Großer Beleg

Uniform Mandatory Access Control in L4Env

Maria Klemm

20. Oktober 2008

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Betriebssysteme

Betreuender Hochschullehrer: Prof. Dr. rer. nat. Hermann Härtig
Betreuender Mitarbeiter: Dipl.-Inf. Stefan Kalkowski

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Dresden, den 20. Oktober 2008

Maria Klemm

Danksagung

Inhaltsverzeichnis

1 Einführung **1**
1.1 Aufgabestellung 1
1.2 Überblick 2

2 Grundlagen und Stand der Technik **3**
2.1 Sicherheit 4
2.2 Zugriffssteuerung 4
2.3 Modelle und Mechanismen 4
2.4 L4 Environment 5
2.5 Verwandte Arbeiten 6

3 Entwurf **9**
3.1 Designraum 9
3.2 Compartment Manager 10
3.3 Instrumentierung der Basis 11

4 Implementierung **15**
4.1 Basis 15
4.2 Manager 16
4.3 SLOC 17

5 Leistungsbewertung **19**
5.1 Messungen 19
5.2 Bewertung 20
5.3 Offene Probleme 20

6 Zusammenfassung und Ausblick **21**

Glossar **25**

Literaturverzeichnis **27**

1 Einführung

In einem Multitasking System müssen physische und abstrakte Ressourcen (z.B. Namen oder Kommunikationskanäle) geeignet verwaltet werden, so dass möglichst alle Tasks arbeiten können. Dies trifft auch für das L4 Environment (L4Env) und den L4 Mikrokern Fiasco zu.

Das L4Env hat keine einheitliche Schnittstelle für die Zugriffsteuerung auf Ressourcen und ist somit nicht in der Lage den Zugriff auf physische oder abstrakte Objekte zu vereinheitlichen. Mit einer Schnittstelle, die über eine globale Regelbasis verfügt und einer Isolierung in lokale Bereiche kann die Nutzung von Ressourcen zuverlässiger organisiert werden.

Der Zugriff auf Ressourcen im L4Env unterscheidet sich von Server zu Server und unterliegt teilweise gar keiner Kontrolle. Es können für eine Anwendung, der man nicht vertraut, keine beschränkenden Regeln aufgestellt werden, die das gesamte L4Env befolgt.

Die Zuverlässigkeit im L4Env kann gefährdet sein, wenn eine Task eine Ressource (z.B. den gesamten physischen Speicher) für sich reserviert und den übrigen Tasks die Nutzung dieser Ressource vorenthält.

Anwendungen denen man nicht vertraut, sollen isoliert zu den vertrauenswürdigen Servern und Bibliotheken des L4Envs stehen, so dass man sie in ihrem Ressourcenverbrauch und ihrer Kommunikation einschränken kann.

Eine Isolierung und Beschränkung dieser Art wird als „Sandboxing“ oder „Compartmentalization“ bezeichnet, mit einem Compartment oder einer Sandbox als abgegrenzte Sektion.

Mit der zentralen Verwaltung von Rechten auf Ressourcen und einer Isolierung von Objekten (physische Ressourcen) und Subjekten (Tasks) kann man das L4Env zuverlässiger im Bezug auf Bedrohungen der Integrität und Verfügbarkeit gestalten.

1.1 Aufgabestellung

Das aktuelle L4Env hat verschiedene Formen der Zugriffsteuerung (Access Control), abhängig von der jeweiligen Komponente, die Ressourcen verwaltet (z.B. DMphys) oder Kommunikation kontrolliert (z.B. IPCmon). Wenn man Mandatory Access Control (MAC) im L4Env einsetzen will, wäre ein Interface notwendig, mit dem man einheitliche Regeln für das ganze System definieren kann, so dass die Zugriffsteuerung von Tasks in Bezug auf Ressourcen und andere Tasks eingeschränkt werden kann.

Das Ziel meiner Arbeit ist die Entwicklung eines zentralen Compartment Managers für das L4Environment. Tasks sollen zu Compartments gruppiert werden, so dass ihr Ressourcenverbrauch und ihre Kommunikation beschränkbar sind.

1.2 Überblick

Die Arbeit ist wie folgt aufgebaut. Im 2. Kapitel Grundlagen und der Stand der Technik beschrieben. Der Entwurf des Compartment Managers und die Instrumentierung des L4Environments bilden das 3. Kapitel. Das 4. Kapitel zeigt einige Details der Implementierung und das 5. Kapitel bewertet die entstandene Lösung mit Hilfe kleiner Benchmarks. Die Zusammenfassung und ein Ausblick auf zukünftige Aufgaben schließen diese Arbeit ab.

2 Grundlagen und Stand der Technik

Dieses Kapitel beschreibt die grundlegenden Ideen und Begriffe die für meine Arbeit wichtig sind. Es geht auch auf Themen ein, die mit meiner Arbeit in Verbindung stehen, aber von der Aufgabenstellung abgegrenzt wurden.

Einem Betriebssystem obliegt die Aufgabe, die Ausführung der Programme zu steuern und die vorhandenen Ressourcen für diese Prozesse zu verwalten. Im Gegensatz zu einem monolithischen Kern werden in einem L4 Mikrokern nur Threads, separate Adressräume und Interprozesskommunikation (IPC) realisiert. Alle weiteren Mechanismen, deren Implementierung eine bestimmte Strategie (im Englischen „Policy“) zugrunde liegt, befinden sich außerhalb des Mikrokerns. So bildet das L4 Environment (L4Env) für den L4 Mikrokern Fiasco eine Umgebung, die L4 Anwendungen verschiedene Dienste wie Speicherverwaltung, Synchronisationsmechanismen oder einen Namensdienst zur Verfügung stellt.

Eine Zugriffsbeschränkung im Allgemeinen lässt sich auf der Ebene der Anwendungen realisieren, wo z.B. durch eine Java Virtual Machine eine abgeschottete Umgebung (Sandbox) für eine bestimmte Java Anwendung bereit gestellt wird. Ein weiteres Beispiel ist Security Enhanced Linux (SELinux). Bei SELinux wird der Kern um Mechanismen erweitert die es ermöglichen, jede Anwendung in einer eigenen Sandbox unterzubringen. Das Prinzip Sandboxing lässt sich aber auch auf ganze Betriebssysteme anwenden, die dann mit Unterstützung eines Virtual Machine Monitors (z.B. Xen) in ihrem jeweils eigenen Compartment ausgeführt werden.

Diese Form der Isolierung macht es möglich, dass Anwendungen in einem Compartment zwar ausgeführt werden, aber im Verbrauch ihrer Ressourcen und in der Kommunikation mit anderen Anwendungen einschränkt werden. Unbegrenzter Zugriff eines Subjekts auf gemeinsam verwendete Ressourcen kann die Verfügbarkeit oder Lebendigkeit des Systems beeinträchtigen. Durch uneingeschränkte Kommunikation kann die Geheimhaltung sensibler Daten oder die Integrität zuverlässiger Dienste gefährdet werden. Eine Anwendung, die das L4 Environment benutzt, wird beispielsweise nicht daran gehindert, sich allen verfügbaren Arbeitsspeicher auf einmal zu reservieren. Sie kann sich ohne großen Aufwand z.B. für den Logserver ausgeben, wenn sie es schafft sich vor ihm, mit seinem Namen, beim Namensdienst anzumelden.

Das gegenwärtige L4Env kann eine einheitliche Zugriffsbeschränkung nicht realisieren, weil es, je nach Komponente, verschiedene Zugriffsformen hat, z.B. Discretionary Access Control (DAC) bei der Verwaltung des physischen Speichers. Die einheitliche Beschränkung einer Anwendung in all ihren Aktionen kann mit einem Server verwirklicht werden, der die globale Zugriffsstrategie zur Verfügung stellt. Durch eine geeignete Anpassung der zuständigen Dienste im L4Env, wären diese dann in der Lage, die Anwendung in ihrem Ressourcenverbrauch und ihrer Kommunikation zu beschränken.

2.1 Sicherheit

Einen umfassenden Überblick zu Theorie und Umsetzung sicherheitsrelevanter Probleme hat Matt Bishop in seinem Buch „Computer Security - Art and Science“ [Bis03] zusammengetragen. Die drei wichtigsten Schutzziele sind Eindeutigkeit von Sender und Empfänger, Verfügbarkeit von Ressourcen und integeren Daten und Vertraulichkeit. Entsprechend dazu beschreibt er vier potenzielle Bedrohungen der Sicherheit: Der Zugriff auf geheime Informationen, Betrug, die Störung des korrekten Ablaufs und eine unbefugte Steuerung eines Teilsystems.

2.2 Zugriffssteuerung

J. Shapiro sieht in der Zugriffskontrolle das grundlegende Problem, welches im Hinblick auf Sicherheit in Computersystemen untersucht und gelöst werden muss. Der folgende Abschnitt basiert auf seinem Essay über Capabilities [Sha99]:

Wie und auf welche Objekte kann ein gegebenes Programm zugreifen? Objekte sind hier Dateien, Geräte, andere Programme oder ähnliches. Der Zugriff auf Objekte lässt sich gleichsetzen mit den Operationen, die auf diesen Objekten ausgeführt werden können. Ein Subjekt kann beispielsweise aus einer Datei lesen, sie verändern, löschen oder eine neue Datei erzeugen und sie mit einem anderen Programm gemeinsam nutzen. Wenn man über Zugriffssteuerung spricht, sind hauptsächlich vier Vorgänge gemeint. Der Zugriff soll verhindert oder beschränkt werden bzw. will man Zugriff gewähren und widerrufen können.

2.3 Modelle und Mechanismen

Alle Modelle für Zugriffskontrolle stellen die Beziehung zwischen Subjekten und Objekten dar. M. Bishop beschreibt das grundlegende Modell, die Access Control Matrix (ACM) folgendermaßen: *Jede Zeile definiert für ein Subjekt, auf welche Objekte es Zugriff hat und in welcher Art. Analog dazu definiert jede Spalte für ein Objekt die zugriffsberechtigten Subjekte und ihre möglichen Operationen. Eine sparsame Darstellung listet entweder die Zeilen oder die Spalten der ACM auf. Daraus hat sich ergeben, dass man Systeme, die Zugriffsmechanismen realisieren, in zwei Kategorien einteilt. Sie basieren entweder auf Access Control Lists (Spalten) oder auf Capability Lists (Zeilen der ACM).*

J.Shapiro [Sha99] erklärt den Begriff Capability noch ausführlicher: *Angenommen man entwirft ein Computersystem so, dass ein Programm eine spezielle Marke besitzen muss, um auf ein Objekt zugreifen zu können. Diese Marke bezeichnet ein Objekt und gibt dem Programm die Berechtigung, eine bestimmte Menge von Aktionen (wie Lesen oder Schreiben) auf diesem Objekt auszuführen. So eine Marke nennt man Capability.* Desweiteren zählt er folgende Eigenschaften von Capabilities auf:

- *Sie können dasselbe Objekt bezeichnen, aber zu verschieden Aktionen berechtigen.*
- *Capabilities können übertragen werden.*
- *Sie können kopiert werden.*

- *Sie müssen fälschungssicher sein.*

M. Bishop [Bis03] definiert die drei grundlegenden Formen der Zugriffssteuerung so:

Discretionary Access Control (DAC) *Wenn ein individueller Nutzer, den Mechanismus für die Zugriffssteuerung einstellen kann, um den Zugriff auf ein Objekt zu erlauben oder zu verbieten, dann realisiert dieser Mechanismus DAC. Diese Form der Zugriffskontrolle wird auch identitätsbasierte Zugriffsteuerung (IBAC) genannt.*

Mandatory Access Control (MAC) *Wenn ein Systemmechanismus den Zugriff auf ein Objekt steuert und ein individueller Nutzer die Zugriffsteuerung nicht ändern kann, dann ist das MAC, manchmal auch regelbasierte Zugriffskontrolle genannt.*

Originator Controlled Access Control (ORCON/ORGCON) *Bei einer vom Urheber kontrollierten Zugriffssteuerung basiert der Zugriff auf ein Objekt (oder auf die Information, die es enthält) auf dem Erzeuger.*

MAC wird zum Schutz von Vertraulichkeit und Integrität eingesetzt. In Multi-Level-Security Systemen werden den Subjekten und Objekten Security- oder Integrity-Level zugeordnet. Durch das Aufstellen von Regeln, welche den Zugriff von einem Level auf ein anderes definieren, wird festgelegt, welche Grenzen dem individuellen Nutzer gesetzt sind. Die bekanntesten Vertreter solcher formaler Modelle sind das **Bell-LaPadula**- und das **Biba**-Modell.

Type Enforcement weist Subjekten Domänen und Objekten Typen zu und vereinfacht somit die Regelbasis. Mit eindeutigen Zugriffsregeln von Domänen auf Typen lässt sich so das Designprinzip „Least Privilege [1]“ realisieren.

2.4 L4 Environment

Die vorliegende Arbeit wurde für das L4Env und den L4 Mikrokern Fiasco [Fia] entwickelt. Beide entstanden im Rahmen des Projektes DROPS, dem Dresden Real-Time Operating System [DRO] an der TU Dresden.

Das L4Env ist eine einheitliche Programmierumgebung zur Entwicklung und Ausführung von Anwendungen für L4 bzw. für Fiasco. Neben den grundlegenden L4 Abstraktionen wie Tasks, Threads und IPC stellt es Speichermanagement, Synchronisationsmechanismen, Konsolenein & -ausgabe, einen globalen Namensdienst und weitere Funktionen zur Verfügung. Diese Server und Bibliotheken bilden keine abgeschlossene Menge, sondern beinhalten neben dem **Core Interface** [Kau06] auch experimentelle Dienste, die noch in der Entwicklung stehen oder für spezielle Anwendungen eingesetzt werden. Im Folgenden werden die für meine Arbeit relevanten Elemente kurz beschrieben.

Dataspace Manager

Speicherverwaltung im L4Env basiert auf dem Modell von Dataspaces und Dataspace Managern [ALE+01]. Als Dataspace (DS) bezeichnet man einen Container der Daten beliebiger Art

[1] Einem Subjekt sollen lediglich die Rechte gegeben werden, die es braucht um seine Funktion zu erfüllen.

enthalten kann, beispielsweise anonymen Speicher, Dateien oder physischen Speicher. Der Containerinhalt wird im Falle des L4Env durch spezifische DS Manager verwaltet. Die Verwaltung des physischen Speichers übernimmt der Server DMphys.

Der physische Arbeitsspeicher wird den Tasks in Form von DSs zur Verfügung gestellt. Der Zugriff auf einen DS unterliegt der Kontrolle seines Eigentümers, ursprünglich jener Task die den DS geöffnet hat. In welcher Form andere Tasks auf diesen DS zugreifen dürfen, bestimmt eine taskbasierte ACL, die nur durch den Eigentümer verändert werden kann. Zudem darf nur er den Transfer, das Schließen und Verkleinern (oder Erweitern) des DSs veranlassen. DMphys realisiert also Discretionary Access Control, wobei für die Speichermenge, die eine Task benutzen kann, kein Maximum definiert ist.

Loader und Taskbibliothek

Über den Loader können L4 Anwendungen geladen und gestartet werden. Die Binärdatei der Anwendung (oder ein Konfigurationsskript mit zusätzlichen Ausführungsoptionen) erhält der Loader in einem Dataspace. Die experimentelle Taskbibliothek, Nachfolger des Taskservers (jetzt ein Thread des Loaders) erzeugt und zerstört L4 Tasks zur Laufzeit. Die Tasks bilden eine baumartige Struktur, in der jede Task durch ihren *Parent* erzeugt wird und eine Anfrage, wie das Reservieren einer weiteren TaskID, an den *Parent* weitergeleitet wird. Eine Sicherheitspolitik kann somit verfolgt werden in dem eine Task Ressourcen oder Information bei ihrem Parent erfragt und gegebenfalls zugewiesen bekommt. Somit kann von der Wurzel aus (oder einem anderen *Parent*) eine Sicherheitsstrategie für die *Child* Tasks realisiert werden.

Capability Faulthandler

Kontrollierte Kommunikation zwischen Tasks wird in Fiasco mit Capabilities realisiert. Tasks können nur miteinander kommunizieren, wenn sie die passenden Capabilities besitzen. Das Fehlen der entsprechenden Capability löst einen Fault aus, der vom Faulthandler außerhalb des Kerns behandelt wird. Beim Start der Tasks wird ihnen ein Capability Faulthandler zugewiesen, der ihnen Capabilities die er besitzt weitergeben kann. Er bekommt eine Policy zugewiesen oder kann von einem Policy Manager konfiguriert werden. IPCmon ist die prototypische Implementierung eines konfigurierbaren Faulthandlers.

2.5 Verwandte Arbeiten

Zwei wissenschaftliche Arbeiten des Lehrstuhls Betriebssysteme, die sich mit Geräten und Dateien auseinandergesetzt haben, kann man mit einem Policy Manager verbinden:

C. Weinhold [Wei06] hat in seinem Diplom ein vertrauenswürdiges Dateisystem entwickelt. Dieses Dateisystem für L4 wurde mit dem Ziel entwickelt die eingebauten Zugriffskontrollmechanismen traditioneller Betriebssysteme zu verbessern bzw. zu ersetzen. In seinem Ausblick beschreibt Weinhold, dass der Namensdienst und der Zugriff auf (bzw. das Finden von) Dateien in einer weiteren vertrauenswürdigen Komponente zu realisieren ist.

L. Hänel hat in seinem Großen Beleg [Hän07] einen ACPI Treiber auf das L4Env portiert. Mit seinem Design ist es möglich den Zugriff der Gerätetreiber auf ein bzw. ihr Gerät zu begrenzen. Dabei steuert eine Capability den Zugriff eines Treiber auf ein Gerät (und dessen Ressourcen).

Der *I/O-Guard* erhält die Policy in Form von Capabilities von einem Policy Manager und der *Driver Loader* setzt sie durch. Dieser Ansatz kann mit Hilfe eines Policy Managers MAC verwirklichen.

Bastei

Dieser Abschnitt basiert auf Auszügen aus dem Technischen Bericht über die Betriebssystemarchitektur Bastei von N. Feske und C. Helmuth [FH06]:

Das Policy Management für Ressourcen wird über Quotas realisiert, wobei die Prozesse in einer Baumstruktur angeordnet sind und der Parent-Prozess Entscheidungen für seine Kinder trifft. Um einen Dienst anzubieten nutzt der Parent Prozess die Ressourcen von seinen Child-Prozessen. Die Wurzel des Prozessbaumes ist core, das erste User-Level Programm, welches die Steuerung beim Systemstart übernimmt. Der init-Prozess wird von core gestartet und erhält alle verfügbaren Ressourcen. CORE das erste User-Level Programm bietet verschiedene Dienste an, unter anderem einen Verwalter für physischen Speicher (RAM), geschützte Adressräume für Threads (TASK) oder einen Verwalter von Capabilities (CAP). Die Kommunikation in Bastei wird über IPC, basierend auf C++ Streams, realisiert und die Zugriffsteuerung mittels Capabilities. CAP erstellt Capabilities und ermöglicht die Reservierung und Freigabe dieser systemweit einheitlichen ObjektIDs.

FLASK

SELinux basiert auf dem FLux Advanced Security Kernel und ist eine Erweiterung zu Linux, die den Einsatz von MAC ermöglicht. Das grundlegende Konzept in FLASK ist die Trennung von Entscheidung und Durchsetzung einer Policy. Das Modell des Securityservers und der Objektmanager beschreibt, wie der Zugriff der Subjekte auf die Objekte gesteuert: Der Securityserver entscheidet auf Anfrage, ob ein Zugriff legal ist, während die Objektmanager diese Entscheidungen durchsetzen. Über das *Access Vector Cache* Modul können alle Objektmanager auf bereits getroffene Entscheidungen zugreifen.

3 Entwurf

Dieses Kapitel zeigt den Entwurf und beschreibt die gefundene Lösung im Detail. Das Ziel der vorliegenden Arbeit ist es im **L4Env** eine Zugriffssteuerung basierend auf **Mandatory Access Control** einzuführen und somit Anwendungen (die nicht vertrauenswürdig sind) einer Sicherheitsstrategie zu unterwerfen. Über eine **einheitliche Schnittstelle** soll die Kommunikation zwischen Tasks reglementiert und der Verbrauch von Ressourcen begrenzt werden. Ziel ist es nicht, das L4Env von Grund auf anders zu realisieren, so dass die gesamte Zugriffkontrolle einem formalen Modell für MAC entspricht. Es soll vielmehr eine Möglichkeit geschaffen werden, für Tasks eine konkrete Policy anzugeben und das L4Env so zu instrumentieren, dass es in der Lage ist diese umzusetzen. Dabei übernimmt das L4Env die Rolle der vertrauenswürdigen Basis, die Anwendungen kontrolliert und gegebenenfalls einschränkt. Zu dieser Basis gehört der Namensdienst, der Dataspace Manager für physischen Speicher und der Loader. Die Basis wird instrumentiert, ist aber nicht in ihrer Kommunikation eingeschränkt und wird in der Ressourcennutzung nicht vom Manager beschränkt. Die kontrollierten Anwendungen erhalten bei ihrem Start Capabilities zur Kommunikation mit den Basisservern.

3.1 Designraum

Für Gestaltung der Kooperation zwischen dem Manager und dem L4Env gibt es mehrere Varianten. Sie unterscheiden sich in der Lokalisierung der Regelbasis, der Protokollierung aktuell genutzter Ressourcen und im Grad der Zwischenspeicherung bereits getroffener Entscheidungen.

Aktualisierungen des Verbrauchs durch die Basisserver und Regelveränderungen durch den Manager müssen der jeweils anderen Komponente rechtzeitig und zuverlässig zur Verfügung gestellt werden. Durch eine Speicherung bereits getroffener Entscheidungen können wiederholte Anfragen an den CM eingespart werden, aber wie bei jedem Cache muss die Konsistenz zum globalen Speicher gewahrt werden. Es bieten sich drei Implementierungsvarianten an:

- Regelbasis und Verbrauchsprotokoll im CM, keine Zwischenspeicherung
- Regelbasis und Verbrauchsprotokoll für CM und Basis zugänglich
- Regelbasis und Verbrauchsprotokoll verteilt auf die Basisserver

In dieser Arbeit wurde die erste Variante zur Implementierung ausgewählt, weil sich dadurch die Änderungen in den Basisservern auf eine Instrumentierung beschränken. Der Hauptteil des Implementierungsaufwands ist der Compartment Manager selbst. Der Verzicht auf eine Zwischenspeicherung von Entscheidungen (außer in IPCmon) wird sich auf die Geschwindigkeit auswirken. In welcher Größenordnung dies geschieht, werden primitive Benchmarks (als Ersatz für reale Anwendungen) zeigen.

3.2 Compartment Manager

Mit einem Policy Manager entsteht eine zentrale, unabhängige Komponente, die eine Compartmentdefinition erhält und dementsprechend Anwendungen startet, die einer Sicherheitspolitik unterliegen. Die Verwendung dieses Servers bleibt optional, die Basisserver arbeiten auch ohne CM. Ähnlich wie die Sicherheitsabfragen in der Tasklib spezifiziert werden können, bietet der CM die Möglichkeit entsprechend einer Sicherheitspolitik zu verfahren. Ein Compartment wird in diesem Fall als eine Taskmenge definiert, innerhalb der uneingeschränkt kommuniziert werden darf, die einheitliche Kommunikationsrechte zu anderen Compartments besitzt und die sich einen Pool von Ressourcen teilt.

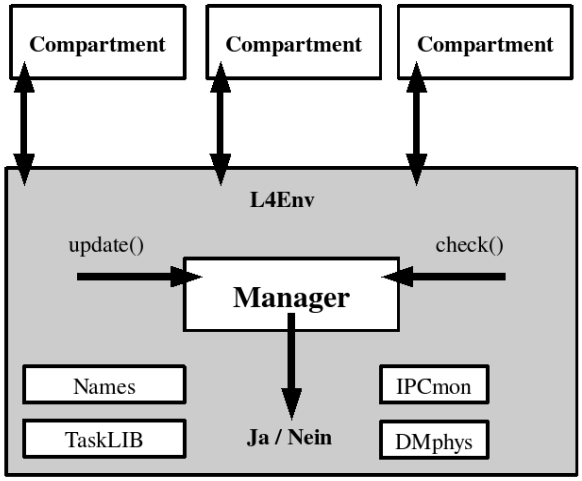

Abbildung 3.1: Design

Inspiriert durch das FLASK Konzept sollen auch im L4Env Entscheidung und Durchsetzung einer Sicherheitspolitik in separaten Komponenten realisiert werden. Dazu wird das L4Env um einen Server erweitert, der die Rolle des Securityservers übernimmt. Dieser Server trifft alle Entscheidungen betreffs der Zugriffskontrolle mit Hilfe einer Regelbasis, die der umzusetzenden Sicherheitspolitik entspricht. Die Basisserver des L4Env übernehmen die Rollen der Objektmanager, indem sie ihre Ressourcen und Kommunikationsbeziehungen so verwalten, wie es der Compartment Manager vorgibt.

Die Threads einer Task teilen sich einen Adressraum, über den sie kommunizieren können. Es reicht also aus, im CM mit einer Granulariät von Tasks zu operieren.

Eine Regelbasis (entsprechend der Sicherheitspolitik) beinhaltet die Zuordnung der Tasks zu genau einem Compartment, eine Auflistung der Kommunikationsmöglichkeiten für jedes Compartment und die Festlegung der Ressourcenaufteilung zwischen den Compartments.

Schnittstelle

Der Compartment Manager bietet den Basisservern eine Schnittstelle für Abfragen und protokolliert den Ressourcenverbrauch. Er besitzt eine Abbildung von TaskIDs auf Compartments und die entsprechenden Ressourcenmaxima der Compartments. Die Schnittstelle des CM besteht aus eingehenden Protokollmitteilungen und Abfragen, sowie ausgehenden Policyentscheidungen. Protokolliert werden aktuell genutzte Ressourcen und überprüft werden Kommunikationsanfragen sowie Ressourcenzuteilungen.

Konfiguration

Bevor eine Anwendung in einem Compartment gestartet werden kann, muss der Manager konfiguriert werden. So erhält er Informationen, wie die Anwendungen gegeneinander abgegrenzt werden. Zusätzlich muss er wissen, welche Ressourcen durch die Sicherheitsstrategie reglementiert werden sollen und wie die Ressourcen der Compartments begrenzt werden.

Mit diesen Vorgaben baut sich der CM die interne Datenstruktur auf und nutzt sie zur Laufzeit, um Entscheidungen für konkrete Zugriffsanfragen zu treffen. Der Manager startet die Anwendungen und ordnet sie genau einem Compartment zu. Weil die L4Env Server auf dem Namensraum der TaskIDs agieren, besitzt der CM eine eindeutige Zuordnung von TaskIDs auf Compartments. Zu jedem Compartment werden die vorgegebenen Maxima, sowie die aktuellen Werte der genutzten Ressourcen gespeichert und eine Liste anderer Compartments, mit denen die Kommunikation zulässig ist.

3.3 Instrumentierung der Basis

Begrenzt vorhandene Ressourcen im L4Env sind Speicher, Dateien, Geräte und Dienstnamen. Kommunikation findet über IPC und gemeinsam genutzten Speicher statt. Exemplarisch für alle Ressourcen werden die Taskanzahl, der physische Speicher und die Interprozesskommunikation einer einheitlichen Zugriffskontrolle unterworfen.

Unabhängig von der Realisierung muss zunächst ermittelt werden, in welchen Situationen dem einzelnen L4Env Server die Entscheidung für oder gegen eine konkrete Aktion abgenommen werden soll. Eine Aktion kann die Nutzung oder Freigabe einer Ressource sein oder das Zustandekommen einer Kommunikation. Für diese Situationen müssen die L4Env Server instrumentiert werden, so dass sie die Sicherheitspolitik beachten bevor sie eine entsprechende Aktion ausführen.

Tasks

Die Taskbibliothek ist bereits mit Sicherheitsabfragen instrumentiert, diese müssen lediglich mit dem Compartment Manager verbunden werden. So kann vor der Vergabe einer TaskID

geprüft werden, ob dem Compartment noch weitere Tasks zustehen, und die Reservierung bzw. Freigabe jeder TaskID wird an den Manager übermittelt.

Geräte

Eine Zugriffskontrolle auf Geräte die über Capabilities realisiert wird, ist Teil der Belegarbeit „ACPI for L4Env“ [Hän07]. Der Zugriff wird reglementiert, indem Gerätetreiber nur mit entsprechender Capability auf die Ressourcen eines Gerätes zugreifen können. Der Compartment Manager kann durch Übergabe der Capabilities an den I/O-Guard die Policy bestimmen.

Dateien

Das virtuelle Dateisystem und die Fileserver des L4Envs wurden ohne spezielle Zugriffkontrollen entwickelt. Der Manager kann aber mit dem vertrauenswürdigen Dateisystem [Wei06] verbunden werden. Je nach dem, ob die Compartments hierarchisch oder parallel organisiert sind, kann der Manager Capabilities für die Pfade eines hierarchischen File Systems erstellen oder jedem Compartment einen Teil des Dateisystems zur Verfügung stellen.

Kommunikation

IPCmon ist die prototypische Implementierung eines Capability Faulthandlers für Kommunikation via IPC. Es kann die Kommunikation zwischen Tasks reglementieren und speichert die erlaubten oder verbotenen Kommunikationskanäle. Da dieser Server schon existiert, bietet es sich an, ihn entsprechend anzupassen und die Capabilityverwaltung nicht als Teil des CM zu realisieren. Für die Realisierung von MAC für L4Env wird die Konfiguration von IPCmon in die Verantwortung des CM übergeben. Bisher gab es zwei Konfigurations- und eine Abfragefunktion.

- allow(Task A, Task B)
- deny(Task A, Task B)
- query(Task A, Task B)

Physischer Speicher

Der DS Manager für physischen Speicher soll unter zwei Gesichtspunkten instrumentiert werden. Zum einen, soll die gemeinsame Nutzung eines Dataspaces nur unter der Voraussetzung erlaubt werden, dass die beteiligten Tasks miteinander kommunizieren dürfen. Zum anderen muss eine Task in allen Situationen, in denen sie Speicher verwendet unter dem Quota ihres Compartments bleiben, deswegen erhält der CM Mitteilungen zu der aktuell genutzten Speichermenge.

Namen

Bestimmte Dienstnamen sind für die Basis reserviert. Die Taskanzahl kann einfach vom CM beschränkt werden, die Organisation der Namensreservierung obliegt dem Namensdienst. Bis

jetzt finden die Tasks den Namensdienst, indem sie die Roottask fragen, deren ID immer dieselbe ist. Dieser initiale Ressourcen Manager soll mit kontrollierten Anwendungen jedoch nicht kommunizieren, nur mit den Basisservern. Für eine Unabhängigkeit von Roottask kann man die ID des Nameservers in der L4Env Infopage der Task hinterlegen. Dafür muss die Task allerdings vom Loader gestartet werden. Zur Nutzung lokaler Namen im jeweiligen Compartment müsste der aktuelle Nameserver des L4Envs anders implementiert sein. Ein Namensdienst der als Teil des CM realisiert wird, soll bei der Registrierung die ID der Compartments mit einbeziehen.

4 Implementierung

Hier wird die Instrumentierung der Basisserver und die Realisierung des Compartment Managers im Detail beschrieben. Abschließend folgt ein Überblick über das Maß der Veränderungen und die Größe des Compartment Managers.

4.1 Basis

Die Funktionsweise der Basis wurde bereits im 2. Kapitel beschrieben, während die zu instrumentierenden Server im 3. Kapitel aufgezählt werden. Die konkrete Instrumentierung erfolgt durch das Einfügen der Funktionen `check()` und `update()`, wodurch die relevanten Aktionen von der Entscheidungen des Managers abhängig gemacht werden.

Capability Faulthandler

IPCmon erfährt vom Manager mit welchen anderen Tasks eine bestimmte Task kommunizieren darf. Sobald ein Capability Fault auftritt erfragt es die Kommunikationsbeziehung der zwei Tasks beim CM und übernimmt die Entscheidung in seine interne Datenstruktur.

Dataspace Manager

Der DS Manager für physischen Speicher (DMphys) wurde so angepasst, dass er beim Öffnen, Kopieren und gemeinsamen Benutzen eines Dataspaces zunächst Rücksprache mit dem CM hält. Er erhält ein „Ja“, wenn das Maximum des Compartments noch nicht erreicht ist. Die jeweilige Operation wird auf dem Dataspace ausgeführt und der CM erhält eine Mitteilung zur Aktualisierung der aktuell genutzten Speichermenge.

Wenn ein Dataspace an eine andere Task transferiert werden soll, also den Eigentümer wechselt, muss der DSM zwei Dinge überprüfen:

- Dürfen alter und neuer Eigentümer kommunizieren?
- Bleibt der neue Eigentümer mit diesem Dataspace unter dem Speicherlimit?

Sind die beiden Voraussetzungen erfüllt sind, wird der Dataspace übergeben. Dem CM wird mitgeteilt, welche Compartments am Transfer beteiligt sind und wie groß der DS ist. Beim Schließen eines Dataspaces ist wird dem CM nur mitgeteilt, welche Task wieviel Speicher zurück gibt.

Weitere Anpassungen

Die Roottask unterliegt nicht der Kontrolle des CM und soll auch nicht mit den kontrollierten Anwendungen kommunizieren. Jede Anwendung bekommt aber die TaskID des Nameservers

von Roottask, dies läßt sich bei Anwendungen die vom Loader gestartet werden leicht umgehen. Sie erhalten eine L4Env Infopage, in der grundlegende Information gespeichert werden, wie die ID des Pagers oder die der Parent-Task. Hier wird jetzt auch die ID des zuständigen Nameservers hinterlegt. Wenn eine Task sich beim Nameserver anmelden will, wird überprüft ob sie eine L4Env Infopage besitzt. Falls nicht wurde sie nicht vom Loader gestartet und ist folglich keine der kontrollierten Anwendungen.

4.2 Manager

Es gibt drei Abfragefunktionen, mit denen die L4Env Server beim CM überprüfen können, ob der betroffenen Task eine bestimmte Aktion erlaubt werden darf. Für den Transfer eines Dataspaces existiert eine kombinierte Abfragefunktion, weil dabei sowohl Ressourcenlimits, als auch Kommunikationsbeziehungen beachtet werden müssen.

Mit Hilfe der Aktualisierungen übermitteln die Basisserver dem CM Informationen über die erfolgte Aktion. Somit kann dieser die Menge an verwendeten Ressourcen protokollieren.

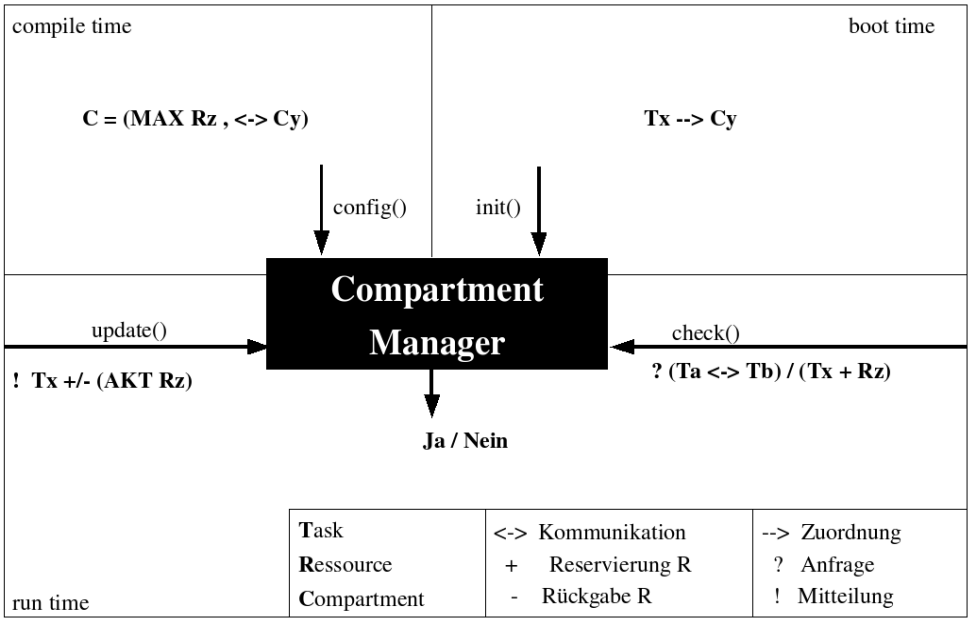

Abbildung 4.1: Implementierung

Schnittstelle

Alle Abfragen und Aktualisierungen beginnen mit der Übersetzung der TaskID in eine Compartment ID und enden mit dem Rückgabewert Null oder Eins, entsprechend der Aussagen Ja oder Nein. Beim Aktualisierungen wird eine 1 nur im Fehlerfall zurückgegeben.

check_com (src, dest) Diese Abfrage wird gestellt, wenn eine Kommunikation zustande kommen soll. Das Zielcompartment wird in der Whitelist des Quellcompartments gesucht, an erster Stelle steht dort immer die eigene Compartment ID.

check_ram (caller, amount) Für eine Task wird ermittelt, ob ihr Compartment noch weiteren Arbeitsspeicher in der genannten Größe reservieren darf.

check_task (caller, amount) Eine Task erfragt, ob das Reservieren von weiteren TaskIDs erlaubt ist.

check_transfer_ram (caller, amount, nowner) Ein Kombination aus der Kommunikationsabfrage und Speicheranfrage, speziell für den Transfer eines Dataspaces.

update_task (caller, amount, updown) Eine Aktualisierung der Zuordnung von TaskIDs auf Compartments und eine Änderung der Taskanzahl des Compartments.

update_ram (caller, amount, updown) Mit dieser Aktualisierung wird die Verteilung des Arbeitsspeichers auf Compartments protokolliert.

Eine Task ist entweder ein Basisserver oder eine kontrollierte Anwendung. In der Bibliothek des Compartment Managers wird zu Beginn aller Funktionsaufrufe überprüft, ob eine Task zur Basis gehört und somit nicht reglementiert werden muss.

Konfiguration

Mit einem Perl Skript wird eingelesen, wieviele Compartments es gibt, welche Grenzen ihnen im Ressourcenverbrauch gesetzt sind und mit welchen anderen Compartments sie kommunizieren dürfen (Abbildung 4.1). Das Skript generiert daraus eine C-Datei, die zur Initialisierung der internen Datenstruktur des Compartment Managers verwendet wird.

Compartment	Tasks	RAM	Whitelist
C1	3	50 MB	C2
C2	4	50 MB	C1
C3	15	50 MB	
C4	7	56 MB	C1, C2

Tabelle 4.1: Definition der Compartments

4.3 SLOC

Die Größe des CM liegt im Bereich der anderen Basisserver, die Instrumentierung kostet zusätzlich zwischen 12 und 60 Zeilen. DMphys hat die meisten zu instrumentierenden Funktionen,

während im Loader eine konstante ID für den Nameserver (5.0) hinterlegt wurde und die Callbacks der Tasklib initialisiert werden. Die Names-Bibliothek wurde kopiert und für kontrollierte Anwendungen angepasst.

L4 Env		**Instrumentierung**
Names	802	+ 60
Loader	6795	+ 12
IPCmon	446	+ 28
DMphys	5936	+ 54
Manager	1352	$\sum$ 154

Tabelle 4.2: Größe des CMs und Summe der Instrumentierungen

5 Leistungsbewertung

Wieviel der Gewinn an Zuverlässigkeit bzw. Sicherheit kostet, wird mit Hilfe von Geschwindigkeitsmessungen ermittelt. Dazu habe ich kleine Benchmarks geschrieben, welche die Dauer der instrumentierten Funktionen im L4Env messen.

5.1 Messungen

Einfach strukturierte Tasks sollen zeigen, wieviel der Einsatz von Compartments kostet. Zunächst wird ermittelt, welche Ausführungszeiten die instrumentierten Funktionen der Basisserver vor der Anpassung an den Compartment Manager haben. Dabei werden die Funktionen in 5.1 von Tasks aufgerufen, die weder der Kontrolle des CMs noch der des Capability Faulthandlers unterliegen. Im Vergleich dazu wird die Dauer der einzelnen Funktionsaufrufe gemessen, wenn die Tasks einem Compartment angehören.

		Zyklen(+)	**Prozent(+)**
IPC Call	1.	10.036	1463,0
	2.-100.	5	0,7
	100 x 100	105	15,3
TaskID	allocate	6.860	72,3
	free	7.311	117,0
Dataspace	open	7.016	59,9
	close	4.656	84,1
	share	5.003	97,0
	copy	8.803	40,8
	– resize	6.215	59,2
	+resize	8.821	76,0
	transfer	11.011	309,2

Tabelle 5.1: Zeitlicher Mehraufwand gemessen in CPU Zyklen

Durch die zusätzlichen Abfragen und Aktualisierungsnachrichten an den CM ist für die Funktionsaufrufe unter „kontrollierten“ Bedingungen eine Verzögerung zu erwarten. Diese Zeiten zeigen, wieviel der Verzicht auf eine Zwischenspeicherung kostet.

Die Messungen wurden auf einem Testrechner mit folgenden Komponenten durchgeführt:

- AMD Duron 1,3 GHz CPU
- 256 MB RAM

- 64 KB L1 Data Cache / Instruction Cache
- 64 KB L2 Cache

Bei der Messung des IPC Calls wird im kontrollierten Szenario zwischen dem ersten IPC Call und den Nachfolgenden unterschieden, um genau den zeitlichen Aufwand zu erhalten der entsteht, wenn eine Task die erforderliche Capability nicht besitzt. Dazu wurde nach jedem 100. IPC Call die Kommunikation untersagt, so dass der Capability Faulthandler entsprechend seiner Instrumentierung, die Legalität dieser Kommunikation erst beim CM abfragen muss.

5.2 Bewertung

Bis auf eine Ausnahme zeigen die Werte deutlich, dass diese Implementierung von Kontrollmechanismen das System erheblich verlangsamt. Die Entwurfsentscheidung, den CM an jeder einzelnen Aktion zu beteiligen (um die Veränderung des L4Envs und den Implementierungsaufwand des CM möglichst gering zu halten), erweist sich als nachteilig, da im Mittel ein zeitlicher Mehraufwand von ca. 85% entsteht. Dieser Wert darf aber nicht mit der Verlangsamung des Gesamtsystems gleichgesetzt werden, da sich diese Werte nur auf einzelne Funktionen beziehen. Für eine durchschnittliche Anwendung ist weder die Häufigkeit der einzelnen Funktionsaufrufe ermittelt worden; Noch ist das Verhältnis bekannt zwischen den hier instrumentierten Funktionen und all denen die von einer Anwendung benutzt werden. Trotzdem erscheinen die Differenzen recht groß und stellen somit den Nutzen der vorliegenden Implementierung in Frage.

Einen Hinweis zur Verbesserung dieser Zeiten gibt der verhältnismäßig niedrige Messwert des IPC Calls. Der zeitliche Mehraufwand beträgt hier ca. 15% und hebt sich damit deutlich von den übrigen Messwerten ab, die zwischen 41% und 310% liegen. Es ist jedoch zu beachten, dass in diesem Fall der günstigere von zwei Fällen als Beispiel dient. Wenn die Kommunikation zwischen zwei Tasks erlaubt ist, tritt nur beim ersten Versuch eine IPC zu schicken ein Capability Fault auf, der eine Verzögerung um 1.463 % bewirkt. Ist die Kommunikation hingegen verboten, tritt diese Verzögerung bei jedem Versuch auf. Dies spricht für eine Verschmelzung des Capability Faulthandlers mit dem Compartment Manager.

Für den Einsatz regelbasierter Zugriffssteuerung, die weniger Zeit kostet, scheint eine Instrumentierung, die lediglich aus Abfragen und Mitteilungen besteht nicht geeignet. Die Zwischenspeicherung bereits getroffener Entscheidungen, wie es IPCmon realisiert ist effizienter.

5.3 Offene Probleme

Die Konfiguration des CM ist statisch und wird spätestens zur Bootzeit festgelegt. Neue Compartments erfordern einen Neustart, das soll flexibler gestaltet werden. Die vorgegebene Policy ist ebenfalls statisch, man kann zwar die Kommunikation zu einem Compartment unterbinden, oder einzelne Maxima ändern. Aber das ist nicht akzeptabel implementiert.

Welche Anforderungen haben durchschnittliche zu kontrollierende Anwendungen an das L4Env im Hinblick auf Geschwindigkeit und welchen Prozentsatz machen die instrumentierten Funktionen im Vergleich zu allen genutzten Funktionen aus ?

6 Zusammenfassung und Ausblick

Im L4Env kann jetzt MAC eingesetzt werden, das heißt eine Anwendung oder eine Taskmenge (Compartment) kann in ihrem Zugriff auf Ressourcen und in ihrer Kommunikation zu anderen Compartments beschränkt werden. Begrenzbare Ressourcen sind die Taskanzahl und die Menge des verwendeten Arbeitsspeichers.

Mit einem weiteren vertrauenswürdigen Server, dem Compartment Manager unterstützt das L4Env jetzt eine einheitliche Beschränkung von Anwendungen. Diese Anwendungen kommunizieren mit den Basisservern des L4Envs wie vorher, es ist ihnen jedoch verboten die Roottask zu kontaktieren. Die vom Compartment Manager vorgegebene Policy wird von der Taskbibliothek, dem Capability Faulthandler und dem physischen Dataspace Manager durchgesetzt. Die Zugriffssteuerung auf Geräte und Dateien wurde nicht implementiert.

Ausblick

Der Nachfolger des L4Envs, das L4 Runtime Environment (L4RE), ist noch in der Entwicklung. Es wird Capabilities als Mechanismen zur Zugriffskontrolle einsetzen und lokale Namensräume verwalten können. Damit kann eine einheitliche Zugriffssteuerung besser realisiert werden, als im L4Env.

Für eine Anpassung des Compartment Managers an das L4RE, ist eine Analyse der Anforderungen auf beiden Seiten notwendig, gerade in Bezug auf Geräte und Dateien. Ich vermute, dass MAC im L4RE mit meiner Implementierung nicht effizient realisiert wird.

Ein ganzheitliches Konzept, dass die einzelnen Anforderungen der Server und Bibliotheken des L4RE berücksichtigt, erfordert ein anderes Design. Meine Lösung, ein zentraler Manager, der keine Capabilities verteilt sondern auf Anfrage Entscheidungen bekannt gibt, kann optimiert werden indem die Schnittstelle reduziert wird, d.h. die Abfragen und Mitteilungen können kompakter gestaltet werden.

Abbildungsverzeichnis

3.1 Design 10

4.1 Implementierung 16

Glossar

DAC Discretionary Acces Control

MAC Mandatory Access Control

MLS Multi Level Security

L4Env L4Environment

IPC Inter Process Communication

Task Adressraum für mindestens einen Thread

Capability objektorientiertes Zugriffrecht

Policy Strategie, Richtlinie, Regelwerk

Compartment Sandbox, Taskmenge

PoLP Principle of Least Privilege

Literaturverzeichnis

[ALE+01] Mohit Aron, Jochen Liedtke, Kevin Elphinstone, Yoonho Park, Trent Jaeger, and Luke Deller. The sawmill framework for virtual memory diversity. In *ACSAC '01: Proceedings of the 6th Australasian conference on Computer systems architecture*, 2001. 5

[Bis03] Matt Bishop. *Computer Security - Art and Science*. 2003. 4, 5

[DRO] Dresden Real-Time Operating System. `http://os.inf.tu-dresden.de/drops/`. 5

[FH06] Norman Feske and Christian Helmuth. Design of the Bastei OS architecture. Technical report, TU Dresden, 2006. 7

[Fia] L4 Microkernel Fiasco. `http://os.inf.tu-dresden.de/fiasco/`. 5

[Hän07] Lukas Hänel. ACPI for L4Env, 2007. 6, 12

[Kau06] Bernhard Kauer. L4Env Core Interface, 2006. 5

[Sha99] Jonathan Shapiro. What is a capability, anyway?, 1999. 4

[Wei06] Carsten Weinhold. Design and Implementation of a Trustworthy File System for L4, 2006. 6, 12