I/O - Flexpages

Frank Réolon
Universität des Saarlandes
Institut für Rechnerarchitektur

28. Oktober 2004

Inhaltsverzeichnis

1 Basisinformationen **2**
1.1 L4 - Mikrokern (die 2.Generation) . . . 2
1.2 I/O - Ports . . . 2
1.3 I/O - Space . . . 3
1.4 Sigma 0 . . . 3
1.5 Pager . . . 3

2 Speicherverwaltung mit Flexpages **3**
2.1 Definition von Flexpages . . . 3
2.2 Operationen auf Flexpages . . . 4
2.3 Beispiel . . . 4

3 Übergang zu I/O - Flexpages **6**
3.1 I/O - Instruction . . . 6
3.2 I/O - Schutzmechanismen . . . 6
3.3 Einführung der I/O - Flexpages . . . 8
3.4 Funktionen mit I/O - Flexpages . . . 8
3.5 Sigma 0 . . . 9
3.6 RPC Protokoll für general protection exceptions . . . 9
3.7 Legacy Support . . . 11

4 Realisierung der I/O - Flexpages **11**
4.1 I/O - Privilege - Levels . . . 11
4.2 I/O-Permission-Bitmap und das Mapping - Konzept . . . 11
4.3 I/O-Mapping-Datenbank . . . 12
4.3.1 Datenstruktur eines I/O-Mapping-Nodes . . . 12
4.3.2 Mapping Algorithmen . . . 12

5 Schlusswort **14**

1 Basisinformationen

In diesem ersten Abschnitt möchte ich einige Basisinformationen geben. Zunächst werde ich kurz etwas zu dem Mikrokern L4 sagen und erläutern, wo die Flexpages und I/O - Flexpages einzuordnen sind. Danach werde ich einige Begriffe und Objekte einführen, welche für diese Arbeit Relevanz haben.

1.1 L4 - Mikrokern (die 2.Generation)

Wie der Name schon sagt, geht es bei Mikrokernen darum, den Kern zu minimieren, d.h. viele Anwendungen die vorher im Kernel-Mode liefen in den User-Mode zu verbannen. Vom Ansatz her verfolgen die erste und zweite Generation von Mikrokernen verschiedene Ziele. Bei der ersten Generation hatte die Flexibilität und die Transparenz auf verteilten Systemen Vorrang. Bei der zweiten Generation stehen die Trennung von Mechanismus und Strategie, sowie die Minimalität im Vordergrund.
Dem L4 - Mikrokern obliegt zusätzliche die Aufgabe das Geschwindigkeitsproblem der ersten Generation zu lösen. Somit hat der L4 - Kern nur noch folgende Aufgaben zu bewältigen:

- **Schnelle Inter-Prozess-Komunikation zur Verfügung stellen**
  Die schnellere Prozess-Kommunikation wurde durch Optimieren der IPC-Messages ermöglicht.
- **Bereitstellen von flexiblen Konzepten um Hauptspeicher zu managen**
  Die Flexpages und die dazugehörigen Funktionen sind hier einzuordnen.
- **Kontrolle und Schutz der unterliegenden Hardware**
  Aus dem Konzept der Flexpages wurde von Jochen Liedtke ein ähnliches Konzept für Hardwarekontrolle erarbeitet, nämlich die I/O - Flexpages. Die I/O - Flexpages sind der Kontrolle und dem Schutz der Hardware zuzuordnen.

1.2 I/O - Ports

I/O - Ports sind Adressen, die sich auf die Peripheriegeräte beziehen. Jedes solches Peripheriegerät, das mit dem I/O - bus verbunden ist, kann durch eine Menge von I/O - Ports angesprochen werden. Der I/O - bus verbindet den Prozessor mit den Peripheriegeräten. Was allgemein als I/O - bus bezeichnet wird, besteht eigentlich aus drei speziellen buses. Der data bus besteht aus einer Anzahl von Kabeln, die Daten parallel übertragen. Der address bus, der auch aus einer Anzahl von Kabeln besteht, überträgt Adressen parallel. Und der control bus sendet Kontrollinformationen zu dem angeschlossenen Schaltkreisen.

1.3 I/O - Space

In der x86-Architektur handelt es sich bei dem I/O - Space um einen separater Speicherbereich neben dem Hauptspeicher. Jedes Peripheriegerät hat eigene Register, in welche Daten abgespeichert werden können. Der I/O - Space fasst diese Register, welche auf der Umgebungshardware liegen, zusammen.

1.4 Sigma 0

Sigma 0 ist im Besitz des gesamten verfügbaren Speichers, also I/O - Space und Hauptspeicher, bei Systemstart.

1.5 Pager

Pager sind auf Benutzerebene laufende Prozesse, die sich um die Speicherverwaltung kümmern. Wie sich dies genau vollzieht, wird im Laufe dieser Arbeit gezeigt.

2 Speicherverwaltung mit Flexpages

In diesem Abschnitt werde ich nun kurz darauf eingehen, wie verschiedenen Anwendungen Speicher zur Verfügung gestellt wird. Wenn ich in diesem Abschnitt von Speicher rede, meine ich Hauptspeicher bzw. virtueller Speicher.

Jede auf Benutzerebene laufende Anwendung hat eigenen virtuellen Speicher. Virtueller Speicher beinhaltet alle Speicherplätze des Hauptspeichers (physikalischer Speicher) als auch die des Sekundärspeichers (Festplatte). Ein Grund hierfür ist, dass viele Programme zu groß für den zur Verfügung stehenden Hauptspeicher sind. Die Anwendung kann also immer nur einen Teil des Programms in den Hauptspeicher einlagern. Dazu muss sie aber zuerst physikalischen Speicher in ihren Adressraum gemappt bekommen. Der Adressraum der Anwendung ist die Menge aller verwendbaren Adressen und den an diesen Adressen abgelegten Speicherinhalten. Um der Anwendung Teile des physikalischen Speichers zumappen zu können ist ein Objekt erforderlich, das Bereiche eines Adressraums beschreiben kann.

2.1 Definition von Flexpages

Eine Flexpage beschreibt einen Speicherbereich in einem Adressraum.
Die Flexpage hat eine gegebene Basisadresse und eine dazugehörige Größe.
Basisadresse und Größe bestimmen den beschriebenen Bereich genau.
Die Pager haben die Aufgabe, Anwendungen Speicher zur Verfügung zu stellen.
Dies tun sie mit Hilfe des Flexpage-Mechanismus.

Bemerkung: Es muss nicht unbedingt ein Pager sein, der einen Bereich seines Adressraums an eine andere Anwendung weitergibt, sondern jede beliebige Anwendung kann anderen Anwendungen Speicher zur Verfügung stellen.

2.2 Operationen auf Flexpages

1. **map:**
   An dieser Funktion sind zwei Anwendungen beteiligt, der Sender und der Empfänger. Beide Anwendungen haben einen Bereich in ihrem jeweiligen Adressraum mittels Flexpage spezifiziert. Nun mappt der Sender seinen durch die Flexpage beschriebenen Bereich dem Empfänger zu. Der Bereich wird an die Stelle gemappt, an die der Empfänger ihn benötigt. Dies wird durch die Flexpage des Empfängers spezifiziert.

2. **grant:**
   Bei der grant-Funktion geht fast dasselbe vor sich wie bei map. Wieder sind ein Sender und ein Empfänger beteiligt, sowie jeweils eine Flexpage. Der einzige Unterschied besteht darin, dass der Sender seine Zugriffsrechte auf den Bereich, der von seiner Flexpage beschrieben wird, verliert.

3. **unmap:**
   Wie der Name schon sagt, wird bei unmap der Bereich einer Anwendung, der durch eine Flexpage beschrieben wird, aus dem Zugriffsbereich der Anwendung gelöscht.

2.3 Beispiel

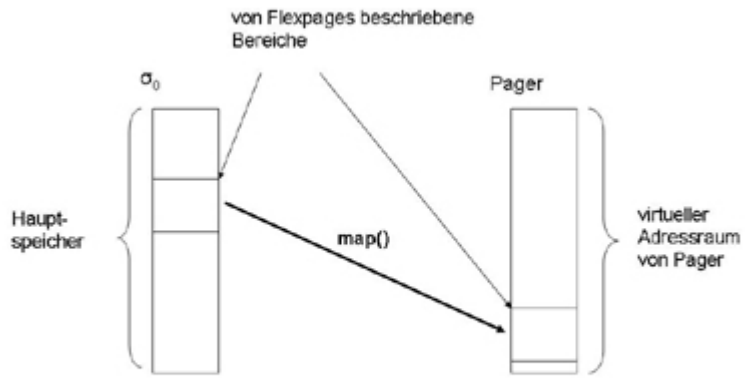

Abbildung 1: Beispiel

Abbildung 1 zeigt den Adressraum zweier Anwendungen, den von Sigma 0 und den eines Pagers. Das besondere an Sigma 0 ist, dass es sich bei des-

sen Adressraum um den physikalischen Speicher, bei dem Pager um virtuellen Speicher handelt. Nun wird die map-Funktion auf die beiden Anwendungen angewandt. Hierzu wird ein Bereich im Adressraum von Sigma 0, sowie ein Bereich im Adressraum des Pagers, mittels Flexpage spezifiziert. Der Pager bekommt nun den spezifizierten Bereich von Sigma 0 zugemappt (vergleiche hierzu die Beschreibung der map-Funktion auf Seite 4).

3 Übergang zu I/O - Flexpages

Im Folgenden gehe ich näher auf die I/O - Flexpages ein. Dafür werde ich im ersten Teil dieses Abschnittes zunächst allgemein etwas über die Nutzung der Hardware sagen. Es wird gezeigt, wie der Prozessor mit der Hardware kommuniziert und welche Schutzmechanismen zur Verfügung gestellt werden. Danach werde ich konkret die I/O - Flexpage mit ihren Funktionen einführen und Vorgänge beschreiben, in denen sie zum Einsatz kommen.

3.1 I/O - Instruction

Wie bereits erwähnt, ist I/O - Space in der x86-Architektur ein separater und unabhängiger Speicherbereich neben dem Hauptspeicher. Der Prozessor kann Daten von einem I/O - Ports lesen oder in einen I/O - Port schreiben. Hierfür gibt es im wesentlichen folgende I/O - Instructions:

in(port)
out(port)

Wird eine solche Instruction aufgerufen, benutzt der Prozessor den address bus, um den benötigten I/O - Port auszuwählen. Daraufhin nutzt er den data bus um Daten zwischen seinen Registern und dem Port zu übertragen. Jeder Port ist individuell ansprechbar und wird durch ein 16bit Wort gekennzeichnet. I/O - Ports sind immer nur an der selben Port-Adresse erreichbar. Was das konkret für unsere I/O - Flexpage bedeutet, wird später deutlich.

3.2 I/O - Schutzmechanismen

Für Hardware werden zwei Schutzmechanismen zur Verfügung gestellt. Die I/O - Privilege Levels (IOPL) und die I/O - Permission Bitmap (IOPBM).

Schutz via I/O - Privilege Levels

Jede Anwendung hat einen eigenen current privilege level (CPL). Der CPL ist eine Zahl zwischen 0 und 3. Auf Benutzerebene laufende Prozesse haben einen CPL von 3, auf Kernelebene laufende Anwendungen haben einen CPL von 0. Zusätzlich wird jeder Anwendung ein IOPL zugeordnet, welche die Privilegiertheit beschreibt auf die Hardwareumgebung zuzugreifen. Bei dem IOPL handelt es sich ebenfalls um eine Zahl zwischen 0 und 3. Ist der CPL einer Anwendung kleiner oder gleich dem IOPL der Anwendung, so wird Zugriff auf alle I/O - Ports gewährt, ohne sonstige Checks durchführen zu müssen. Anderenfalls muss die IOPBM der Anwendung überprüft werden.

Schutz via I/O - Permission Bitmap

Die IOPBM gewährt oder verweigert Zugriff auf einzelne Ports. Sie enthält,

wie der Name schon vermuten lässt, eine Reihe von Bits, wobei jedes einzelne Bit sich auf einen I/O - Port bezieht. Ist das Bit gesetzt, bedeutet dies, dass der Zugriff auf den I/O - Port blockiert ist. Ist es nicht gesetzt, ist der Zugriff frei für diesen I/O - Port. Wenn der Zugriff auf einen bestimmten Port blockiert ist, wird eine general protection exception (GP) geworfen.

In der folgenden Abbildung stelle ich nun das gerade beschriebene Vorgehen noch eimal visuell dar. Zuerst versucht eine Anwendung mittels einer I/O - Instruction auf einen Port zuzugreifen. Dann werden die zwei oben beschriebenen Schutzmechanismen ausgeführt. Nun wird entweder eine GP geworfen oder die Arbeit der Anwendung kann fortgesetzt werden.

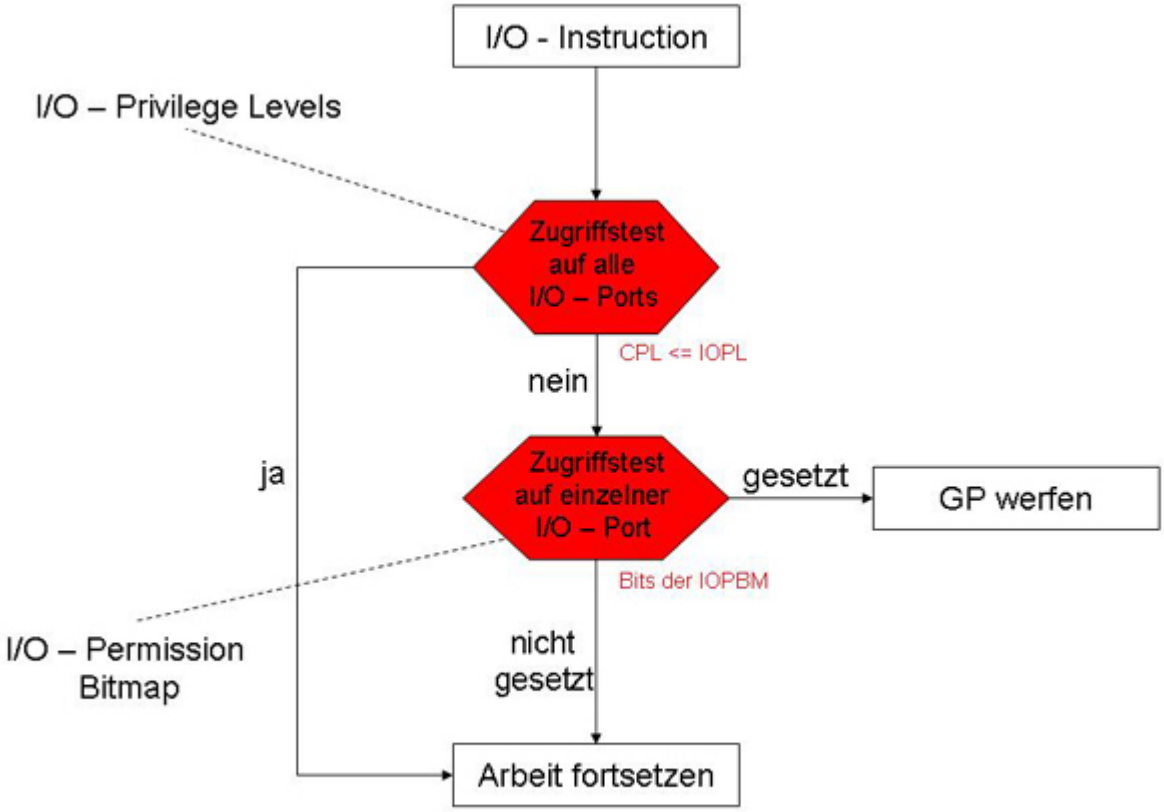

Abbildung 2: I/O - Schutzmechanismen

3.3 Einführung der I/O - Flexpages

Die Idee I/O - Flexpages einzuführen besteht darin eine Analogie zur Speicherverwaltung herzustellen. Man möchte also die Funktionalitäten der Flexpages auf I/O - Space erweitern. Flexpages, die sich auf I/O - Space beziehen, werden I/O - Flexpages genannt.
Woher sollen die bereits implementierten Funktionen (map, grant, unmap) aber wissen, ob sie sich auf Hauptspeicher oder I/O - Space beziehen. Eine mögliche Lösung wäre einfach den Funktionen einen Identifier als zusätzlichen Parameter mitzugeben, welcher bestimmt, ob nun Hauptspeicher oder I/O - Space angesprochen werden soll. Das bedeutet aber, dass die Funktionen neu implementiert werden müssen. Dies führt wiederum dazu, dass ein neues L4 API (Application Programming Interface) zu realisieren ist. Ein API ist die Schnittstelle, die ein Betriebssystem oder auch ein anderes Softwaresystem anderen Programmen zur Verfügung stellt. Es definiert nur die Verwendung der Schnittstellen, nicht aber deren Realisierung. Zusätzlich wäre es noch nötig gewesen bereits vorhandene Software anzupassen.
Die Lösung, welche vorgezogen wurde, bestand darin, einen gewissen Bereich des Kernel - Space dafür zu missbrauchen um anzudeuten, dass Operationen auf I/O - Ports angewendet werden sollen.

Definition: Eine I/O - Flexpage ist ein Objekt mit bestimmter Größe,
welches eine Auswahl von I/O - Ports beschreibt.
Die I/O - Flexpage hat eine Basisadresse p und eine Größe,
welche durch eine 2er Potenz beschrieben wird.
Die Basisadresse und die Größe bestimmen den beschriebenen Bereich.

3.4 Funktionen mit I/O - Flexpages

Wie bereits in der Einführung der I/O - Flexpages erwähnt, gibt es keine besonders großen Unterschiede zwischen Flexpages und I/O - Flexpages. Bei I/O - Flexpages müssen aber dennoch einige Besonderheiten beachtet werden:
Ports können im Gegensatz zu Speicher nicht virtualisiert werden. Dies bedeutet, dass sie immer nur über die selbe Portadresse angesprochen werden können. Die daraus resultierende Folge ist, dass die I/O - Flexpage des Empfängers immer mindestens den Bereich enthalten muss, der durch die I/O - Flexpage des Senders spezifiziert wurde. Bei unsinnigen Bereichsangaben wird der Vorgang einfach abgebrochen. Normalerweise gibt man als I/O - Flexpage des Empfängers einfach den gesamten I/O - Adressraum an, so kann es nicht zu Problemen kommen.

Zur Beschreibung von **map**, **grant** und **unmap** vergleiche Seite 4 *Operationen mit Flexpages*. Der einzige Unterschied liegt darin, dass der beschriebene Bereich von Flexpages sich auf Speicher bezieht, bei I/O - Flexpages bezieht sich der beschriebene Bereich auf I/O - Ports.

3.5 Sigma 0

Sigma 0 verteilt I/O - Ports in der selben Art und Weise wie virtuellen Speicher. Die ersten Anwendungen die I/O - Ports anfordern bekommen diese auch. Typischerweise sind das die initialen Pager. Von dem Moment an, an dem Sigma 0 die I/O - Ports weiter gegeben hat, kümmert er sich nicht mehr um die Verwaltung. Nun sind die initialen Pager dafür zuständig. Weitere Anfragen an Sigma 0 werden immer abgelehnt.

3.6 RPC Protokoll für general protection exceptions

Zunächst beschreibe ich was passiert, wenn eine Zugriffsverletzung auf I/O - Ports vollzogen wird. Da dies nicht besonders effizient ist, wird das RPC Protokoll eingesetzt. Das RPC Protokoll ermöglicht eine gebündelte Anfrage, die den teuren Kommunikationsaufwand mit dem Pager dramatisch senkt. Das Vorgehen bei dem RPC - Protokoll für GP gebe ich nach dem Erklären des Vorgehens bei Zugriffsverletzung.

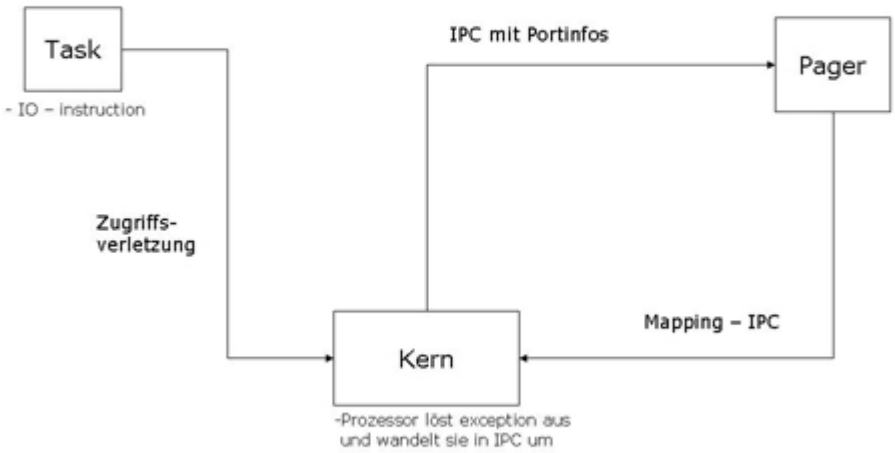

Abbildung 3: Vorgang bei Zugriffsverletzung

Vorgang bei Zugriffsverletzung

Als erstes versucht eine Task (Anwendung) mittels einer I/O - Instruction auf einen I/O - Port zuzugreifen. Aus diesem Grund muss nun der IOPL mit dem CPL der Anwendung überprüft werden. In diesem Fall schlägt das fehl. Jetzt

wird das Bit der IOPBM überprüft. Dieses ist in unserem Fall gesetzt, also löst der Prozessor eine GP aus. Die GP wird nun in eine IPC mit den Portinformationen aus der I/O - Instruction umgewandelt und vom Kern zu dem für den Port verantwortlichen Pager geschickt. Der Pager antwortet im Gegenzug auf diese IPC mit einer Mapping - IPC, welche zurück zum Kern geschickt wird. Mit dieser Mapping - IPC wird nun der Anwendung das Zugriffsrecht auf den Port gewährt.

Möchte die Task nun auf n I/O - Ports zugreifen, müsste dieser Vorgang auch n - *mal* durchgeführt werden. Dies ist extrem ineffizient. Deswegen wird das RPC - Protokoll für GP benutzt.

Vorgang mit RPC

Statt einer I/O - Instruction werden nun von dem Thread mittels RPC - Protokoll direkt die nötigen Ports beim Pager *„bestellt“*. Dabei wird keine GP ausgelöst, sondern der Thread schickt eine IPC mit den Portinformationen direkt an den Pager. Allerdings muss auch hier der Umweg über den Kern gemacht werden.
Der Pager antwortet nun wieder mit einer Mapping - IPC, welche er zum Kern schickt. Die Zugriffsrechte auf die I/O - Ports werden nun auf einen Schlag gewährt.

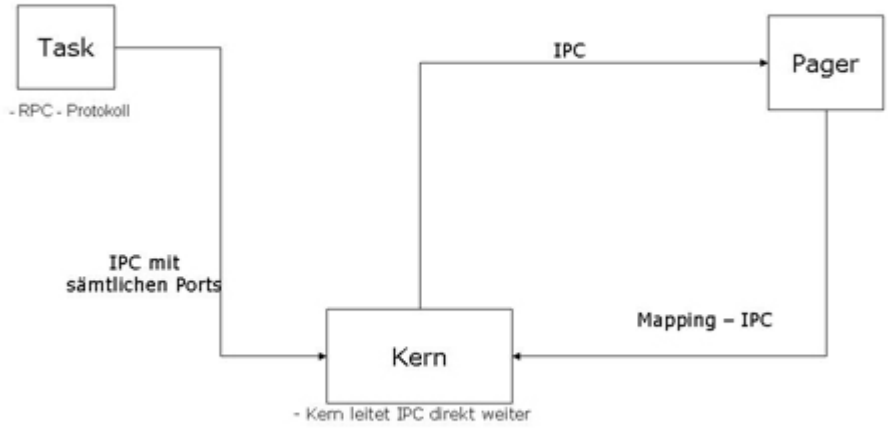

Abbildung 4: Vorgang mit RPC - Protokoll

3.7 Legacy Support

Bei Legacy Support geht es darum mit bereits vorhandener Legacy Software umzugehen. Bei Legacy Software handelt es sich um Software, die nur in kompiliertem Zustand vorliegt (z.B. kommerzielle Programme). Grundsätzlich sind zwei Einstellungen möglich:

- **Implizites I/O - Mapping**
  Grundsätzlich haben Anwendung allumfassenden Zugriff auf I/O - Ports. Sollen Zugriffe nicht gewährt werden müssen sie explizit ungemappt werden.
- **Explizites I/O - Mapping** Grundsätzlich haben Anwendung keinerleih Zugriffsrechte auf I/O - Ports. Zugriffe müssen explizit durch map oder grant gewährt werden.

4 Realisierung der I/O - Flexpages

In diesem letzten Abschnitt werde ich nun etwas zur konkreten Realisierung der I/O - Flexpages sagen. Bei der Realisierung handelt es sich um *L4-Ka/Hazelnut*. Zuerst gehe ich kurz auf die I/O Privilege Levels ein. Dann werde ich genaueres zur IOPBM und dem damit zusammenhängenden Mapping - Konzept sagen. Ein Problem, welches ich bisher nicht angesprochen habe, stellt die unmap - und die grant - Funktion dar. Werden diese beiden Funktionen auf Tasks angewendet, die selber bereits eigene Zugriffsbereiche an andere Tasks weitergegeben haben, müssen die unterliegenden Tasks ebenfalls ungemappt werden bzw. bei grant einem neuen Bezugspartner zugeordnet werden. Dieses Problem wird durch die I/O - Mapping - Datenbank gelöst, welche ich als letzten Punkt dieses Abschnittes vorstelle.

4.1 I/O - Privilege - Levels

Alle auf Benutzerebene laufenden Prozesse haben einen IOPL von 0. Einen Sonderfall stellt Sigma 0 dar, welche einen IOPL von 3 hat. Letztendlich bedeutet dies, dass alle auf Benutzerebene laufende Prozesse die IOPBM überprüft bekommen müssen. Bei Sigma 0 wird die IOPBM jedoch niemals überprüft.

4.2 I/O-Permission-Bitmap und das Mapping - Konzept

Jede Anwendung braucht gezwungenermaßen eine IOPBM, in der die Zugriffsrechte gespeichert sind. Die IOPBM einer Anwendung befindet sich im Kernel - Space. Die Bits in der IOPBM zeigen, ob Zugriffsrechte auf den jeweiligen I/O - Port vorhanden sind oder nicht. Bei map, grant und unmap werden die verantwortlichen Bits in der IOPBM des Empfängers geändert, z.B. werden bei der map-Funktion die entsprechenden Bits auf 0 gesetzt. Bei grant müssen die entsprechenden Bits in der IOPBM des Senders auf 1 gesetzt werden. Bei unmap geschieht dies analog.

4.3 I/O-Mapping-Datenbank

Die Datenbank entstand aus dem Wunsch heraus hierarchisch gegliederte Adressbereiche bereitzustellen. Für ein solches Konzept bietet sich eine baumartige Struktur an, in der die Knoten einen I/O - Bereich beschreiben und die Kanten jeweils ein Mapping darstellen. Als erstes werde ich nun die Datenstruktur eines Knotens darstellen. Danach gehe ich auf die Realisierung der Funktionen map, grant und unmap ein.

4.3.1 Datenstruktur eines I/O-Mapping-Nodes

Die I/O-Mapping-Datenbank bzw. der I/O-Mapping-Tree hat nur eine Datenstruktur, nämlich den I/O-Mapping-Node. Jeder Anwendung ist bekannt, welche Knoten zu ihr gehören. Die Liste der Knoten, die zu der Anwendung gehören, nenne ich Taskliste. Ein solcher I/O - mapping - node hat folgende Attribute:

- **taskID**
  Die taskID stellt die Verbindung zwischen Knoten und dazugehöriger Task her.
- **low und high**
  low ist der I/O - Port mit der kleinsten Adresse. Zusammen mit high beschreibt er den I/O - Bereich, welcher vom Knoten spezifiziert wird. High ist dementsprechend der I/O - Port mit der größten Adresse.
- **depth**
  Diese Zahl gibt die Tiefe des Knotens an.
- **prev und next**
  prev und next sind Pointer auf den nächsten und den vorherigen Knoten im Mapping - tree.
- **prev_taskliste und next_taskliste**
  prev_taskliste gibt den Vorgänger in der Taskliste, next_taskliste den Nachfolger in der Taskliste an.
- **parent**
  Der parent stellt den Knoten dar, von welchem dieser Knoten seinen Bereich zugemappt bekommen hat.

4.3.2 Mapping Algorithmen

Hier beschreibe ich nun das Vorgehen bei map- , grant- und unmap-Funktionen im Zusammenspiel mit der Datenbank.

- **map:** MAP_IO_FPAGE(from, to, fpage)
  from beschreibt den Sender, *to* den Empfänger und *fpage* spezifiziert den I/O - Bereich, welcher gemappt werden soll.

1. Zunächst wird in der Taskliste des Senders nach allen relevanten Knoten gesucht. Hierzu schneidet man den Bereich, der von dem Knoten spezifiziert wird, mit dem Bereich, der von *fpage* beschrieben wird. Ist der Schnitt leer, ist der Knoten nicht relevant und wird übergangen. Erweist sich der Schnitt als nicht leer, handelt es sich um einen relevanten Knoten und es wird mit 2. fortgefahren.
2. Nun wird ein neuer Knoten kreiert, der als Bereich gerade die Schnittmenge (von nun an i genannt) der fpage und dem aus 1. relevanten Knoten hat.
3. Jetzt wird in dem Bereich i vom Empfänger bereits vorhandene Mappings mittels unmap aufgehoben.
4. Der neu kreierte Knoten wird nun in den Mapping-Tree und in die Tasklist vom Empfänger eingetragen.
5. Jetzt müssen nur noch die verantwortlichen Bits in der I/O-Permission-Bitmap vom Empfänger auf 0 gesetzt werden.

- **grant:** GRANT_IO_FPAGE(from, to, fpage)
  from beschreibt den Sender, *to* den Empfänger und *fpage* spezifiziert den I/O - Bereich, welcher gegrantet werden soll.

  1. Zunächst wird in der Taskliste des Senders nach allen relevanten Knoten gesucht. Hierzu schneidet man den Bereich, der von dem Knoten spezifiziert wird mit dem Bereich, der von *fpage* beschrieben wird. Ist der Schnitt leer, ist der Knoten nicht relevant und wird übergangen. Erweist sich der Schnitt als nicht leer, handelt es sich um einen relevanten Knoten und es wird mit 2. fortgefahren.
  2. Nun wird in dem Bereich, welcher gerade die Schnittmenge (von nun an i genannt) der *fpage* und dem aus 1. relevanten Knoten ist, vom Empfänger bereits vorhandene Mappings mittels unmap aufgehoben.
  3. Mittels der Funktion grant_subtree verweißt der Zeiger *parent* der unterliegenden Knoten nun auf *to*.
  4. In *from* werden nun die Mappings im Bereich i mittels unmap aufgehoben.
  5. Als letztes werden nun die Bits in der IOPBM vom Sender auf 1 und vom Empfänger auf 0 gesetzt.

- **unmap:** UNMAP_IO_FPAGE(task, fpage, mapmask)
  In *task* wird die Anwendung spezifiziert, für welche das unmap durchgeführt werden soll. *fpage* beschreibt den I/O - Bereich und *mapmask* ist ein boolscher Wert, der angibt, ob die Mappings des Knoten und seiner Unterknoten oder nur die Mappings der Unterknoten aufgehoben werden sollen.

1. Zunächst wird in der Taskliste nach allen relevanten Knoten gesucht. Hierzu schneidet man den Bereich, der von dem Knoten spezifiziert wird mit dem Bereich, der von der fpage beschrieben wird. Ist der Schnitt leer, ist der Knoten nicht relevant und wird übergangen. Erweist sich der Schnitt als nicht leer, handelt es sich um einen relevanten Knoten und es wird mit 2. fortgefahren.
2. Ist ein solcher Knoten gefunden, werden dessen Unterknoten ungemappt. Ist mapmask gesetzt, werden auch die Zugriffsrechte des Knotens gelöscht.

5 Schlusswort

Vor der Implementierung der I/O - Flexpages war jeder Anwendung der Zugriff auf alle I/O-Ports möglich. Nach der Einführung der I/O-Flexpages ist diese Sicherheitlücke geschlossen, auch wenn es die Geschwindigkeit verringert. Mapping, granting und unmapping sind weit davon entfernt optimal zu sein.
I/O-Space zu handhaben ist ein kritischer Teil des Betriebssystems, da Zugriffe auf I/O - Ports einfach mißbraucht werden können. Deswegen sollte Zugriff nur vertrauenswürdigen Komponenten gewährt werden. Mit den I/O - Flexpages können I/O - Ports wie Hauptspeicher verteilt und geschützt werden. Durch das Nutzen des Flexpagekonzepts für I/O - Space kann das System einfach erweitert werden ohne zu viel neuen Code einfügen zu müssen. Pager können benutzt werden um Hauptspeicher und I/O - Ports zu managen. Die I/O - Flexpages führen zu besserem Schutz und mehr Durchsichtigkeit in der Systembedienung von L4.

Literatur

[1] Jan Stöß: I/O-FlexPages on the x86-Architecture(2002)

[2] J. Liedtke: L4 Nucleus Version X Reference Manual (1999)

[3] A.Au, G.Heiser: L4 User Manual (1999)

[4] Intel Architecture Software Developer's Manuals

[5] Prof. P.P.Spies: Ansätze für Betriebssysteme der Zukunft, http://wwwspies.informatik.tu-muenchen.de/lehre/seminare/SS02/hauptsem/

[6] www.wikipedia.org