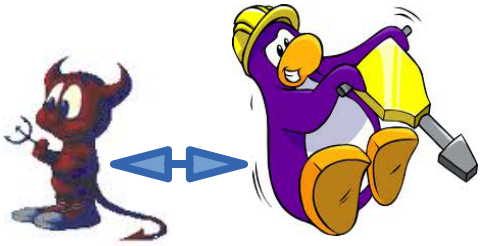
Was ist ein Dämon?



Unter "daemon" versteht man in der LINUIX-Welt, im Gegensatz zur realen Welt, einen hilfreichen Geist, der für den Benutzer quasi unsichtbar nützliche Dienste leistet.

"Unsichtbar" heißt in diesem Zusammenhang, daß es sich um Prozesse handelt, die keine Interaktion des Benutzers erfordern und somit auch keine graphische Benutzeroberfläche benötigen. Einmal gestartet wartet er auf bestimmte Ereignisse und tritt dann in Aktion. Dies kann beispielsweise ein Datenbankserver sein, der darauf wartet, daß jemand eine Verbindung zu ihm aufbaut und Daten aus der Datenbank z.B. mittels SQL abfragt.

In einem UNIX-System gibt es zahlreiche daemons, die allerhand nützlicher Dienste leisten, ohne daß der Benutzer es merkt. Dies sind nicht selten wichtige Systemfunktionen, die mit root-Rechten ausgestattet sein müssen.
Beispiele hierfür sind der secure shell daemon (sshd), der ssh Sitzungen abwickelt, der at-daemon (atd), der zu bestimmten Zeiten jobs im Hintergrund startet, u.v.a. mehr. Diese "traditionellen" UNIX-Dienste werden üblicherweise in den Init-Skripten beim wechseln des runlevels (z.B. beim booten) automatisch gestartet und auch gestoppt. Es gibt aber durchaus Fälle, in denen man dieses Verhalten nicht wünscht. Zum einen hat nicht jeder Benutzer root-Privilegien oder auch die Erfahrung, die man braucht, um die Init-Skripte zu modifizieren. Zum anderen gibt es Dienste, die nicht in den üblichen Linux-Distributionen vorhanden sind (zu Recht). Zwei praktische Beispiele, die keine root-Rechte benötigen, seien hier erläutern:

- File-sharing Systeme
- Dedicated game server

Zunächst jedoch, soll das "Skelett" eines Dämonen (eigentlich ein "Wrapper-Skript", das den eigentlichen Dämon einhüllt) vorgestellt werden, das durch einfachste Konfiguration an die eigenen Bedürfnisse angepasst werden kann.

Das Skelett

In der folgenden Tabelle ist das Skript dargestellt.

skeleton.sh

```
 1  #!/bin/sh
 2  #-----------------------------------------------------
 3  #
 4  # generic daemon wrapper script
 5  #
 6  # OPTIONS: (start|stop|restart|status)
 7  #
 8  #-----------------------------------------------------
 9
10  # --- edit here ---
11
12  DAEMON_DESC="my fancy daemon"
13  DAEMON_BIN="/path/to/my_daemon"
14  DAEMON_PROC="daemon_ps_name"
15  DAEMON_LOGFILE="/path/to/my/logfile/${DAEMON_PROC}.log"
16  DAEMON_OPTIONS=" -foo -bar "
17
18  # --- don't edit below ---
19
20  #-----------------------------------------------------
21  #
22  # get_pid()
23  #
24  #-----------------------------------------------------
25
26  get_pid()
27  {
28          PID=`ps aux | grep -v grep | grep ${DAEMON_PROC} | awk '{print $2}'`
29  }
30
31  #-----------------------------------------------------
32  #
33  # start_daemon()
```

```
34 #
35 #---------------------------------------------------------
36
37 start_daemon()
38 {
39         echo -n "Starting ${DAEMON_DESC}"
40         ${DAEMON_BIN} ${DAEMON_OPTIONS} 2> ${DAEMON_LOGFILE}&
41         echo " ... done."
42 }
43
44 #---------------------------------------------------------
45 #
46 # stop_daemon()
47 #
48 #---------------------------------------------------------
49
50 stop_daemon()
51 {
52         echo -n "Stopping ${DAEMON_DESC}"
53         kill -s SIGKILL $PID
54         echo " ... done."
55 }
56
57 #---------------------------------------------------------
58 #
59 # main
60 #
61 #---------------------------------------------------------
62
63 case "$1" in
64         start)
65                 get_pid
66                 if [ -z "$PID" ] ; then
67                         start_daemon
68                 else
69                         echo "${DAEMON_DESC} is running."
70                         exit 1
71                 fi
72                 ;;
73
74         stop)
75                 get_pid
76                 if [ -z "$PID" ] ; then
77                         echo "${DAEMON_DESC} is not running."
78                         exit 1
79                 else
80                         stop_daemon
81                 fi
82                 ;;
83
84         restart)
85                 get_pid
86                 if [ ! -z "$PID" ] ; then
87                         stop_daemon
88                 fi
89                 start_daemon
90                 ;;
91
92         status)
93                 get_pid
94                 if [ -z "$PID" ] ; then
95                         echo "${DAEMON_DESC} is not running."
96                 else
97                         echo "${DAEMON_DESC} is running. PID is $PID"
98                 fi
99                 ;;
100
101         *)
102                 echo "Usage: $0 {start|stop|status|restart}"
```

```
103             exit 1
104             ;;
105  esac
106
107  exit 0
108
109  # --- EOF ---
```

Erklärung:

Zeile 10-18: Hier wird der daemon durch eigene Modifikationen instanziiert.

Zeile 26-29: Funktion findet die process ID des daemons.

Zeile 37-42: Funktion startet den daemon.

Zeile 50-55: Funktion stoppt den daemon.

Zeile 63-107: Body des Skripts. Hier werden die Parameter ausgewertet.

Das Skript akzeptiert jeweils einen von vier Parameter:

start: Startet den daemon.

stop: Stoppt den daemon, wenn er läuft.

status: Informiert darüber, ob der daemon läuft oder nicht.

restart: Stoppt den daemon (falls er läuft) und startet ihn danach wieder neu.

Vorteil eines solchen generisches Ansatzes ist, dass die Steuerung der *daemons* einheitlich ist und man sich nicht für jeden diverse Besonderheiten merken muss.

Beispiele

Indem man zwischen Zeile 10 und Zeile 18 des Skripts Änderungen macht, kann man sich leicht eigene Dämonen bauen.

File-sharing (overnet)

Erstes Fallbeispiel: Der user "horst-kevin" möchte am file sharing teilnehmen und zwar mit einem overnet core (man kann natürlich auch den edonkey core benutzen). Zu diesem Zweck hat er sich in seinem Home-Verzeichnis ein Verzeichnis ed2k angelegt, in dem das Executable liegt und später auch das Logfile. Die Änderungen am Skelett sind einfach folgende:

Änderungen für overnet core
DAEMON_DESC="overnet core 0.51.2"
DAEMON_BIN="/home/horst-kevin/ed2k/overnet0.51.2"
DAEMON_PROC="overnet0.51.2"
DAEMON_LOGFILE="/home/horst-kevin/ed2k/${DAEMON_PROC}.log"
DAEMON_OPTIONS=" - ! -g -l "

Wenn man kein logfile erzeugen möchte nimmt man einfach /dev/null. Man benötigt hier keine root-Rechte. Zum Starten:

>> ./overnetd.sh start

eingeben. Der overnet core hat die (nicht ungewöhnliche) Eigenschaft mehrere Threads zu erzeugen, mit der unangenehmen Nebenwirkung, daß diese alle in der Prozessliste mit dem selben Namen auftauchen. Dies ist aber kein Problem. Mit

>> ./overnetd.sh stop

werden alle wieder gekillt (und der Spuk ist vorbei).

Dedicated game server (Unreal Tournament)

Zweites Fallbeispiel: Der user "horst-kevin" spielt gern UT mit anderen Spielern. Nun möchte er auf seinem älteren Zweitrechner, dessen Resourcen für heutige Verhältnisse recht bescheiden sind, für einen UT dedicated server aber völlig ausreichend sind, einen eben solchen aufbauen. Er macht folgende Änderungen:

```
DAEMON_DESC="UT dedicated server version 436"DAEMON_BIN="/usr/local/games/ut/ucc"
DAEMON_PROC="ucc-bin"
DAEMON_LOGFILE="/home/horst-kevin/${DAEMON_PROC}.log"
DAEMON_OPTIONS1=" server \"DM-Turbine?game=Botpack.DeathMatchPlus\" "
DAEMON_OPTIONS2=" -port=7777 ini=ucc.ini log=ucc.log"
DAEMON_OPTIONS="${DAEMON_OPTIONS1}${DAEMON_OPTIONS2}"
```

Der Server läßt sich problemlos in der Konsole oder remote über ein Netzwerk in einer telnet oder ssh Sitzung Starten und Stoppen wie oben beschrieben.

Fazit

Die Dämonen in einem UNIX-System sollten nun ihren Schrecken verloren haben. Der geneigte Leser sollte darüber hinaus nun in der Lage sein, auch eigene Dämonen zu erschaffen und zu steuern. Das Erstellen beispielsweise eines Quake 3 dedicated servers ist analog zu obigem UT-Beispiel problemlos möglich.

Weitere Quellen:

http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html

https://sites.google.com/site/wzhang85/creatingadaemonindebianlinux

http://esa-matti.suuronen.org/blog/2012/07/22/creating-kiss-daemons-in-linux/

http://www.principlesofprogram.com/concepts/Linux-Daemon