Inhaltsverzeichnis

Dieses Teilprojekt beschäftigt sich mit der Zugriffskontrolle von Benutzern bzw. den von diesen gestartete Prozesse auf Dateien durch Access Control Lists (ACL) unter Linux für das ext2 Dateisystem. Dabei beschränken wir uns darauf, lediglich die Elemente, die für die ACL Unterstützung hinzukommen oder sich ändern, zu spezifizieren und verifizieren.

Der natürlichsprachlichen Beschreibung folgt jeweils eine Z Spezifikation. Teilweise werden Bezeichner aus den Linux Betriebsystem Quellen benutzt. Wo nicht anders gesagt, gilt die folgende Einführung allg. für die gängigen Unix Derivate (Linux, Solaris, HP, ...).

1 Begriffseinführung

1.1 Benutzer, Gruppen, Zugriffsrechte

Benutzer und Gruppen von Benutzern werden durch natürliche Zahlen identifiziert. Je nach Darstellung werden diese wegen Überlauf vorzeichenbehafteter Variablen auch teilweise als negative Zahlen dargestellt. Da auf diesen IDs jedoch keine Arithmetik ausgeführt wird, schadet diese Inkonsistenz nicht. Benutzer- (UID) und Gruppen ID (GID) werden durch den Z Typ *ID* repräsentiert:

$$ID == \mathbb{Z}$$

Unter Unix werden die folgenden Zugriffsrechte auf Dateien unterschieden:

- Lesen der Datei (<u>r</u>ead).
- Schreiben in die Datei (<u>w</u>rite).
- Ausführen der Datei (e<u>x</u>ecute).

Die Bedeutung dieser Rechte ist für reguläre Dateien und Verzeichnisse unterschiedlich, die Semantik ist hier jedoch unerheblich. Diese Rechte werden durch *PERMISSION* repräsentiert:

$$PERMISSION ::= r \mid w \mid x$$

Die Standard Unix Zugriffsrechte von Dateien unterteilen Kandidaten Prozesse für den Zugriff in drei Kategorien:

<u>u</u>ser Die UID des Prozesses ist identisch mit derjenigen der Datei.

group Die GID der Datei ist identisch mit einer der zusätzlichen Gruppen des Prozesses.

other Prozesse, auf die beiden obigen Bedingungen nicht zutreffen.

In jeder Kategorie kann eine beliebige Kombination dieser Rechte gesetzt (gewährt) werden. Einem Prozeß werden jeweils die gesetzten Rechte für seine Kategorie gewährt.

1.2 Prozesse

Prozesse sind durch eine Benutzer Identifikation (UID), einer Gruppen Identifikation (GID) und einer Reihe von zusätzlichen Gruppen (supplementary groups) charackterisiert. Normalerweise ist die GID des Prozeßes auch in der Liste zusätzlicher Gruppen enthalten. Der Linux Kern erlaubt jedoch, beides unabhängig voneinander zu setzen. Diese Konsistenz wird nur durch entsprechend programmierte Anwendungsprogramme realisiert. Zuletzt ist die *umask* relevant. Dies sind Standard Unix Rechte, die neu angelegten Dateien gegeben werden. Der aktuelle Prozeß wird durch das Schema *current* repräsentiert:

$$\begin{array}{|l}
\textit{current} \\
\hline
\textit{fsuid} : \textit{ID} \\
\textit{groups} : \mathbb{P}\,\textit{ID} \\
\hline
\end{array}$$

Die hier betrachteten IDs eine Prozesses sind die für das Dateisystem gültigen IDs. Diese sind unter Linux u. U. verschieden von den effektiven IDs. Diese können wiederum verschieden von den echten (real) IDs des Benutzers sein. Sämtliche IDs werden beim Erzeugen eines neuen Prozesses von dem Vater- an den Kindprozeß vererbt. Initial werden sie von der Login Shell auf die IDs des sich einloggenden Benutzers gesetzt. Daher sind die IDs eines Prozesses i. A. identisch mit denen des Benutzers, der den Prozeß gestartet hat. Prozesse mit einer bestimmten UID und der Benutzer mit der gleichen UID werden im folgenden synonym verwendet. Diese IDs können auf verschiedenste Weisen geändert werden. Die dabei verwendeten Sicherheitsmechanismen sollen jedoch nicht Inhalt dieses Projektes sein. Daher ist auch die Unterscheidung dieser verschiedenen IDs nicht weiter wichtig.

Für den Superuser entfallen alle genannten Einschränkungen und Zugriffskontrollen. Unter anderem kann der Superuser die genannten IDs frei wählen. Dies ist notwendig, damit der Superuser ein defektes System reparieren oder ein neues System aufsetzen kann.

1.3 Dateien

Dateien haben folgende Attribute, die den Zugriff auf diese Dateien regeln:

Art Dateien können ein Verzeichnis (directory) von Dateien sein oder eine reguläre Datei. Eine weitere Unterscheidung ist hier irrelvant.

Besitzer Diese UID wird beim Anlegen der Datei mit derjenigen des anlegenden Prozesses gesetzt. Nur der Superuser kann den Besitzer einer Datei ändern.

Gruppe Diese GID wird beim Anlegen der Datei mit der primären GID des Prozesses gesetzt. Nur der Besitzer oder Superuser kann die Gruppe einer Datei ändern. Die GID kann auf eine beliebige andere Gruppe des Prozesses gesetzt werden. Der Superuser kann eine beliebige Gruppe setzen.

Rechte Die Standard Zugriffsrechte. Nur der Besitzer oder der Superuser kann die Zugriffsrechte ändern.

Dies ergibt folgendes Schema:

FILE

$$
\begin{array}{l}
isdir : \{0, 1\} \\
user : ID \\
group : ID \\
uperm : \mathbb{P}\, PERMISSION \\
gperm : \mathbb{P}\, PERMISSION \\
operm : \mathbb{P}\, PERMISSION
\end{array}
$$

$$RESPONSE ::= deny \mid success$$

Das folgende Schema drückt aus, ob die Standard Unix Zugriffsrechte einer Datei *file*? dem aktuellen Prozeß *current* eine Menge von Zugriffsrechten *mask*? gewähren. Das Ergebnis *result*! ist dann *true*. Dieses Schema entspricht der gleichnamigen Funktion bei [1].

$$
\begin{array}{l}
\textit{ext2\_permission} \\
\hline
\Xi current \\
file? : FILE \\
mask? : \mathbb{P}\, PERMISSION \\
result! : RESPONSE \\
\hline
result! = \\
\textbf{if } file?.user = fsuid \\
\textbf{then} \\
\quad (\textbf{if } mask? \subseteq file?.uperm \textbf{ then } success \textbf{ else } deny) \\
\textbf{else if } file?.group \in groups \\
\quad \textbf{then} \\
\quad (\textbf{if } mask? \subseteq file?.gperm \textbf{ then } success \textbf{ else } deny) \\
\quad \textbf{else} \\
\quad (\textbf{if } mask? \subseteq file?.operm \textbf{ then } success \textbf{ else } deny)
\end{array}
$$

1.4 ACL

Eine ACL erweitert die 3 Kategorien der Standard Unix Zugriffsrechte. Zusätzlich zum Typ des ACL Eintrags wird eine ID angegeben. Folgenden Typ kann ein ACL Eintrag haben:

acl_user_obj Die UID des Prozesses ist identisch mit derjenigen der Datei. Dies entspricht den user Rechten.

acl_user Die UID des Prozesses ist identisch mit der angegebenen UID.

acl_group_obj Die GID der Datei ist identisch mit einer der zusätzlichen Gruppen des Prozesses. Dies entspricht den group Rechten.

acl_group Die angegebene GID ist identisch mit einer der zusätzlichen Gruppen des Prozesses.

acl_other_obj Prozesse, auf die beiden obigen Bedingungen nicht zutreffen. Dies entspricht den other Rechten.

Der entsprechende Z Typ ist $ACLE\_TYPE$:

$$
\begin{array}{l}
ACLE\_TYPE ::= \\
\quad acl\_user\_obj \mid \\
\quad acl\_group\_obj \mid \\
\quad acl\_other\_obj \mid \\
\quad acl\_user \mid \\
\quad acl\_group
\end{array}
$$

Ein ACL Eintrag ist nun aus folgenden Komponenten aufgebaut:

type ist der Typ des Eintrags.

tag ist die UID oder GID, auf die sich der Eintrag bezieht.

perm gibt die Rechte an, die dieser Eintrag gewährt.

$$
\begin{array}{|l}
\hline
\textit{ACLENTRY} \\
\textit{type} : \textit{ACLE\_TYPE} \\
\textit{tag} : \textit{ID} \\
\textit{perm} : \mathbb{P}\,\textit{PERMISSION} \\
\hline
\end{array}
$$

Eine ACL ist eine Sequenz von ACL Einträgen:

$$ACL == \mathrm{seq}\, ACLENTRY$$

Eine valide ACL muß folgende Bedingungen erfüllen:

- Einträge des Typs *user_obj*, *group_obj*, *other_obj* müssen genau einmal vorhanden sein.
- Jedes Paar aus Typ und Tag darf nur einmal vorkommen.

2 Spezifikation der Linux ACL

2.1 Abstrakte Spezifikation

Hier spezifizieren wir auf einer zunächst möglichst abstrakten Ebene das Verhalten des betrachteten Systems, die ACL Komponente des ext2 Dateisystems.

Der betrachtete Teil des ext2 Dateisystem hat dann folgende abstrakte Eigenschaften:

- Verzeichnisse können eine default ACL haben. In diesem Verzeichnis neu angelegte Dateien erhalten dann diese ACL. Zur Unterscheidung wird die bisher diskutierte ACL einer Datei nun als access ACL bezeichnet. Mit diesem Design folgen wir Solaris, HP, sowie der Implementierung von Remy Card.
- Jede Datei hat eine access ACL.
- Dateien sind eindeutig durch eine Inode Nummer identifiziert.

Außerdem fordern wir, daß die Standard Unix Zugriffsrechte aller Dateien jederzeit der ACL der Datei gemäß folgendem Schema entsprechen müssen:

$$
\begin{array}{|l}
\hline
\textit{AclMatchStandard} \\
\textit{file} : \textit{FILE} \\
\textit{acl} : \textit{ACL} \\
\hline
\exists_1\, uu, gg, oo : \textit{ACLENTRY} \bullet \\
\quad uu \in \operatorname{ran} acl \land \\
\quad uu.type = acl\_user\_obj \land \\
\quad uu.perm = \textit{file}.uperm \land \\
\quad gg \in \operatorname{ran} acl \land \\
\quad gg.type = acl\_group\_obj \land \\
\quad gg.perm = \textit{file}.gperm \land \\
\quad oo \in \operatorname{ran} acl \land \\
\quad oo.type = acl\_other\_obj \land \\
\quad oo.perm = \textit{file}.operm \\
\hline
\end{array}
$$

Das ext2 modelliert sich dann als:

$$
\begin{array}{|l}
\hline
\textit{ext2} \\
\textit{files} : \mathbb{N} \pfun \textit{FILE} \\
\textit{access} : \mathbb{N} \pfun \textit{ACL} \\
\textit{default} : \mathbb{N} \pfun \textit{ACL} \\
\hline
\operatorname{dom} \textit{default} \subseteq \operatorname{dom} \textit{access} = \operatorname{dom} \textit{files} \\
\forall\, i\_ino : \mathbb{N} \mid i\_ino \in \operatorname{dom} \textit{files} \bullet \\
\quad \mathbf{let}\ \textit{file} == \textit{files}(i\_ino) \bullet \mathbf{let}\ acl == access(i\_ino) \bullet \\
\quad\quad \textit{AclMatchStandard} \\
\forall\, i\_ino : \mathbb{N} \mid i\_ino \in \operatorname{dom} \textit{default} \bullet \\
\quad \mathbf{let}\ f == \textit{files}(i\_ino) \bullet f.isdir = 1 \\
\hline
\end{array}
$$

Initial gibt es nur das root Verzeichnis im Dateisystem:

$$
\begin{array}{|l}
\hline
\textit{ext2\_init} \\
\textit{ext2} \\
\hline
\exists\, root : \textit{FILE} \mid root.isdir = 1 \bullet \\
\quad \textit{files} = \{0 \mapsto root\} \\
\hline
\end{array}
$$

ext2_acl_permission überprüft, ob durch eine ACL einem Benutzer bestimmte Zugriffsrechte gewährt werden. Eingabe des Schemas in der ersten Version ist:

die ID des Benutzers

die ID der Gruppen, denen der Benutzer angehört, inklusive der Gruppe, die er mit newgrp und Paßwort angenommen hat.

die Datei

die verlangten Zugriffsrechte

$$
\begin{array}{|l}
\hline
\textit{ext2\_acl\_permission} \\
\Xi \textit{FILE} \\
acl? : ACL \\
test\_user? : ID \\
test\_groups? : \mathbb{P}\, ID \\
test\_perm? : \mathbb{P}\, PERMISSION \\
result! : RESPONSE \\
\hline
result! = \\
\textbf{if } (test\_user? = 0) \\
\textbf{then} \\
\quad success \\
\textbf{else if } (test\_user? = user) \\
\textbf{then} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\, i).type = acl\_user\_obj \land \\
\qquad test\_perm? \subseteq (acl?\, i).perm) \textbf{ then } success \textbf{ else } deny \\
\textbf{else if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\, i).type = acl\_user \land (acl?\, i).tag = test\_user?) \\
\textbf{then} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\, i).type = acl\_user \land \\
\qquad (acl?\, i).tag = test\_user? \land \\
\qquad test\_perm? \subseteq (acl?\, i).perm) \textbf{ then } success \textbf{ else } deny \\
\textbf{else if } (group \in test\_groups? \lor (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\, i).tag \in test\_groups?)) \\
\textbf{then} \\
\quad \textbf{if } ((group \in test\_groups?) \land \\
\qquad (\exists\, i : \mathrm{dom}\, acl? \bullet ((acl?\, i).type = acl\_group\_obj \land \\
\qquad\quad test\_perm? \subseteq (acl?\, i).perm))) \lor \\
\quad (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\, i).tag \in test\_groups? \land \\
\qquad (acl?\, i).type = acl\_group \land \\
\qquad test\_perm? \subseteq (acl?\, i).perm) \textbf{ then } success \textbf{ else } deny \\
\textbf{else} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet \\
\qquad (acl?\, i).type = acl\_other\_obj \land test\_perm? \subseteq (acl?\, i).perm) \\
\quad \textbf{then } success \textbf{ else } deny \\
\hline
\end{array}
$$

Unsere erste Annahme zu dieser Spezifikation war, daß die angeforderten Rechte nur gewährt werden, wenn genau ein acl_group oder acl_group_obj-Eintrag in der ACL diese Rechte enthält. Dies ist auch im Moment so unter Linux implementiert. Unter Solaris und mit ziemlicher Sicherheit auch gemäß POSIX 1003.6, werden die Rechte jedoch dann gewährt, wenn die angeforderten Rechte in der Vereinigung der gewährten Rechte aller passenden acl_group und acl_group_obj- Einträge enthalten sind. Dies wird im folgendem Schema so definiert. Außerdem werden User und Gruppen-IDs nicht mehr als Eingaben angenommen sondern aus *current* gelesen. Stattdessen wird *file* als Eingabewert betrachtet.

Außerdem werden User und Gruppen-IDs nicht mehr als Eingaben angenommen sondern aus *current* gelesen. Stattdessen wird Eingabewert betrachtet.

$ext2\_acl\_permission1$

$$
\begin{array}{l}
\Xi current \\
file? : FILE \\
acl? : ACL \\
mask? : \mathbb{P}\, PERMISSION \\
result! : RESPONSE \\
\hline
fsuid \neq 0 \\
\\
result! = \\
\textbf{if } (fsuid = file?.user) \\
\textbf{then} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_user\_obj \land \\
\quad\quad mask? \subseteq (acl?\ i).perm) \\
\quad \textbf{then } success \textbf{ else } deny \\
\textbf{else if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_user \land (acl?\ i).tag = fsuid) \\
\textbf{then} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_user \land \\
\quad\quad (acl?\ i).tag = fsuid \land \\
\quad\quad mask? \subseteq (acl?\ i).perm) \\
\quad \textbf{then } success \textbf{ else } deny \\
\textbf{else if } (file?.group \in groups \lor \\
\quad (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).tag \in groups \land \\
\quad (acl?\ i).type = acl\_group)) \\
\textbf{then} \\
\quad (\textbf{let } perms == \\
\quad\quad \{\, p : PERMISSION \mid \exists\, acle : \mathrm{ran}\, acl? \bullet \\
\quad\quad (acle.type = acl\_group\_obj \land file?.group \in groups \land p \in acle.perm) \lor \\
\quad\quad (acle.type = acl\_group \land acle.tag \in groups \land p \in acle.perm)\} \bullet \\
\quad\quad \textbf{if } (mask? \subseteq perms) \textbf{ then } success \textbf{ else } deny) \\
\textbf{else} \\
\quad \textbf{if } (\exists\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_other\_obj \land mask? \subseteq (acl?\ i).perm) \\
\quad \textbf{then } success \textbf{ else } deny
\end{array}
$$

Verfeinerung des Schemas:

$$
\begin{array}{l}
\textit{ext2\_acl\_permission2} \\
\hline
\Xi current \\
file? : FILE \\
acl? : ACL \\
aclsize? : \mathbb{N} \\
mask? : \mathbb{P}\, PERMISSION \\
result! : RESPONSE \\
\hline
fsuid \neq 0 \\
(\exists\, i : 1 \mathrel{..} aclsize? \bullet \\
\quad (acl?\ i).type = acl\_user\_obj \land \\
\quad file?.user = fsuid \land \\
\quad result! = \mathbf{if}\ mask? \subseteq (acl?\ i).perm\ \mathbf{then}\ success\ \mathbf{else}\ deny) \lor \\
(\exists\, i : \operatorname{dom} acl? \bullet \\
\quad (acl?\ i).type = acl\_user \land \\
\quad (acl?\ i).tag = fsuid \land \\
\quad \neg\, (\exists\, j : \operatorname{dom} acl? \mid j < i \bullet \\
\qquad (acl?\ i).type = acl\_user\_obj \land file?.user = fsuid) \land \\
\quad result! = \mathbf{if}\ mask? \subseteq (acl?\ i).perm\ \mathbf{then}\ success\ \mathbf{else}\ deny) \lor \\
(file?.group \in groups \lor (\exists\, i : \operatorname{dom} acl? \bullet \\
\quad (acl?\ i).type = acl\_group \land \\
\quad (acl?\ i).tag \in groups) \land \\
\quad \neg\, (\exists\, j : \operatorname{dom} acl? \mid j < i \bullet \\
\qquad ((acl\ i).type = acl\_user\_obj \land file?.user = fsuid) \lor \\
\qquad ((acl\ i).type = acl\_user \land (acl\ i).tag = fsuid)) \land \\
\quad (\mathbf{let}\ perms == \\
\qquad \{\, p : PERMISSION \mid \exists\, acle : \operatorname{ran} acl? \bullet \\
\qquad (acle.type = acl\_group\_obj \land file?.group \in groups \land p \in acle.perm) \lor \\
\qquad (acle.type = acl\_group \land acle.tag \in groups \land p \in acle.perm)\} \bullet \\
\qquad result! = \mathbf{if}\ mask? \subseteq perms\ \mathbf{then}\ success\ \mathbf{else}\ deny)) \lor \\
(\exists\, i : \operatorname{dom} acl? \bullet \\
\quad (acl?\ i).type = acl\_other\_obj \land \\
\quad \neg\, (\exists\, j : \operatorname{dom} acl? \mid j < i \bullet \\
\qquad ((acl\ i).type = acl\_user\_obj \land file?.user = fsuid) \lor \\
\qquad ((acl\ i).type = acl\_user \land (acl\ i).tag = fsuid) \lor \\
\qquad ((acl\ i).type = acl\_group\_obj \land file?.group \in groups) \lor \\
\qquad ((acl\ i).type = acl\_group \land (acl\ i).tag \in groups)) \land \\
\quad result! = \mathbf{if}\ mask? \subseteq (acl?\ i).perm\ \mathbf{then}\ success\ \mathbf{else}\ deny) \\
\hline
\end{array}
$$

2.2 Spezifikation relevanter Systemaufrufe

Für die Unterstützung von ACL im ext2 kommt lediglich der Systemaufruf *sys_acl_ctl* hinzu, um die ACL einer Datei abzufragen und zu setzen. Die Implementation der übrigen Systemaufrufe ändert sich nicht, da die Schnittstelle zum ext2 zum Test der Autorisierung gleich bleibt. Lediglich der Zusammenhang derjenigen Systemaufrufe, die die Zugriffsrechte von Dateien ändern, mit der ACL Unterstützung muß spezifiziert werden. Das sind folgende Systemaufrufe:

sys_acl_ctl Setzen und lesen der access und default ACL einer geöffneten Datei oder einer Datei gegebenen Namens.

sys_creat Anlegen einer regulären Datei.

sys_mknod Anlegen einer Spezialdatei, zählt für uns als reguläre Datei.

sys_mkdir Anlegen eines Verzeichnisses.

sys_chmod Ändern der Standard Unix Zugriffsrechte einer Datei gegebenen Namens.

sys_fchmod Ändern der Standard Unix Zugriffsrechte einer geöffneten Datei.

sys_chown Ändern des Besitzers einer Datei gegebenen Namens. dem Dateinamen. Symbolische links werden verfolgt.

sys_lchown Ändern des Besitzers einer Datei gegebenen Namens. Symbolische links werden nicht verfolgt.

sys_fchown Ändern der Besitzers einer geöffneten Datei.

Die verschiedenen Systemaufrufe, die jeweils auf einer Datei gegebenen Namens und auf einer bereits geöffneten Datei arbeiten, werden nicht weiter unterschieden: Lediglich die Suchen des Inode, die beim Öffnen einer Datei stattfindet, muß nachgeholt werden. Links brauchen ebenfalls nicht behandelt zu werden: Es wird lediglich ein neuer Name für denselben Inode eingeführt. Die Zugriffskontrolle findet aber nur durch den Inode statt. Symbolische links werden hier wie reguläre Dateien behandelt.

Die Funktionalität von *sys_acl_ctl* wird in sechs Schemata aufgeteilt:

get_acl gibt die ACL einer Datei zurück.

get_default_acl gibt die ACL einer Datei zurück.

get_acl_size gibt die Länge der ACL einer Datei zurück.

$get\_default\_acl\_size$ gibt die Länge der ACL einer Datei zurück.

set_acl setzt die access ACL einer Datei.

$set\_default\_acl$ setzt die default ACL einer Datei.

$$
\begin{array}{l}
\textbf{get\_acl} \\
\hline
\Xi ext2 \\
i\_ino? : \mathbb{N} \\
acl! : ACL \\
\hline
i\_ino? \in \operatorname{dom} files \\
acl! = access(i\_ino?)
\end{array}
$$

$$
\begin{array}{l}
\textbf{get\_default\_acl} \\
\hline
\Xi ext2 \\
i\_ino? : \mathbb{N} \\
acl! : ACL \\
\hline
i\_ino? \in \operatorname{dom} files \\
acl! = default(i\_ino?)
\end{array}
$$

$$
\begin{array}{l}
\textbf{get\_acl\_size} \\
\hline
\Xi ext2 \\
i\_ino? : \mathbb{N} \\
length! : \mathbb{N} \\
\hline
i\_ino? \in \operatorname{dom} files \\
length! = \#(access(i\_ino?))
\end{array}
$$

$$
\begin{array}{l}
\textbf{get\_default\_acl\_size} \\
\hline
\Xi ext2 \\
i\_ino? : \mathbb{N} \\
length! : \mathbb{N} \\
\hline
i\_ino? \in \operatorname{dom} files \\
length! = \#(default(i\_ino?))
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad set\_acl \quad}} \\
\Delta ext2 \\
i\_ino? : \mathbb{N} \\
acl? : ACL \\
result! : RESPONSE \\
\hline
i\_ino? \in \operatorname{dom} files \\
default' = default \\
files' = files \\
access' = access \oplus \{i\_ino? \mapsto acl?\} \\
result! = success \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad set\_default\_acl \quad}} \\
\Delta ext2 \\
i\_ino : \mathbb{N} \\
acl? : ACL \\
result! : RESPONSE \\
\hline
i\_ino \in \operatorname{dom} files \\
\mathbf{let}\ f == files(i\_ino) \bullet f.isdir = 1 \\
files' = files \\
access' = access \\
default' = default \oplus \{i\_ino \mapsto acl?\} \\
result! = success \\
\hline
\end{array}
$$

Bei den folgenden Systemaufrufen müssen wir oft aus Standard Unix Zugriffsrechten eine nach *AclMatchStandard* entsprechende ACL bilden. Dafür definieren wir die Funktion *acl_from_mode*:

$$
\begin{array}{|l}
acl\_from\_mode : FILE \to ACL \\
\hline
\forall f : FILE \bullet acl\_from\_mode(f) = \langle (\mu\, u : ACLENTRY \mid \\
\quad u.type = acl\_user\_obj \land \\
\quad u.tag = f.user \land \\
\quad u.perm = f.uperm), \\
(\mu\, g : ACLENTRY \mid \\
\quad g.type = acl\_group\_obj \land \\
\quad g.tag = f.group \land \\
\quad g.perm = f.gperm), \\
(\mu\, o : ACLENTRY \mid \\
\quad o.type = acl\_other\_obj \land \\
\quad o.perm = f.operm \land \\
\quad o.tag = 0)\rangle
\end{array}
$$

Durch *sys_creat* wird eine neue Datei in einem Verzeichnis dir? angelegt. Dabei werden Standard Unix Zugriffsrechte , Besitzer und Gruppe der Datei festgelegt. Wir lassen undefiniert, wie diese Attribute vom umgebenden System gesetzt werden. Die Default ACL des Verzeichnisses, in dem die Datei angelegt wird (*dir?*), muß verwendet werden, wenn vorhanden. Wenn nicht, ist eine ACL aus den Standard Unix Rechten zu bilden.

$$
\begin{array}{|l}
\hline
sys\_creat \\
\Delta ext2 \\
dir? : \mathbb{N} \\
result! : RESPONSE \\
\hline
result! = success \\
default' = default \\
\exists\, i\_ino : \mathbb{N} \mid i\_ino \notin \operatorname{dom} files \bullet \\
\exists\, file : FILE \bullet \\
\quad files' = files \oplus \{i\_ino \mapsto file\} \land \\
\quad file.isdir = 0 \land \\
\quad access' = \\
\qquad \textbf{if } dir? \in \operatorname{dom} default \\
\qquad \textbf{then } access \cup \{i\_ino \mapsto default(dir?)\} \\
\qquad \textbf{else } access \cup \{i\_ino \mapsto acl\_from\_mode(file)\} \\
\hline
\end{array}
$$

sys_mknod hat für uns die gleichen Eigenschaften wie *sys_creat*:

$$
sys\_mknod \mathrel{\widehat{=}} sys\_creat
$$

Die Spezifikation von *sys_mkdir* unterscheidet sich von *sys_creat* nur darin, daß wir ein Verzeichnis statt einer regulären Datei anlegen. Die default

ACL wird nur als access ACL, nicht aber als eigene default ACL wiederum übernommen!

$$
\begin{array}{l}
\textbf{sys\_mkdir} \\
\hline
\Delta ext2 \\
dir? : \mathbb{N} \\
result! : RESPONSE \\
\hline
result! = success \\
default' = default \\
\exists\, i\_ino : \mathbb{N} \mid i\_ino \notin \operatorname{dom} files \bullet \\
\exists\, file : FILE \bullet \\
\quad files' = files \oplus \{i\_ino \mapsto file\} \land \\
\quad file.isdir = 1 \land \\
\quad access' = \\
\qquad \textbf{if } dir? \in \operatorname{dom} default \\
\qquad \textbf{then } access \cup \{i\_ino \mapsto default(dir?)\} \\
\qquad \textbf{else } access \cup \{i\_ino \mapsto acl\_from\_mode(file)\}
\end{array}
$$

Werden mit `chmod` oder `fchmod` die Standard Unix Zugriffsrechte geändert, so muß auch die access ACL entsprechend geändert werden:

$$
\begin{array}{l}
\textbf{sys\_chmod} \\
\hline
\Delta ext2 \\
i\_ino? : \mathbb{N} \\
uperm? : \mathbb{P}\, PERMISSION \\
gperm? : \mathbb{P}\, PERMISSION \\
operm? : \mathbb{P}\, PERMISSION \\
\hline
default' = default \\
\textbf{let } f == files(i\_ino?) \bullet \\
\textbf{let } f' == (\mu\, f' : FILE \mid \\
\quad f'.isdir = f.isdir \land \\
\quad f'.user = f.user \land \\
\quad f'.group = f.group \land \\
\quad f'.uperm = uperm? \land \\
\quad f'.gperm = gperm? \land \\
\quad f'.operm = operm?) \bullet \\
\qquad access' = access \oplus \{i\_ino? \mapsto acl\_from\_mode(f')\} \land \\
\qquad files' = files \oplus \{i\_ino? \mapsto f'\}
\end{array}
$$

In [1] wird lediglich die ID des Datei Besitzer oder der Datei Gruppe geändert. Hier ist noch mit anderen Unix zu vergleichen.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{sys\_chown}} \\
\hline
\Xi current \\
\Delta ext2 \\
i\_ino? : \mathbb{N} \\
uid? : ID \\
gid? : ID \\
\hline
default' = default \\
access' = access \\
\mathbf{let}\ f == files(i\_ino?) \bullet \\
\mathbf{let}\ f' == (\mu f' : FILE \mid \\
\quad f'.isdir = f.isdir \land \\
\quad f'.user = uid? \land \\
\quad f'.group = gid? \land \\
\quad f'.uperm = f.uperm \land \\
\quad f'.gperm = f.gperm \land \\
\quad f'.operm = f.operm) \bullet \\
\qquad files' = files \oplus \{i\_ino? \mapsto f'\} \\
\hline
\end{array}
$$

2.3 Verfeinerung der Spezifikation

Hier folgt nun eine Verfeinerung der obigen Spezifikation. Außerdem erweitern wir nun die Schemata um die Striktheit: Das Ergebnis aller Operation muß immer definiert sein. Wir spezifizieren Zustände und Operationen nur soweit, wie sie für die Betrachtung der ACL Unterstützung interessant ist. Wir berücksichtigen nun auch Aspekte der Implementation. Das sind bei [1] folgende Punkte:

1. In der abstrakteren Spezifikation war es unerheblich und blieb offen, in welcher Reihenfolge ACL Einträge in einer ACL vorkamen. aus Performanz Gründen müssen wir jedoch bei der Implementierung eine günstige Reihenfolge definieren, die dann zwingend notwendig für das korrekte Funktionieren der Implementation ist.

2. Da eine Datei beim Anlegen eine ACL erhält, die eindeutig aus den Standard Unix Zugriffsrechten gebildet wird, möchten wir darauf verzichten, diese ACLs abzuspeichern. Statt dessen wird die entsprechende ACL bei Bedarf gebildet.

3. Beim Anlegen einer Datei in einem Verzeichnis mit default ACL erhält jede Datei initial die gleiche ACL. Damit diese nicht immer wieder neu angelegt werden muß, sollen Dateien sich eine ACL teilen können: Erst

beim Verändern wird die ACL kopiert. Die Referenzierung findet über ACL Header statt.

4. Die ACL Header und Daten werden in Blöcken abgelegt, die durch jeweils einen Inode und indirekte Blöcke verwaltet werden. Beide werden über Indizes referenziert, die die Nummer der Datenstrukurt in den vom Inode referenzierten Blöcken sind.

2.3.1 Inodes und Blöcke

Bei einem Inode Dateisystem werden die Daten einer Datei in Blöcken abgespeichert. Die Daten einer Datei werden durch Angabe der Inode und eines Index referenziert. *Inode* definieren wir als einen abstrakten Datentyp. Außerdem definieren wir bereits zwei spezielle Instanzen: $s\_acl\_idx\_inode$ ist die reservierte ACL Index Inode, $s\_acl\_data\_inode$ ist die reservierte ACL Daten Inode. Die Blöcke der ersteren beinhaltet Verweise auf die in den Blöcken der letzteren Inode gespeicherten ACLs.

$$Inode ::= s\_acl\_idx\_inode \mid s\_acl\_data\_inode$$

Die im Dateisystem vorhanden Blöcke werden durch das folgende generische Schema ausgedrückt. Dabei dient der Typ der im Block abgespeicherten Daten als Parameter des Schemas:

$$\begin{array}{|l}
\hline
ext2_blocks[T] \\
blocks : (Inode \times \mathbb{N}) \rightarrow\!\!\!\!\!+ \; T \\
\hline
\end{array}$$

2.3.2 ACL Strukturen

Alle verwendeten Strukturen wurden mit padding bytes zur gleichen Größe aufgefüllt.

Die Struktur ext2_acl_header hat folgende Komponenten:

- aclh_valid Muß für einen validen Header gesetzt sein.
- aclh_version Version der ACL.
- aclh_acle_count Anzahl der ACL Einträge.
- aclh_ref_count Anzahl der Dateien, die diesen ACL Header referenzieren.
- aclh_first_acle Index des ersten ACL Eintrags.

- aclh_no Nummer des Header für Gegenprobe.
- aclh_reserved[3] padding bytes, nicht modelliert.

$$
\begin{array}{|l}
\hline
\textit{ext2\_acl\_header} \\
aclh\_valid : \mathbb{N} \\
aclh\_version : \mathbb{N} \\
aclh\_acle\_count : \mathbb{N} \\
aclh\_ref\_count : \mathbb{N} \\
aclh\_first\_acle : \mathbb{N} \\
aclh\_no : \mathbb{N} \\
\hline
\end{array}
$$

Die Anzahl der ACL Header, die pro Block Platz finden, ist durch $header\_per\_block$ gegeben. Da die Strukturen gleich groß sind, ist dies auch die Anzahl der ACL Einträge.

$$
\begin{array}{|l}
header\_per\_block : \mathbb{N}
\end{array}
$$

Die Struktur $ext2\_acl\_desc$ beschreibt einen Block mit ACL Headern. $acld\_max\_acl$ ist die Länge der längsten Folge freier ACL Header in dem beschrieben Block. $acld\_min\_acl$ ist die Länge der kürzesten Folge freier ACL Header in dem beschrieben Block. $acld\_free\_acl$ ist die Anzahl freier ACL Header in dem Block insgesamt.

$$
\begin{array}{|l}
\hline
\textit{ext2\_acl\_desc} \\
acld\_max\_acl : \mathbb{N} \\
acld\_min\_acl : \mathbb{N} \\
acld\_free\_acl : \mathbb{N} \\
\hline
acld\_min\_acl \leq acld\_max\_acl \leq acld\_free\_acl \\
\hline
\end{array}
$$

Die Blöcke mit den ACL Headern sind vollständig in Stücke der Größe eines ACL Header aufgeteilt. Der erste Platz wird durch einen ACL Deskriptor ($ext2\_acl\_desc$) in Anspruch genommen. Der zweite Platz wird für eine Bitmap benutzt, die die besetzten Plätze bezeichnet. Folglich müssen die beiden ersten Plätze immer als belegt gekennzeichnet sein. Ansonsten würden die Verwaltungsinformationen überschrieben werden.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{header\_block}} \\
\hline
ext2\_acl\_desc \\
bitmap : \{ i : \mathbb{N} \mid i < header\_per\_block \} \rightarrow \{0, 1\} \\
header : \mathbb{N} \nrightarrow ext2\_acl\_header \\
\hline
\forall i : \mathbb{N} \mid i \in \operatorname{dom} header \bullet \\
\quad i \neq 0 \land \\
\quad i \neq 1 \land \\
\quad i < header\_per\_block \land \\
\quad bitmap(i) = 1 \\
\forall i : \mathbb{N} \bullet bitmap(i) = 1 \Rightarrow i \in \operatorname{dom} header \lor i = 0 \lor i = 1 \\
bitmap(0) = 1 \land bitmap(1) = 1 \\
\mathbf{let}\ ranges == \{ r : \mathbb{N} \mid \exists i : \mathbb{N} \bullet \\
\quad \forall j : \mathbb{N} \mid i \leq j < i + r \bullet \\
\qquad j \in \operatorname{dom} bitmap \land bitmap(j) = 0 \} \bullet \\
\qquad\quad acld\_max\_acl = max(ranges) \land \\
\qquad\quad acld\_min\_acl = min(ranges) \\
acld\_free\_acl = \#\{ i : \mathbb{N} \mid i \in \operatorname{dom} bitmap \land bitmap(i) = 0 \} \\
\hline
\end{array}
$$

Ein ACL Eintrag wird durch die Struktur *ext2_acl_entry* repräsentiert.

- acle_version Version der ACL.
- acle_perms Die gewährten Zugriffsrechte.
- acle_acle_count Die Anzahl der ACL Einträge.
- acle_type Der Typ des ACL Eintrags.
- acle_tag Die UID oder GID.
- acle_aclh Die Nummer des zugehörigen ACL header. Diese ist nur zum Gegenprüfen vorhanden.
- acle_no Die Nummer des Eintrags. Diese ist nur zum Gegenprüfen vorhanden.
- acle_next Zeiger auf (Nummer des) nächsten (freien) Eintrags.

$$
\begin{array}{|l}
\hline
\textit{ext2\_acl\_entry} \\
\textit{acle\_version} : \mathbb{N} \\
\textit{acle\_perms} : \mathbb{P}\,\textit{PERMISSION} \\
\textit{acle\_acle\_count} : \mathbb{N} \\
\textit{acle\_type} : \textit{ACLE\_TYPE} \\
\textit{acle\_tag} : \textit{ID} \\
\textit{acle\_aclh} : \mathbb{N} \\
\textit{acle\_no} : \mathbb{N} \\
\textit{acle\_next} : \mathbb{N} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{entry\_block} \\
\textit{entries} : \mathrm{seq}\,\textit{ext2\_acl\_entry} \\
\hline
\#\textit{entries} \leq \textit{header\_per\_block} \\
\hline
\end{array}
$$

Analog zu *header_block* wird die Zuordnung von Daten des Typs *entry_block* zu den Blöcken, in denen sie abgespeichert sind, wird durch das Schema *entries* spezifiziert:

$$\textit{entries} \mathrel{\widehat{=}} \textit{ext2\_blocks}[\textit{entry\_block}]$$

Eine Datei erhält nun die Attribute i_file_acl und i_dir_acl, die die Nummer des ACL Header für den access ACL bzw. default ACL enthalten. 0 bedeutet, daß keine ACL zugewiesen wurde.

$$
\begin{array}{|l}
\hline
\textit{FILE\_Impl} \\
\textit{FILE} \\
\textit{i\_file\_acl} : \mathbb{N} \\
\textit{i\_dir\_acl} : \mathbb{N} \\
\hline
\end{array}
$$

ext2_Impl ist eine Verfeinerung von *ext2*. headers ist die Zuordnung von Daten des Typs *header_block* zu den Blöcken, in denen sie abgespeichert sind.

$$
\begin{array}{|l}
\hline
\textit{ext2\_Impl} \\
\textit{files} : \textit{Inode} \rightarrow \textit{FILE\_Impl} \\
\textit{headers} : \textit{ext2\_blocks}[\textit{header\_block}] \\
\hline
\end{array}
$$

$$\begin{array}{l}
\textbf{ext2_Impl} \\
\hline
files : Inode \rightarrow FILE_Impl \\
\hline
\forall f : FILE_Impl \mid f \in files \bullet \\
\quad f.i_file_acl \neq 0 \Leftrightarrow \\
\mathbf{let}\ no == f.i_file_acl\ \mathsf{div}\ header_per_block \bullet \\
\quad no \in \mathrm{dom}\, s_acl_idx_inode \wedge \\
\quad LET\ block == s_acl_idx_inode\ no \bullet \\
\quad\quad f.i_file_acl\ \mathsf{mod}\ header_per_block \in \mathrm{dom}\, block.header \\
\mathbf{let}\ no == f.i_dir_acl\ \mathsf{div}\ header_per_block \bullet \\
\quad no \in \mathrm{dom}\, s_acl_idx_inode \wedge \\
\quad LET\ block == s_acl_idx_inode\ no \bullet \\
\quad\quad f.i_dir_acl\ \mathsf{mod}\ header_per_block \in \mathrm{dom}\, block.header \\
\hline
\end{array}$$

$$\begin{array}{l}
\textbf{ext2_Impl_init} \\
\hline
ext2_Impl \\
\hline
files = \emptyset \\
headers.blocks = \emptyset \\
\hline
\end{array}$$

2.3.3 Operationen

Die spezifizierten Funktionen nehmen teilweise als Parameter *int* an, prüfen jedoch nur, ob der Parameter 0 oder nicht 0 ist. Für diesen Fall benutzen wir folgenden Datentyp für den Parameter:

$$BOOL ::= TRUE \mid FALSE$$

$$\begin{array}{l}
\textbf{ext2_bread}[T] \\
\hline
\Delta ext2_blocks[T] \\
i? : Inode \\
block? : \mathbb{N} \\
create? : BOOL \\
err! : RESPONSE \\
t! : T \\
\hline
(i?, block?) \in \mathrm{dom}\, blocks \Rightarrow err! = success \wedge t! = blocks(i?, block?) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{_\ allocate_acl_header\ __________} \\
\Xi ext2_Impl \\
\hline
\end{array}$$

$set\_acl\_Impl \mathrel{\widehat=} \mathbin{\raise 0.8ex\hbox{$\scriptstyle\circ$}\mkern-4mu\lower 0.3ex\hbox{$\scriptstyle 9$}} allocate_acl_header$

$$\begin{array}{|l}
\multicolumn{1}{l}{_\ set_acl_Impl\ __________} \\
\Xi ext2_Impl \\
i_ino? : \mathbb{N} \\
acl? : ACL \\
result! : RESPONSE \\
\hline
result! = success \\
check_ext2_acl \\
i_ino? \in \operatorname{ran} files \\
LETf == files(i_ino?) \bullet \exists\, i : \mathbb{N} \mid i \notin s_acl_idx_inodef.i_file_acl : \mathbb{N} \\
f.i_dir_acl : \mathbb{N} \\
\# acl? = 3 \land udate_mode_from_acl \lor \\
\# acl? \neq 3 \land udate_mode_from_acl \lor \\
LETlen == \# acl? \bullet \land\ neuerBlock\ \exists\, block? : \mathbb{N} \mid \\
\quad index \in \operatorname{dom} s_acl_idx_inode \land \\
\quad letbh : header_block == s_acl_idx_inode(index) \bullet \qquad bh.\ \exists\, block? : \mathbb{N} \mid block? \notin \operatorname{dom} s_acl_idx_inode \bullet \exists\, s_acl_idx_inode' = s_acl_idx_inode \oplus \ddot{\imath}nit_header_block\ \exists\, i_ino : \mathbb{N} \mid i_ino \notin \operatorname{dom} files \bullet \\
\exists\, file : FILE \bullet \\
\quad files' = files \oplus \{i_ino \mapsto file\} \land \\
\quad file.isdir = 0 \land \\
s_acl_data_inodeaclhs_acl_idx_inode' = s_acl_idx_inode \oplus \{\mapsto\} s_acl_data_inode : \mathbb{N}_1 \pfun ext2_acl_headeraccess' = access \oplus \{file? \mapsto acl?\} \\
default' = default \\
files' = files \\
\hline
\end{array}$$

Als zusätzliche Bedingung füer die Validität einer ACL führen wir ein, daß die Einträge in der Reihenfolge acl_user_obj, acl_user, acl_group_obj, acl_group, acl_other_obj sortiert sein müssen. Dies ist ie Reihenfolge, in der die ACLENTRYs untersucht und abgespeichert werden - dies ist auch die Reihenfolge in der sie unter Linux abgespeichert werden - wird durch den Typ des einzelnen ACLENTRY bestimmt. Die folgende Sequenz vom Typ $\operatorname{seq} ACLE$ dient dazu, eine Ordnungsrelation zu definieren:

$$ato == \langle acl\_user\_obj, acl\_user, acl\_group\_obj, acl\_group, acl\_other\_obj \rangle$$

Damit läßt sich die folgende Ordnungsrelation definieren:

$$\_ \leq_{ACL} \_ : ACLE\_TYPE \leftrightarrow ACLE\_TYPE$$

$$\begin{array}{l} \forall\, at1, at2 : \\ \quad ACLE\_TYPE \bullet (at1 \leq_{ACL} at2) \Leftrightarrow \\ \quad (\exists\, i, j : \mathrm{dom}\, ato \bullet (ato\ i) = at1 \land (ato\ j) = at2 \land i \leq j) \end{array}$$

Das folgende Schema ist erfüllt, wenn die ACL valide ist. Dieses Schema entspricht der gleichnamigen Funktion bei [1]. Wir prüfen die oben neu eingeführte Bedingung sowie die Invarianten aus der bastrakten Spezifikation:

$$\begin{array}{l} check\_ext2\_acl \\ \hline acl? : ACL \\ result! : RESPONSE \\ \hline \exists_1\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_user\_obj \\ \exists_1\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_group\_obj \\ \exists_1\, i : \mathrm{dom}\, acl? \bullet (acl?\ i).type = acl\_other\_obj \\ \forall\, i, j : \mathrm{dom}\, acl? \bullet i < j \Rightarrow (acl?\ i).type \leq_{ACL} (acl?\ j).type \\ \forall\, i, j : \mathrm{dom}\, acl? \bullet \\ \quad ((acl?\ i).type = (acl?\ j).type \land (acl?\ i).tag = (acl?\ j).tag) \\ \quad \Rightarrow i = j \\ result! = success \end{array}$$

3 Verifikation

- ext2_permission mit ext2_acl_permission
- check_ext2_acl

4 Test

Literatur

[1] ACL Patch für den 2.1.99 kernel von Remy Card.