Aus dem Nähkästchen geplaudert: Locks

Konkurrenzdenken

Unter einem echten Multitasking-System wie Linux laufen viele Prozesse gleichzeitig. Dabei kommt es nicht selten zu konkurrierenden Zugriffen auf Dateien. Um Datenverluste zu vermeiden, gibt es mehrere Methoden, um Dateien mit Locks zu versehen. So sichert sich ein Programm exklusiven Zugriff. Marc André Selig

Fast jeder Linux-Rechner hat einen MTA, einen Mail Transfer Agent installiert. Sei es Postfix, Exim oder gar Sendmail. Er nimmt E-Mails entweder lokal, zum Beispiel über einen Fetchmail-Prozess, oder über TCP entgegen. Stellt der MTA fest, dass die Mail an einen lokalen Benutzer adressiert ist, gibt er sie an den Local Delivery Agent (LDA) weiter. Dieser schreibt sie – eventuell nach einem kurzen Umweg über Filter wie Procmail – in eine Mailbox-Datei.

Verschwundene E-Mails

Kommen zwei E-Mails gleichzeitig an, gibt der MTA sie an zwei verschiedene LDA-Prozesse weiter. Jeder Prozess versucht in dieselbe Mailbox zu schreiben (siehe Abbildung 1). Im günstigsten Fall landen die Mails in der falschen Reihenfolge in der Datei. Wahrscheinlicher ist es jedoch, dass eine der E-Mails verloren geht, weil beide Prozesse ihre Daten gegenseitig überschreiben.

Der erste Prozess öffnet die Datei, danach auch der zweite. Dann schreibt die erste Instanz des LDA die E-Mail in die Datei und schließt diese. Die andere Instanz hat die Mailbox aber noch geöffnet und überschreibt einfach die Daten der ersten Mail. Nach dem Schließen der Datei ist die erste Mail verschwunden.

Locks sorgen für Ordnung

So genannte Locks sind die gängige Lösung für dieses Problem. Ein Lock (Schloss) sperrt eine Ressource, solange ein Prozess sie nutzt. Wer zum Beispiel auf eine gelockte Datei zugreifen will, erhält eine Fehlermeldung, oder die Funktion zum Öffnen der Datei wartet eine Weile. Erst wenn der aktuelle Prozess sie wieder freigibt, kehrt die Funktion in das Programm zurück und der nächste darf die Datei öffnen.

Locks lösen so unter anderem das Problem der gleichzeitig ankommenden E-Mails: Der LDA mit der ersten E-Mail sperrt die Mailbox. Der zweite LDA erhält eine Fehlermeldung beim Öffnen, wartet ein Weilchen und versucht es später noch einmal. Ist die erste Mail komplett ausgeliefert, die Mailbox sicher geschlossen und das Lock entfernt, flattert die zweite Nachricht an.

Mit Datei-Locks ist man aber gleich wieder neuen Problemen ausgeliefert. Eines der kleinsten ist die Tatsache, dass sie die Reihenfolge von scheinbar offensichtlichen Abläufen durcheinander bringen. Folgendes Szenario verdeutlicht die Bredouille: Der LDA stellt eine E-Mail zu und hat die Mailbox gelockt, ein zweiter Prozess wartet auf die Freigabe der Datei, um eine zweite E-Mail zuzustellen. Just in dem Moment, in dem der LDA die Mailbox wieder freigibt, trudelt eine dritte E-Mail ein und der dritte Prozess lockt die Datei. Der zweite Prozess muss sich wieder hinten anstellen. Eine klassische Race Condition.

Ein gravierenderes Problem sind Stale Locks, wörtlich übersetzt also abgestandene oder veraltete Locks. Sie entstehen, wenn ein Prozess seine Dateien nicht ordnungsgemäß freigibt, weil er abgestürzt ist oder ein Benutzer ihn mit dem »kill«-Kommando abgeschossen hat.

Deadlock

Andere Prozesse warten dann ewig auf die Freigabe der Datei. E-Mails werden nicht mehr zugestellt und auch der Benutzer darf die Mailbox nicht mehr verändern. Besonders gefährlich sind Stale Locks bei der Verwendung eines NFS-Servers. Unter bestimmten Umständen friert der Server komplett ein.

Manche Locks verschwinden von selbst, wenn der zugehörige Prozess beendet wird. Andere Lockfiles enthalten die Prozesskennung im Namen oder als Inhalt. Bei der Verwendung von Locks sollten Programme also immer prüfen, ob es

zu einer gelockten Datei überhaupt noch einen aktiven Prozess gibt.
Es gibt zahlreiche verschiedene Verfahren, um ein File-Locking zu implementieren. Grundsätzlich unterscheidet man zwischen Mandatory und Advisory Locks, also zwingenden und beratenden. Beide Varianten stellt unter anderem der Betriebssystem-Kernel als Systemaufrufe zur Verfügung.

Zwingend und beratend

Mandatory Locks können von Prozessen nicht umgangen werden. Der Kernel sorgt dafür, dass ein Zugriff auf die besetzte Ressource nicht möglich ist, solange das Lock existiert. Das ist eine sichere Methode, die aber anfällig ist für Stale Locks und obendrein kaum anwendbar auf Netzwerk-Dateisystemen wie NFS und AFS.
Ein Advisory oder Discretionary Lock ist nur ein Hinweis. Ob Prozesse sich an diese Locks halten, bleibt ihnen überlassen. Natürlich ist es immer sinnvoll, möglichst vorsichtig mit gelockten Dateien umzugehen. Diese sanften Locks können sowohl im Kernel als auch in Userspace-Bibliotheken implementiert sein. Außerdem ist es möglich, Advisory Locks mit einfachen Dateien zu realisieren. Diese Methode ist dann auch auf NFS und Konsorten anwendbar, sofern eine halbwegs atomare Funktion zum Öffnen der Dateien vorhanden ist.

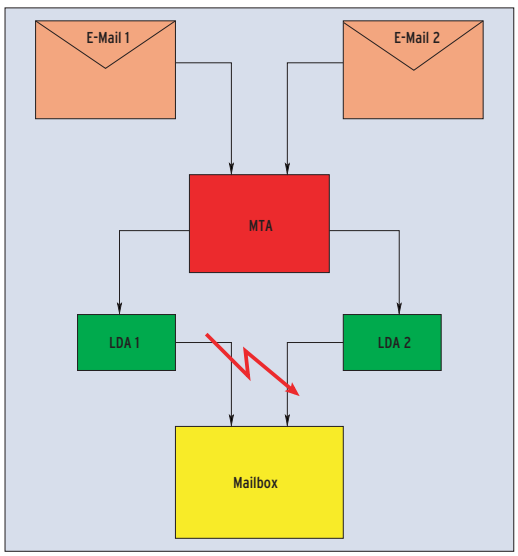

Abbildung 1: Ohne Locking würde bei zwei gleichzeitig ankommenden E-Mails eine verloren gehen. Denn der eine LDA-Prozess merkt nicht, was der andere tut.

Wer Programme schreibt, die auf vielen Unix-Varianten laufen sollen, handelt sich mit Mandatory Locks garantiert Probleme ein. Programme, die lediglich unter verschiedenen Linux-Versionen laufen sollen, sind einfacher zu implementieren. Im Wesentlichen gibt es hier die Posix-kompatible »fcntl()«-Funktion (in Linux ist sie identisch mit »lockf()«) sowie die BSD-kompatible »flock()«. Bibliotheken erleichtern oft den Umgang mit diesen Ungetümen. Listing 1 zeigt ein Beispiel in Perl.

Lockfiles

Einfacher als die beschriebenen Kernel-Locks sind primitive Semaphore. Ursprünglich ist ein Semaphor ein weithin sichtbares Zeichen, mit dem früher Schiffe kommunizierten. In Unix bezeichnet der Begriff eine Datei oder Datenstruktur, die einen bestimmten Zustand darstellt, etwa den Zustand einer Ressource.
Programme signalisieren Mailbox-Locks traditionell mit einem angehängten ».lock« im Dateinamen. Die Mailbox »/var/mail/mas« hätte also die Lockdatei »/var/mail/mas.lock«. Diese Variante funktioniert sogar auch über NFS hinweg, jedoch benötigt jedes Programm, das die Mailbox ändert, Rechte zum Anlegen von Dateien in »/var/mail/«.
Außerdem müssen alle Programme – wie oben beschrieben – prüfen, ob bereits ein Lockfile existiert, sowie die Prüfung und das Anlegen der Lock-Datei in einer einzigen Operation durchführen. Ansonsten kommt es zu einer Race Condition. Der folgende Pseudocode zeigt, wie eine solche Race Condition aussieht:

```
IF: not exists file.lock
  THEN: create file.lock
  ELSE: wait
```

Führen zwei Programme derartigen Code gleichzeitig aus, springen beide in den »THEN«-Teil der Abfrage und legen die Lockdatei an. Damit hat sich das Locking erübrigt, da jetzt wieder beide Programme gleichzeitig auf die Mailbox zugreifen. Besser ist folgende Vorgehensweise:

```
IF: create exclusive file.lock
  THEN: do something
  ELSE: wait
```

Hier bestehen sowohl die Abfrage als auch das Anlegen der Lockdatei aus einer einzigen atomaren Aktion. Führen zwei Programme diesen Code gleichzeitig aus, kommt es nicht mehr zu einer Race Condition, da eines der Programme die Datei vor dem anderen anlegt. Listing 2 zeigt, wie sich der obige Pseudocode in Perl realisieren lässt. Das Shellskript in Listing 3 bedient sich dazu des Programms »lockfile«, es ist Teil der Procmail-Distribution.
Zum Abschluss noch eine Warnung: Bei Netzwerk-Dateisystemen sind selbst die sonst sehr zuverlässigen Lockfiles nicht absolut zuverlässig. Man sollte sich also als Programmierer nie darauf verlassen, dass ein Programm sowohl auf dem lokalen Dateisystem als auch über NFS gleichermaßen funktioniert. *(mwe)* ■

Infos

[1] Marc André Selig: „Dienstgespräche", Linux-Magazin 05/04, S. 76

Listing 1: Datei-Locking mit Perl

```
01 #!/usr/bin/perl -wT
02 use Fcntl ':flock';
03 sysopen(FH, "Datei", O_RDWR|O_CREAT) or die "Fehler
   sysopen: $!";
04 flock(FH, LOCK_EX) or die "Fehler flock: $!";
05
06 # ... Datei beschreiben ...
07
08 flock(FH, LOCK_UN) or die "Fehler flock: $!";
09 close FH or die "Fehler close: $!";
```

Listing 2: Atomares Locking

```
01 #!/usr/bin/perl -w
02 use Fcntl;
03 sysopen(FH, "file.lock", O_WRONLY|O_EXCL|O_CREAT)
04   or die "Fehler sysopen: $!");
```

Listing 3: Einfaches File-Locking in der Shell

```
01 #!/bin/sh
02 # lockfile gehört zu procmail und ist bei den meisten
03 # Linux-Distributionen verfügbar
04 lockfile file.lock
05 # ...
06 rm -f file.lock
```