Op consult bij Dokter Unix

Het antwoord op prangende Unix vragen...

Q: *Beste Dr. Unix, hoe zit het nu precies met file locking onder UNIX? Ik kom uit een mainframe omgeving en daar was het toch niet gebruikelijk dat je met meer dan 1 persoon in een file kon rommelen. Onder UNIX lijkt dit wel heel gewoon? Ik heb wel eens wat gelezen over locking, maar ik heb het toch niet helemaal begrepen.*

-- "Nieuweling" uit A.

A: Beste "Nieuweling",

Soms doen zich van die dingen voor die vrijwel geen toeval kunnen zijn. Je zult het wel niet geloven, maar Dr. Unix heeft vorige week ditzelfde fenomeen nog tijdens het Sinterklaasfeest uitgelegd aan zijn kleinkinderen.

Vroeger, toen Dr. Unix nog jong was, kende UNIX geen file locking. Dit waren mooie tijden, waarbij je je als programmeur volledig uit kon leven, toen de computers nog van hout waren en de mannen van ijzer en toen C nog C was zoals Brian & Dennis het bedoeld hadden! Maar goed, ik dwaal af. Alweer een hele tijd geleden kwam Berkeley UNIX met de *flock()* system call. Ik kan mij nog goed herinneren dat dit werd geintroduceerd. Conform het toenmalige tijdsgewricht was het *not done* om anderen ergens toe te dwingen. *flock()* implementeert dan ook de zogenaamde a*dvisory file locking*. Dit werkt eigenlijk net als het verkeerslicht om de hoek bij onze kliniek: het functioneert omdat we hebben afgesproken dat we er allemaal naar kijken en dat we ons overeenkomstig de lichtsignalen gedragen. *Advisory locking* in UNIX komt erop neer dat het systeem centraal een tabel met locks bijhoudt. Iedere lock entry beschrijft een gedeelte van een file (een byte range) die "gelockt" is. Echter iedereen die aan dit systeem mee wil doen moet zelf *flock()* aanroepen en zich hiernaar gedragen: de lees- en schrijfroutines van UNIX trekken zich van *advisory locking* niets aan! De *flock()* call ondersteunt zowel *shared* als *exclusive* locks. Als iemand een *shared* lock heeft op een gedeelte van een file dan zal een *exclusive* lock voor dat gedeelte (of een deelverzameling ervan) door *flock()* worden geweigerd. Als iemand een *exclusive* lock op een reeks bytes uit de file heeft dan zal natuurlijk iedere lock aanvraag (*shared* of *exclusive*) voor dat gedeelte worden geweigerd.

Als een applicatie van *advisory locking* gebruik maakt is het makkelijk mogelijk om daaromheen te komen. Een command als *cat* of *more* trekt zich immers van die locks niets aan! *Advisory locking* is een mooi en sociaal systeem wat Dr. Unix's goedkeuring kan wegdragen. *Advisory locking* via *flock()* komen we tegen in alle BSD 4.2 compatible UNIX systemen. Waar Dr. Unix echter nog wel eens tegenaan loopt is dat bij *flock()* de locks op files liggen, en niet op file descriptors! Dit betekent dat als een file descriptor wordt gedupliceerd door een aanroep naar *dup()* of *fork()* er nog steeds maar 1 lock op de file is. Via een dergelijke contructie kan een kind proces dus een lock opheffen wat door een ouder is gezet! Dr. Unix moet regelmatig met software echo's en operatief ingrijpen dergelijke situaties oplossen.

Als antwoord op de Berkeley *flock()* kwamen onze vrienden van System V ook met een locking feature, wat in dit geval wordt bestuurd met de *fcntl()* of de *lock()* call. Hier wordt Dr. Unix dus heel moe van. Waarom nu niet elkaar even opbellen om een eenduidige afspraak te maken? Ze doen maar, en de medische stand komt met de brokken te zitten! Wat wel aardig is van *fcntl()* en *lockf()* is dat er handige mogelijkheden in zetten om te testen of een bepaald gebied van een file is gelocked. (Dit kan natuurlijk in BSD *advisory locking* ook, maar in Dr. Unix zijn mening iets minder handig --red)

Maar, "Nieuweling", nu zijn er altijd van die scherpslijpers die anderen precies de wet voor willen schrijven en voor wie *advisory locking* veel te vriendelijk is. Zij kwamen in System V met het *mandatory locking* schema (ook wel *enforcement mode locking* genoemd). Bij *mandatory locking* is er geen ontkomen aan de lock. Zodra iemand een regio van een file heeft gelocked zal het hele UNIX systeem zich hier naar richten. *Mandatory locking* is dan volledig doorgevoerd in de UNIX kernel, en iedere *read()* en/of *write()* controleert eerst of het gedeelte van het bestand wat moet worden gelezen of geschreven niet is gelocked. Indien dat het geval is moeten die processen wachten totdat de lock is vrijgegeven.

Dit is natuurlijk erg gevaarlijk! Als iemand een *mandatory lock* zet op een file kan het vervolgens ook niet met *cat* of *more* worden gelezen totdat de lock is vrijgegeven. Indien een file in O_NONBLOCK mode staat gaat de operatie fout met de foutcode EAGAIN (*Try Again*). Het potentiele gevaar van *mandatory locking* heeft de ontwikkelaars van Linux ertoe genoopt om dit onderhevig te maken aan een mount optie. Alleen op file systemen waarvan tijdens het mounten is aangegeven dat *mandatory locks* zijn toegestaan kan dan een file op die heftige wijze worden gelocked. Dr. Unix vergelijkt *mandatory locking* altijd met de toegangspoort van de kazerne waar hij als hospik zijn dienstplicht heeft vervuld. Een stuk onvriendelijker dan een verkeerslicht, nietwaar?

Een *mandatory lock* wordt net als een System V *advisory lock* gezet met *fcntl()* of met *lockf()*. Of een lock slechts een advies is, of wordt afgedwongen, wordt gestuurd door de *access mode* van de file! Let dus goed op, "Nieuweling", dat er dus geen aparte parameters voor *fcntl()/lockf()* zijn die dit bepalen! In hun oneindige wijsheid hebben de bedenkers van dit schema besloten dat *mandatory locks* alleen kunnen worden gezet op bestanden waarvan het *setgid bit* aan en het *group execute bit* uit is! Dit is natuurlijk een combinatie die normaal gesproken niet voorkomt. Wat heb je namelijk aan *setgid* op een niet executable file? Deze file mode kan worden gezet met "chmod g+s,g-x <bestand>". Als je als programmeur er zeker van wilt zijn dat eventuele locks afgedwongen worden moet je met *fstat()* en *fchmod()* zelf de *access mode* van het bestand afdwingen.

Wel "Nieuweling", zoals je ziet ondersteunt UNIX ook volledige file locking. Ik hoop dat dit je overgang van het mainframe naar UNIX wat gemakkelijker doet verlopen.

Q: *Beste Dr. Unix, mijn manager heeft af en toe van die buien waarin hij niets van UNIX wil weten. Wat kan ik hier aan doen?*

-- "Onzeker" uit L. aan de IJ.

A: Beste "Onzeker",

Toen Dr. Unix nog jong en onbezonnen was danste hij samen met de huidige Mw. Unix rond de ruines bij Stonehenge. Daar heeft hij geleerd wat een krachtige voorspeller voor het menselijk gedrag de maan eigenlijk is. In Berkeley UNIX (onder andere in FreeBSD) zit het programma *pom*, wat je vertelt wat de huidige stand van de maan is:

```
$ pom
The Moon is Waning Gibbous (78% of Full)
$ _
```

Dr. Unix gebruikt dit programma altijd om uit te rekenen wanneer het tijd is om budget voor een nieuwe UNIX machine los te peuteren bij onze geneesheer directeur (Dr. Djia). Ik hoop dat dit helpt.

Zit je ook met een prangende UNIX vraag? Mail het aan Dokter UNIX!