adressraum an Threads in anderen Adressräumen weitergeben, d.h. ein Thread kann immer nur über die Ressourcen verfügen, die er selbst besitzt. Somit lässt sich die Partitionierung von Speicherressourcen direkt über P4-Adressräume abbilden: beim Start des Systems

existiert zunächst nur ein einziger Adressraum, der sämtliche Speicherseiten des Systems enthält. In diesem Adressraum läuft die System Software. Diese erzeugt nun anhand der vorgegebenen Konfiguration für jede Partition einen Adressraum, an den sie die entsprechende Anzahl Speicherseiten abgibt.

4.2 Ein/Ausgabegeräte

Ein-/Ausgabegeräte werden über Register entweder memory-mapped oder (bei x86er Prozessoren) port-mapped angesteuert. Das bedeutet, die Zugriffsrechte auf memory-mapped Register sind nichts grundsätzlich anderes als Zugriffsrechte auf Speicherseiten. Auch der Port-Adressraum kann unter P4 mit den gleichen Mitteln aufgeteilt werden, wie der Speicher-Adressraum. Der den Operationen zugrunde liegende Mechanismus (die I/O Permission Bitmap) ist in diesem Fall zwar völlig anders, aber die Mikrokern-Schnittstelle versteckt dies. Interessant ist in diesem Zusammenhang, dass P4 den Port-Adressraum Bytegranular verwalten kann, d.h. jede einzelne Portadresse kann individuell einem anderen Adressraum zugewiesen werden.

Interrupts werden vom Mikrokern in IPC-Messages umgewandelt und an zuvor registrierte Interrupthandler Threads übermittelt. Somit stellen sich Interrupts als virtuelle Absender von Nachrichten dar.

P4 erlaubt Interruptsharing über Partitions Grenzen hinweg, d.h. mehrere Threads aus verschiedenen Partitionen können sich für den gleichen Interrupt registrieren.

4.3 Zeitpartitionierung

Für die Aufteilung der Ressource „Rechenzeit“ auf verschiedene Partitionen existiert der ARINC 653-Standard [3]. Das dort beschriebene Verfahren arbeitet mit einer festen Zykluszeit, innerhalb derer die einzelnen Partitionen in einer festgelegten Reihenfolge für die ihnen zugesicherte Dauer den Prozessor zugeteilt bekommen. Der PikeOS Partition Scheduler kann unter anderem diesen Standard vollständig abbilden, er bietet jedoch darüber hinaus noch Möglichkeiten, die im ARINC-Standard nicht vorgesehen sind.

Dazu können „Prioritätsbänder“ definiert werden, innerhalb derer die Rechenzeit nach dem ARINC-Verfahren verteilt wird. Threads, deren Priorität unterhalb eines solchen Prioritätsbandes liegen, kommen nur dann zum Zuge, wenn der im ARINC-Bereich gerade aktive Thread keine Verwendung für die ihm zugesicherte Rechenzeit hat.

So können zum Beispiel zusätzlich explizite nicht-Echtzeitpartitionen definiert werden, denen die „übrig bleibende“ Rechenzeit zugeteilt wird.

Darüber hinaus besteht die Möglichkeit, hochprioritäre Threads zu definieren, die sich über die Zeitpartitionierung hinwegsetzen können.

Selbstverständlich sind solche Threads aus der Sicht der Threads, die sie unterbrechen können, als „trusted code“ zu betrachten. Deshalb ist wichtig, dass die Funktionen zur Prioritätenvergabe im Mikrokern realisiert sind, und dass dieser sicherstellt, dass kein Thread sich Rechte erschleichen kann, die ihm nicht zustehen.

P4 löst dies, indem er jedem Thread eine Maximalpriorität zuordnet. Dieser Wert wird beim Erzeugen des Threads festgelegt und er ist auf die Maximal-

priorität des Erzeuger-Threads begrenzt. Er kann während der gesamten Lebensdauer nicht mehr verändert werden.

4.4 Inter-Partition-Kommunikation

PikeOS bietet Möglichkeiten der Kommunikation zwischen den Partitionen. Dies ist erforderlich, um z.B. einen „Health-Monitor“ zu realisieren: Dieser läuft als Task in einer eigenen Partition und hat die Aufgabe, die anderen Partitionen zu überwachen und gegebenenfalls zurückzusetzen. Zudem kann dieser Mechanismus während der Entwicklungsphase genutzt werden, um komfortabel unter Linux Applikationen zu kompilieren und ihren Ablauf in einer anderen Partition zu überwachen/debuggen.

Für die Funktionalität solcher Kommunikationsschnittstellen zwischen Partitionen existiert ebenfalls ein ARINC-Standard [4], an dem sich die PikeOS-Implementierung orientiert. Je nach Betriebssystem stellen sich diese Schnittstellen unterschiedlich dar (unter Linux sind es zum Beispiel zusätzliche „pseudo-tty“). Die Verbindungsmatrix, d.h. die Festlegung welche der Schnittstellen miteinander verbunden werden, wird bei der Systemkonfiguration in Form einer XML-Datei festgelegt.

5 Personalities

Innerhalb der PikeOS-Partitionen können verschiedenste Betriebssysteme ihre Arbeit verrichten. Dabei ist für das einzelne System der Vorgang der Partitionierung transparent: es „merkt“ nicht, dass ihm nur eine Untermenge der tatsächlich vorhandenen Ressourcen zur Verfügung steht.

Die Betriebssysteme müssen, damit sie in einer Partition laufen können, angepasst werden. Derzeit existieren zwei Betriebssysteme, die, bei Bedarf auch mehrfach instantiiert, unter PikeOS laufen können, mehrere weitere sind in der Entwicklung, bzw. Planung.

5.1 Linux/P4

Um dem PikeOS-System Zugang zur Linux-Welt zu geben, wurde eine Portierung des Linux-Kerns auf den Mikrokern P4 vorgenommen. Dabei läuft Linux vollständig im nichtprivilegierten Modus. Dennoch ist es binärkompatibel zum Standard Linux, d.h. sämtliche Anwendersoftware und auch die meisten Gerätetreiber können unverändert übernommen werden.

Als Ausgangspunkt dieser Linux-Portierung diente User Mode Linux [5], das den (technisch ähnlichen) Ansatz verfolgt, Linux als Anwenderprogramm unter Linux verfügbar zu machen.

Der Kern basiert auf der relativ aktuellen Version 2.4.20. Sobald die kürzlich erst freigegebene 2.6er Kernserie hinreichend stabil ist, wird es auch eine Linux/P4 Version dieser Serie geben.

5.2 OSEK/P4

OSEK OS [6] ist ein Standard der Europäischen Automobilindustrie. Es definiert die Programmierschnittstelle eines Echtzeit-Betriebssystems für die Sy-

stem Software, die auf den elektronischen Steuergeräten (ECU) im Auto eingesetzt werden. SYSGO hat eine Implementierung dieser Schnittstelle auf Basis des Echtzeitsystems LynxOS entwickelt, die nun auf die Mikrokernschnittstelle portiert wurde.

Damit besteht die Möglichkeit, u.U. mehrere OSEK Instanzen unter Echtzeitbedingungen in einem Steuergerät völlig voneinander entkoppelt zu betreiben.

5.3 POSIX Threads

Für Herbst 2004 ist eine Implementierung des POSIX Threads Standards (PSE51) unter PikeOS geplant. Ähnlich wie bei OSEK soll damit die Möglichkeit geschaffen werden, kompakte Echtzeitsysteme in separaten Partitionen zu realisieren.

5.4 Java Runtime Environment

Auch eine Java Virtual Machine (JVM) kann in einer eigenständigen Partition ablaufen. Dies ist vor allem für zukünftige Consumer Geräte (Handy, PDA, etc.), die in zunehmendem Maße Web-fähig werden, interessant: Ein solches Gerät kann dann beliebige Java-Programme aus dem Web laden und ausführen, ohne dabei zu riskieren, dass die vitalen Funktionen des Gerätes beeinträchtigt werden, oder dass etwa sicherheitsrelevante Daten „ausspioniert“ werden könnten. Die Portierung einer geeigneten JVM wird zur Zeit bei SYSGO projektiert.

6 Tools

Die Konfiguration eines PikeOS-Systems wird in Form von XML-Dateien definiert. Zur Erstellung dieser Dateien kann ein beliebiger XML-Editor verwendet werden. Zur geplanten Markteinführung wird jedoch auch ein auf die Bedürfnisse von PikeOS zugeschnittener spezieller Wizard verfügbar sein, der mit Hilfe von sinnvollen Defaults und Plausibilitätsprüfungen die Erstellung solcher Konfigurationsdateien erheblich vereinfachen wird.

7 Zusammenfassung und Ausblick

PikeOS ermöglicht das Zusammenspiel verschiedenster Betriebssysteme in einem Gerät, ohne dass dabei Kompromisse hinsichtlich Sicherheit oder Echtzeitfähigkeit gemacht werden müssen. Es existiert bereits eine kleine, aber wachsende Zahl von Systemen, die in dieser Umgebung funktionieren. Für die Zukunft wird die Portierung weiterer Systeme auf PikeOS erwartet. Nahe liegende Kandidaten wären etablierte Echtzeitsysteme (vxWorks, pSOS), aber auch zum Beispiel ein Ada Runtime System.

Literatur

- [1] RTCA, 1992, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc., Document No. RTCA/DO-178B.
- [2] Jochen Liedtke, 1996, *L4 Reference Manual - 486, Pentium and Pentium Pro*, GMD Arbeitspapier 1021, <http://os.inf.tu-dresden.de/L4/14doc.html>.

- [3] Aeronautical Radio, Inc., 1997, *Avionics Application Software Standard Interface*, ARINC Specification 653.
- [4] Aeronautical Radio, Inc., 2003, *Draft 2 of Supplement to ARINC Specification 653 Avionics Application Software Standard Interface*, ARINC, Ref. 03-072/SWM-87 bbb.
- [5] Jeff Dike, 2000 *A User-mode port of the Linux kernel*, <http://user-mode-linux.sourceforge.net/als2000/index.html>.
- [6] OSEK, 1996, *OSEK/VDX Operating System, Version 2.0*, <http://www.osek-vdx.org>.