NAME

alarmd.pl - Übertragene Alarme verarbeiten und entsprechend die Ampel ansteuern und akustischen Alarm geben.

BESCHREIBUNG

Der **alarmd.pl** Server lauscht auf einem konfigurierten Netzwerk-Port, um Alarme von Client-Scripten anzunehmen. Sendet ein Client einen Alarm, so wird anhand der Konfiguration entschieden, wie die Ampel leuchten soll und welcher Sound dazu abgespielt wird. Sind keine Alarme vorhanden, so leuchtet die Ampel grün.

KONFIGURATION

Die Konfiguration erfolgt über die Konfigurationsdatei '`alarmd.yml`', welche in dem selben Ordner wie der **alarmd.pl** Server selbst erwartet wird. Die Konfigurationsdatei ist im YAML Format.

Bei dem YAML Format können mehrere Werte "zusammengefasst" werden. Hierbei ist die Anzahl der Leerzeichen oder Tabulatoren für einen Abschnitt immer gleich zu halten.

Kommentare beginnen in YAML mit einer Raute (#).

Die verschiedenen Alarme werden unter dem Abschnitt '`alarms`' beschrieben. Unter dem Abschnitt `alarms` folgt ein neuer Abschnitt für den entsprechenden Client (Clientname). Unter diesem wiederrum die Alarme, welche ein Client auslösen kann. Unter den Alarmen gibt es jeweils zwei neue Abschnitte: '`ampel`' und '`audio`'. Diese können auch ausgelassen werden, falls notwendig.

Unter '`ampel`' wird die Farbe (`color`), die Priorität (`prio`) des Ampelsignals und die Frequenz (`freq`) in Hertz (Hz), mit welcher die Farbe der Ampel blinken soll festgelegt. Siehe auch **ampel.pl** Die Priorität beschreibt, die wichtigkeit eines Ampelsignals. Liegt auf der Ampel zum Beispiel das Signal 'leuchte gelb' mit der Priorität 1 an und tritt Alarm auf, welcher 'blinke gelb' mit Priorität 2 definiert hat, so wird die Ampel gelb blinken. Das Signal 'leuchte gelb' wird hierbei aber nicht "vergessen", sondern bleibt im Hintergrund erhalten. Wird der Alarm mit der Priorität 2 beendet, so wird die Ampel wieder gelb leuchten. Genaueres Siehe `FUNKTIONSWEISE`.

Unter '`audio`' werden zwei Befehle festgelegt. '`in`' für einkommende Alarme. '`quit`' für beendete Alarme. Der Befehl wird dann entsprechend dem Alarm ausgeführt. Der '`audio`' Abschnitt war ursprünglich dazu gedacht, eine Audiodatei abzuspielen, kann aber auch dazu verwendet werden, andere Scripte zu starten, die dann zum Beispiel eine E-Mail verschicken. Hierbei ist zu beachten, dass Befehle, die hinter '`in`' oder '`out`' definiert werden nicht mit dem Alarm gestartet und beendet werden, sondern im Hintergrund gestartet und nicht beendet werden. Der '`in`' Befehl wird bei einem einkommenden Alarm gestartet, der '`out`' Befehl, wenn der Alarm vorbei ist. Die Befehle werden in dem selben Ordner ausgeführt, in dem sich der **alarmd.pl** Server befindet.

Beispiel-Konfiguration

Ein Beispiel erklärt meistens mehr als jede Beschreibung.

```
 config:      # Die Basis-Konfiguration ist unter 'config:'
  port: 5061    # Der Port, auf welchem gelauscht wird
  pass: test    # Das Server-Passwort
  ampel: "./ampel.pl"   # Das Script zum ansteuern der Ampel. (Siehe auch
'ampel.pl')
  separator: "::"    # Die Werte, welche an den Server geschickt werden,
werden durch diese Zeichenkette getrennt.
       # Siehe auch 'PROTOKOLL'
  verbose: 0    # Debug Informationen ausgeben. (1)
  start: 'mplayer start.mp3'  # Befehl, welcher beim erfolgreichen Start
des alarmd.pl ausgeführt wird.
  error: 'mplayer error.mp3'  # Befehl, welcher bei einem Fehler ausgeführt
 wird.
  stop: 'mplayer end.mp3'   # Befehl, welcher beim erfolgreichen Beenden
des alarmd.pl ausgeführt wird.
```

```
 alarms:      # Die einzelnen Alarme werden unter diesem Abschnitt
definiert.
  nagios:     # Der Clientname
   warning:   # Der Alarm, welcher von dem Client 'nagios' gesendet wird
    ampel:   # Das Ampelsignal wird unter diesem Abschnitt definiert.
     color: gelb # Die Farbe der Ampel, welche verwendet wird.
     prio: 1  # Die Priorität des Signals
     freq: 0  # Die Frequenz, mit welcher geblinkt werden soll. Auslassen
für 0; 0 entspricht leuchten.
    audio:   # Die Befehle, welche
     in:   # beim eintreten des Alarms
     quit:  # oder beim beenden des Alarms ausgeführt werden.


   unknown:
    ampel:
     color: gelb
     prio: 1
       # Der 'audio', sowie 'ampel' Abschnitt kann auch ausgelassen werden,
 falls nicht benötigt.
   critical:
    ampel:
     color: rot
     prio: 1
   down:
    ampel:
     color: rot
     prio: 2
     freq: 1.2
   nagdown:
    ampel:
     color: rot
     prio: 3
     freq: 2.4
  ticket:
   neu:
    ampel:
     color: gelb
     prio: 2
     freq: '0.1/0.9'
    audio:
     in: 'mplayer new_ticket.mp3' # Die Audio-Datei wird in dem Selben
Ordner, wie dieses Script selbst, gesucht.
   ticketdown:
    ampel:
     color: gelb
     prio: 3
     freq: 2.4
  internet:
   weg:
    ampel:
     color: gruen
     prio: 3
     freq: 4.8
    audio:
     in: 'mplayer offline.mp3'
     quit: 'mplayer start.mp3'
```

FUNKTIONSWEISE

Zuerst wird überprüft, ob dieses Script als root ausgeführt wird. Die Ausführung als root ist notwendig, da da **ampel.pl** root-Rechte benötigt, um die Ampel anzusteuern. Danach wird in das Verzeichnis gewechselt, in welchem sich dieses Script befindet, damit später ohne großen Aufwand in der Konfiguration Dateien spezifiziert werden können, welche sich in dem selben Verzeichnis befinden.

Die Funktion `exittsk()` wird mit den Signalen *INT*, *TERM*, *QUIT*, *ABRT* und *HUP* verlinkt. Das bewirkt, dass sich dieses Script nicht sofort beendet, sondern zuerst noch der konfigurierte '`stop`' Befehl ausgeführt werden kann.

Danach wird eine neue Warteschlange erstellt (`$inqueue`), welche zur Kommunikation zwischen den einzelnen *Verbindungs-Threads* (`sub connhandler`) und dem *Arbeiter-Thread* (`sub worker`) genutzt wird. Daraufhin wird versucht, den konfigurierten Port zu öffnen. Sollte dies fehlschlagen, wird der konfigurierte '`error`' Befehl ausgeführt und das Programm beendet. Der '`error`' Befehl wird außerdem ausgeführt, sollte eine Fehlerhafte oder unbekannte Anfrage an den Server gestellt werden.

Sobald der Port geöffnet wurde, wird der *Arbeiter-Thread* gestartet. Dieser schaltet die Ampel initial auf grün leuchten, da noch kein Alarm vorhanden ist. Danach wartet der *Arbeiter-Thread* auf neue Aufträge, welche über die Warteschlange eintreffen.

In dem *Hauptprozess* wird inzwischen der konfigurierte '`start`' Befehl im Hintergrund gestartet. Danach beginnt der *Hauptprozess*, bei dem geöffneten Port auf neue Clients zu warten. Baut ein neuer Client eine Verbindung auf, so wird ein neuer *Verbindungs-Thread* gestartet, und das Verbindungs-Handle an diesen übergeben. Danach wird im *Hauptprozess* sofort weiter auf neue Clients gewartet. Dies ist notwendig, damit nicht ein Client alle eingehenden Verbindungen blockieren kann. Durch dieses Vorgehen "kümmert" sich um jeden Client ein eigener Thread.

In dem *Verbindungs-Thread* wird jetzt auf eine (oder mehrere) Nachrichten des verbundenen Clients gewartet (Siehe auch `PROTOKOLL`). Für jede Nachricht wird zunächst anhand dem Konfigurierten Trenner ('`separator`') zerlegt (gesplittet). Danach wird das Passwort überprüft. Sollte das Passwort nicht mit dem konfigurierten Passwort ('`pass`') übereinstimmen, wird sofort die Verbindung geschlossen und der Thread beendet. Stimmt das Passwort hingegen, wird die Nachricht zunächst auf ihre Vollständigkeit geprüft und dann darauf untersucht, ob der Alarm bekannt ist, den der Client melden will. Ist die Nachricht unvollständig, so wird der konfigurierte '`error`' Befehl ausgeführt und dem Client ein "err" gemeldet. Ist der Alarm unbekannt, wird dem Client ein "unknown" gemeldet und ebenfalls der '`error`' Befehl ausgeführt. Ist der Alarm bekannt, wird er der Warteschlange hinzugefügt und dem Client ein "ok" gemeldet.

Der *Arbeiter-Thread* nimmt den Alarm an und prüft zunächst, ob etwas für diesen Alarm konfiguriert wurde ('`ampel`', '`audio`'). Ist keines von beiden konfiguriert worden, wird nichts unternommen und auf den nächsten Alarm gewartet. Ist ein '`audio`' Befehl konfiguriert worden, so wird dieser im Hintergrund gestartet. Der '`in`' Befehl für neue Alarme, der '`quit`' Befehl für beendete Alarme. Wurde ein Ampelsignal definiert und der Alarm als neu gemeldet, wird dieses in einen Hash (`$signal`) eingetragen. Dieser Hash enthält alle aktuell auf der Ampel anliegenden Signale in folgender Reihenfolge sortiert: *Farbe*, *Priorität*, *Eintrittszeitpunkt*. Wird der Alarm als beendet gemeldet, wird das Signal wieder aus dem Hash entfernt. Nachdem der Hash bearbeitet wurde, wird in dem Hash für jede Farbe nach der Frequenz für mit der höchsten Priorität gesucht und diese in einen weiteren Hash (`$final`) der jeweiligen Farbe zugeordnet. Sollten mehrere Frequenzen mit gleicher Priorität vorliegen, wird die genommen, die als erstes in dem Hash steht, welche in der Regel dem ersten eingetroffenen Alarm dieser Priorität entspricht. Es wird nicht empfohlen, mehrere Signale auf einer Farbe mit der selben Priorität zu vergeben. Der Hash `$final` enthält an dieser Stelle die neue Frequenz für jede Farbe. Jetzt wird jeder **ampel.pl** Prozess beendet, welcher in dem Hash `$procs` vorhanden ist. An dieser Stelle leuchtet keine Farbe der Ampel. Danach wird für jede Farbe in dem Hash `$final` ein neuer **ampel.pl** Prozess gestartet, welcher die jeweilige Farbe der Ampel mit der gegebenen Frequenz ansteuert. Wurde kein neuer **ampel.pl** Prozess gestartet, so wird ein **ampel.pl** Prozess mit der Farbe grün gestartet, damit die Ampel grün Leuchtet, wenn kein Fehler da ist. Dann wartet der *Arbeiter-Thread* wieder auf neue Alarme.

PROTOKOLL

Client zu Server

Folgende Nachricht muss zum auslösen eines Alarms an den Server gesendet werden:

```
Passwort::Clientname::Alarmname::1
```

Bei dem Passwort handelt es sich um das Serverpasswort ('pass'). Der Clientname ist der Name, mit wechem sich der Client gegenüber dem Server meldet. Alarmname ist Name des Alarms, welcher ausgelöst oder aufgehoben werden soll. Die Eins markiert, dass der Alarm ausgelöst wird. Bei den doppelten Doppelpunkten (::) handelt es sich um den vordefinierten Trenner ('separator'). Dieser kann je nach Konfiguration abweichen, muss aber immer beim Server und Client der gleiche sein.

Folgende Nachricht hebt den ausgelösten Alarm wieder auf:

```
Passwort::Clientname::Alarmname::0
```

Die Null markiert das Aufheben des Alarms.

Server zu Client

Die Antwort des Servers beschränkt sich auf ein einfaches

```
'ok'
```

falls der Alarm erfolgreich aufgenommen wurde,

```
'err'
```

falls die Anfrage ungültig war oder

```
'unknown'
```

falls der Alarm auf dem Server nicht definiert ist. Ein 'err' und ein 'unknown' starten den unter 'error' Konfigurierten Befehl, um über die Fehlkonfiguration zu benachrichtigen.

Ist das Passwort falsch, wird der Server sofort und ohne Quittierung die Verbindung beenden.

AUTOR

alarmd.pl - Geschrieben von Florian Schießl (florian@trans.net) im Rahmen des Projektes zur IHK Abschlussprüfung zum Fachinformatiker für Systemintegration Sommer 2013. Entwicklungsbeginn ist der 11.3.2013.

SIEHE AUCH

AlarmClient.pm - Ein Perl Modul zur Kommunikation mit diesem Server.

ampel.pl - Eine Farbe einer Cleware Ampel ansteuern.