tuebix16-ctlimit

June 12, 2016

1 Kinder, Computerzeitbegrenzung und New-Style-Daemonen

Tübix 2016
Anselm Kruis anselm@kruis.de

2 Table of Contents

  - Generischer New Style Daemon in Python
  - Konfiguration: [configparser](https://docs.python.org/3/library/configparser.html)
  - [Python Logging](https://docs.python.org/3/library/logging.html) für New Style Daemons
  - Das `.service`-File
  - [DBus](https://www.freedesktop.org/wiki/Software/dbus/)
  - DBus mit Python: pydbus
    * GIO / pydbus Einschränkungen
    * Beispiel
    * DBus-Signale
  - Eigene DBus Services
    * Beispiel: Echo-Server
    * System DBus Policy File
    * System DBus Policy File Beispiel
  - Debugging Tips
    * Remote Debugging mit DBus starten

2.1 Wer bin ich

- Diplom Physiker
- Verheiratet, 3 Kinder
- Betruflich seit 2001 bei der science + computing ag in München
- Softwareingenieur, Teamleiter
- Open-Source: ein paar Python Projekte bei github und bitbucket

3 Einführung

Worum geht es?

3.0.1 Das Problem

Haben sie Lust, mit ihren Kindern über die Dauer der Computer Nutzung zu diskutieren?
Wir - meine Frau und ich - hatten keine.

3.1 Das Problem

Ausufernde Computerbenutzung

Randbedingungen

- gemeinsam genutzter Computer
- Kinder haben haben keine root-Rechte

Lösung Ein kleines selbstgeschriebenes Programm: ***ctlimit*** beschränkt die Zeit auf faire Weise.

- Zwangs-Logout sobald die Zeit verbraucht ist
- in Python implementiert
- "new style daemon" (sd-daemon(3))
- Beispiel für Linux Desktop Automatisierung, systemd, dbus, GLib ...

Keine Lösung für

- Smartphone Nutzung
- eigene Windows Laptops der Kinder

3.2 Historie

- erste Version um 2010 herum in Java + Shell
- quick and dirty
- funktionierte seit Debian Jessie (logind statt ConsoleKit) nicht mehr
- DBus Bindings für Java werden nicht gepflegt
- Chrisoph Prokop hat mich überredet, hier einen Vortrag zu halten

Daher:

- 2016 Neuimplementierung in Python als Open-Source Projekt ctlimit

4 Anwendung

Wie nutzt man ctlimit?

4.0.1 Installieren

4.0.2 Konfigurieren

4.0.3 Den Computer machen lassen

4.1 Voraussetzungen

- Linux Distribution mit
- systemd
- logind
- Graphischer Desktop
- GLib Version >= 2.46
- Ubuntu 16.04 Xenial Xerus
- Fedora Core 23
- OpenSuse TumbleWeed
- Gentoo
- Arch Linux
- für Debian Jessie: Patch

4.2 Glib2.48 for Debian Jessie

A DBus server with pydbus needs glib >= 2.46. See 1, 2.
Debian Backports provides glib 2.48 in jessie-backports.

4.2.1 Installation

libglib 2.48 Follow the instructions at http://backports.debian.org/Instructions/. Finally run

```
$ sudo apt-get -t jessie-backports install \
libglib2.0-0:amd64
```

GI type libraries Unfortunately the installing libglib is not sufficient. We need to update the GObject introspection (GI) type libraries for glib. The Debian package is gir1.2-glib-2.0. Its source package is gobject-introspection, which is not part of jessie-backports.
Therefore we rebuild it against the new glib 2.48. It does not require root.

1. Install the dependencies und tools

   ```
   $ sudo apt-get install devscripts
   $ sudo apt-get build-dep gobject-introspection
   $ mkdir tmpdir && cd tmpdir
   $ apt-get source gobject-introspection
   $ apt-get source glib2.0/jessie-backports
   ```

2. Save the glib source directory. Without these sources the GI scanner can't create correct type libraties, because the C-header files don't contain all required information.

   ```
   $ GLIBSRC=$(cd glib2.0-2.48.* && pwd)
   ```

3. Convert the old type library to xml. We need it later to compare the new one against it.

   ```
   $ g-ir-generate /usr/lib/x86_64-linux-gnu/girepository-1.0/Gio-2.0.typelib \
   >Gio-2.0_1.42.0-2.2.xml
   ```

4. Build gobject-introspection

   ```
   $ cd gobject-introspection-1.42*
   $ dch -i 'Rebuild against glib 2.48 from jessie-backports'
   $ DEB_CONFIGURE_EXTRA_FLAGS="--with-glib-src=$GLIBSRC" \
   dpkg-buildpackage -b -us -uc
   ```

5. Compare the new and the old type library

```
   diff $ g-ir-generate Gio-2.0.typelib >../Gio-2.0_1.42.0-3.xml
$ cd ..  $ diff -u -w Gio-2.0_1.42.0-2.2.xml Gio-2.0_1.42.0-3.xml
| grep register_object - <method name="register_object"
c:identifier="g_dbus_connection_register_object" throws="1"> + <method
name="register_object" c:identifier="g_dbus_connection_register_object_with_closure
throws="1">
```

If you don't get this output, the build didn't recognise the glib source. The type libraries in the new package *gir1.2-freedesktop_1.42.0-3_amd64.deb* should be identical to the already installed ones (from *gir1.2-freedesktop_1.42.0-2.2_amd64.deb*). If your in doubt, extract them (`dpkg-deb -x`) and compare the files.

5. Install. You need to install *gir1.2-glib-2.0_1.42.0-3_amd64.deb* and *gir1.2-freedesktop_1.42.0-3_amd64.deb*, because *gir1.2-freedesktop* depends on the corresponding *gir1.2-glib* package.

   ```
   $ sudo dpkg -i gir1.2-glib-2.0_1.42.0-3_amd64.deb
   $ sudo dpkg -i gir1.2-freedesktop_1.42.0-3_amd64.deb
   ```

4.3 Installation von ctlimit

Wie bei eigentlich allen guten Programmen gilt: RTFM

1. Pakete der Linux-Distribution installieren. Benötigt werden:
2. Python ab Version 3.4 mit pip (Debian Paket *python3-pip*)
3. Die Kommandos `play` (Debian Paket *sox*), `dbus-send` (*dbus*), `notify-send` (*libnotify-bin*), `pkg-config` (*pkg-config*).
4. Die Python Extension für GObject Introspection (GI) und die GI-Daten für die Libraries GLib, Gio, Gobject (Debian Pakete *python3-gi* und *gir1.2-glib-2.0*).
5. Mit `pip` *ctlimit* und die noch verbleibenden Abhängigkeiten installieren `$ sudo pip3 install ctlimit`
6. Einrichten `$ sudo python3 -m ctlimit -c /dev/null --install-system-files`

4.4 Konfiguration von ctlimit

Alles als root bzw. mit `sudo`.

1. Konfigurationsdatei editieren: `$EDITOR /etc/ctlimit.conf` Danach `systemctl reload ctlimit.service`
2. ctlimit starten: `systemctl start ctlimit.service`
3. Kontrolle: `journalctl _SYSTEMD_UNIT=ctlimit.service`
4. ctlimit automatisch starten: `systemctl enable ctlimit.service`

Die Kofigurationsdatei /etc/ctlimit.conf INI-File bzw Python Configparser Syntax.
Minimales Beispiel:

```
[DEFAULT]
seconds_per_day: 3600

[Users]
users = bob alice

[alice]
name = qx471193
seconds_per_day: 5400
```

User `bob` hat eine Stunde, User `qx471193` hat 1,5 Stunden.

4.5 Demo

Zeit für eine Demonstration

- Benutzer "kind" in die Konfiguration eintragen
- Benutzer anmelden
- Abwarten

5 Technischer Teil

Warum Daemonen Entwicklung einfach ist

5.1 Anforderungen

1. Technisch saubere Umsetzung als New Style Daemon
2. Möglichst geringer Resourcenverbrauch
3. Nur echte Nutzung erfassen:

- Uhr steht, wenn die virtuelle Console nicht aktiv ist
- Uhr steht, wenn die Session idle ist (z.B. Bildschirmschoner aktiv)

4. Bei Zeitablauf Abmeldung mit Ankündigung

- z.B. optisches und akustisches Signal für 5 Minuten

5. Nutzungsstatus muss einen Reboot überstehen
6. Konfigurierbarkeit

- zu überwachende User
- erlaubte Nutzungsdauer pro Tag

Nice to have: 7. Möglichkeit ctlimit zu beeinflussen, z.B. zusätzliche Zeit zu gewähren 8. Abfrage der aktuellen Nutzungsdauer

5.2 Wie könnte das funktionieren?

Beobachtung: automatische Aktivierung des Bildschirmschoners
==> Desktop hat offensichtlich eine Idle Überwachung
Beobachtung: Bildschirmsperre beim Umschaltung virtueller Konsolen
==> auch hier weiß der Computer, welche aktiv ist
Wenn wir diese Informationen anzapfen können, dann läßt sich eine Zeitbeschränkung einfach realisieren.

Wichtigste Infoquelle https://www.freedesktop.org/

5.3 Wichtige Komponenten (neben ctlimit)

- Systemd
- Das heutige *init* (PID 1)
- Startet und kontrolliert Services, u.a. logind, display manager, system-dbus, ctlimit

- Logind (oder ConsoleKit)
- Verwaltet Sessions
- Kennt Session Properties, insbesondere User, Display, Idle-State

- Screen Saver / Screen Locker
- Erkennt Idle Zustand, meldet ihn weiter an logind (oder ConsoleKit)

- DBus
- Ermöglicht die Kommunikation der Komponenten
- System-Bus und je ein Session-Bus pro Session
- ctlimit nutzt den System-Bus
  - um Informationen vom Logind zu bekommen
  - zur Kontrolle von ctlimit

5.4 Auswahl der Programmiersprache

Oder warum gerade Python? Warum nicht C++, C, Perl, Java, Ruby, Lua, eLisp, Shell, ...

- Ich kann es gut
- Es gibt alle nötigen Bindings
- Es gibt gute IDEs
- Die Sprache ist einfach und gut dokumentiert

5.4.1 Warum GLib / Gio / GObject

- Die DBus Anbindung pydbus benötigt GLib und Gio
- Gute Unterstützung für "systemnahe" Linux Programme
- Einfach anwendbar
- Sehr mächtig

5.5 New Style Daemon

Wird auch **sd-daemon** genannt.

- Dokumentiert in daemon(7) und sd-daemon(3)
- Normaler Unix-Prozess, der einige Konventionen einhält
- Beschreibung durch eine systemd.service(5)-Datei in `/lib/systemd/system/`
- Prozess erfordert keine besondere Initialisierung
- Keine UNIX SysV Daemon-Initialisierung nötig
- Am Ende der Initialisierung Ready-Meldung mit sd_notify(3)
- SIGTERM beendet Prozess
- SIGHUP lädt die Konfiguration neu
- Logging: Ausgabe auf STDERR, Optional "<*LEVEL*>" am Zeilenanfang

5.5.1 Vergleich zum klassischen SysV-Daemon

Zwei forks und umfangreiche Initialisierung aller Prozesseigenschaften nötig - Laut daemon(7) benötigt man 15 Schritte, um einen Daemon zu erzeugen - PEP 3143 – Standard daemon process library, Implementierung bei PyPi python-daemon

5.6 Minimales Beispiel - Pure Python

```
In [ ]: import sys, os, signal, selectors, sdnotify
        signal_received = {}
        systemd_notifier = sdnotify.SystemdNotifier()
        sel = selectors.DefaultSelector()

        def _on_signal(signalnum, frame):
            signal_received[signalnum] = True
            raise InterruptedError
        for s in (signal.SIGTERM, signal.SIGHUP): signal.signal(s, _on_signal)

        # additional setup code, i.e. sel.register(sock, selectors.EVENT_READ, acce

        print("<6> Starting, PID is ", os.getpid(), file=sys.stderr)
        systemd_notifier.notify("READY=1")

        while signal.SIGTERM not in signal_received:
            if signal_received.pop(signal.SIGHUP, False):
                systemd_notifier.notify("RELOADING=1")
                print("<6> reconfig", file=sys.stderr)
                systemd_notifier.notify("READY=1")
            # regular events
            for key, mask in sel.select(): key.data(key.fileobj, mask)

        systemd_notifier.notify("STOPPING=1")
        print("<6> Terminated", file=sys.stderr)
```

5.6.1 sdnotify - Notify Service Manager

Neue C-Funktion sd_notify(3): Notify service manager (systemd) about start-up completion and other service status changes.

Python Implementierung: Module sdnotify.

- Der systemd muss erkennen können, wann ein gestarteter Service betriebsbereit ist
- Service schickt Nachricht `"READY=1"`
- Weitere Nachrichten sind optional
- bei Reload der Konfiguation
- bei Beendigung
- und ander mehr
- Praktisch: `sdnotify.SystemdNotifier().notify("...")` ignoriert Sende-Fehler
- Programm kann auch ohne systemd gestartet werden

Vergleich mit SysV Daemon Betriebsbereitschaft nach erfolgreicher Beendigung des 1. Prozesses.

5.7 Minimales Beispiel - GLib

```
In [ ]: import sys, os, sdnotify
        from signal import SIGHUP, SIGTERM
        from gi.repository import GLib
        mainloop = GLib.MainLoop()
        systemd_notifier = sdnotify.SystemdNotifier()

        def _initialize(is_reconfig=False):
            if is_reconfig:
                systemd_notifier.notify("RELOADING=1")
            print("<6> initializing, is_reconfig:", is_reconfig, file=sys.stderr)
            systemd_notifier.notify("READY=1")
            return is_reconfig

        GLib.unix_signal_add(GLib.PRIORITY_DEFAULT, SIGHUP, _initialize, True)
        GLib.unix_signal_add(GLib.PRIORITY_DEFAULT, SIGTERM, mainloop.quit)
        GLib.idle_add(_initialize)
        print("<6> Starting, PID is ", os.getpid(), file=sys.stderr)
        mainloop.run()
        systemd_notifier.notify("STOPPING=1")
        print("<6> Terminated", file=sys.stderr)
```

5.7.1 GLib in Python

Wikipedia: > GLib ist eine in C geschriebene Bibliothek, die verschiedene Funktionen bereitstellt, die normalerweise in C nur mit sehr viel Aufwand möglich sind.

Bibliotheken - GLib: allgemeine Funktionen - GObject: Objektsystem, Signalsystem, Nachrichten + In Python nicht so wichtiug, Python kann das selbst - GIO (GLib In-

terfaces and Objects): Datei- und Datenstromobjekte, Netzwerkfunktionalität, Interprozess-Kommunikationssystem D-Bus, ...

Python Binding: GObject Introspection

- Bibliotheken haben eine maschinenlesbare Schnittstellenbeschreibung (`.typelib`)
- g-ir-scanner(1) erzeugt GIR-XML Daten aus Source und Header Dateien
- g-ir-compiler(1) erzeugt typelib aus GIR-XML.
- typelib-Repository: `/usr/lib/x86_64-linux-gnu/girepository-1.0/*.typelib`
- Python Extension Module gi ließt `.typelib` und erzeugt dynamisch das Python-Binding
- Funktioniert inzwischen für viele weitere Libraries: Ubuntu hat 202 `gir1.2-.*\.deb`-Pakete

5.7.2 GLib-Mainloop

Typischer Programmaufbau

```
In [ ]: from gi.repository import GLib
        # create the mainloop
        mainloop = GLib.MainLoop()

        # define a callback
        counter = 0
        def count_down(limit):
            global counter
            counter += 1
            print(limit - counter, flush=True)
            return counter != limit  # run until 0
        # add callbacks
        GLib.timeout_add_seconds(1, count_down, 5)
        GLib.timeout_add_seconds(6, mainloop.quit) # terminate the loop
        # run it
        mainloop.run()
```

5.7.3 GLib - wichtigste Funktionen

- `mainloop.run()`: startet die Mainloop
- `mainloop.quit()`: beendet die Mainloop
- `GLib.idle_add()`: Callback wenn idle
- `GLib.timeout_add()`: periodischer Callback
- `GLib.timeout_add_seconds()`: periodischer callback
- `GLib.unix_signal_add()`: Unix Signal Callback

Doku für GI-basierte Python Bindings https://lazka.github.io/pgi-docs/index.html

5.8 Generischer New Style Daemon in Python

Den bisher gezeigten Beispielen fehlt noch einiges. - Aktivierung durch Ereignisse und Timer - Konfiguation - Logging

Modul ctlimit.sddaemon

- Abstrakte Basisklasse `EventDrivenDaemon` zuständig für
- Das sd-daemon Protokoll
- Aktivierung durch Events
- Konfiguration und Persistentz
- Klasse SdDaemonLoggingFormatter
- Python-Logging Ausgabe im sd-daemon Format

5.9 Konfiguration: configparser

Parser für INI-Files. Seit Python 3.2 recht gut brauchbar.

Erwähnenswert: - ExtendedInterpolation verwenden. - Tip: Kommandozeilen Optionen und Environment programmatisch in Sections packen - Interpolation erlaubt dann sehr flexibles Verhalten - Mehrere Konfigurationsdateien in einen configparser einlesen + Grundeinstellungen im Python Package Directory `ctlimit_default.conf` + User-Konfiguration in `/etc/ctlimit.conf` + Zustandsinformation `/var/lib/ctlimit/ctlimit.state` - Sections mit Zustand speichern in `/var/lib/ctlimit/ctlimit.state` - Logging wird mit dem selben configparser-Object konfiguriert

5.10 Python Logging für New Style Daemons

Spezielle Konfiguaration zur Ausgabe der Meldungen mit Prefix auf STDERR. - Handler: logging.StreamHandler - Formatter: eigene Klasse `ctlimit.SdDaemonLoggingFormatter`.

Logging Konfiguration im configparser-Format:

```
[loggers]
keys=root
[handlers]
keys=sd_daemon
[formatters]
keys=sd_daemon
[logger_root]
level=${common:log_level}
handlers=sd_daemon
[handler_sd_daemon]
class=StreamHandler
level=DEBUG
formatter = sd_daemon
args=(sys.stderr,)
[formatter_sd_daemon]
format=%(levelname)s: %(message)s
class=ctlimit.SdDaemonLoggingFormatter
```

Weitergehende Infos: https://www.loggly.com/blog/logging-in-new-style-daemons-with-systemd/

5.11 Das `.service`-File

Macht aus einem Programm einen systemd-service. Dokumentation: systemd.service(5) und systemd.directives(7).

Beispiel: `/lib/systemd/system/ctlimit.service`

```
[Unit]
Description=ctlimit computer time limitation service
Documentation=https://doc.for.ctlimit

[Service]
Type=notify
NotifyAccess=main

ExecStart=/usr/bin/env python3 -m ctlimit
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
TimeoutStopSec=5

SyslogIdentifier=ctlimit
StandardOutput=syslog
StandardError=inherit

[Install]
WantedBy=multi-user.target
```

5.12 DBus

Basics - Erlaubt Nachrichtenaustausch zwischen Programmen + Aufruf von Methoden, Abfrage von Properties + Empfang von Signalen + Interfaces als Namensräume - Verwaltet Namen und Objekt-Pfade - Kann Server automatisch starten - Policy regelt, welche Nachrichten transportiert werden

5.12.1 Entwicklungstools DBus

- dbus-send: Kommandozeilen-Tool
- d-feet: Graphisches Pondon zu `dbus-send`
- dbus-monitor: Tool zum Sniffen

Achtung: zum Sniffen am System DBus müssen die Sicherheitseinstellungen gelockert werden. - Weitergehende Hinweise: https://wiki.ubuntu.com/DebuggingDBus - Zum Testen: https://www.piware.de/2012/09/announcing-d-bus-mocker-library/

5.13 DBus mit Python: pydbus

pydbus: high level pythonic DBus library based on GIO. - sehr einfache Anwendung - noch recht jung - Könnte die Zukunft der DBus-Programmierung sein

Alternativen Siehe https://wiki.python.org/moin/DbusExamples

- dbus-python: der Klassiker
- txdbus: DBus für das Twisted Framework (Pure-Python)
- GIO: da kann man gleich pydbus nehmen

Probleme

- Asynchrone Arbeitsweise (Standard bei GLib/GIO) nicht unterstützt
- Teil der GIO DBus Funktionalität ist nicht nutzbar

5.13.1 GIO / pydbus Einschränkungen

- GIO Klasse GDBusObjectManagerServer ist nicht introspection friendly.
- Problem ist die Interface Funktion g_dbus_interface_skeleton_get_vtable()
- Folge: Standard-Interface org.freedesktop.DBus.ObjectManager kann nicht verwendet werden
- pydbus arbeitet synchron. Die asynchrone Arbeitsweise der GLib / GIO wird nicht unterstützt
- Erstellung eigener Services mit pydbus ist noch unausgereift
- Authentisierung des Aufrufers nicht möglich
- pydbus bietet noch keine Policy-Kit Unterstützung

5.13.2 Beispiel

Alle Sessions für den Benutzer “anselm” ausgeben

```
In [ ]: import pydbus
        bus = pydbus.SystemBus()

In [ ]: manager = bus.get(".login1")[".Manager"]

In [ ]: for session_infos in manager.ListSessions():
            session = bus.get(".login1", session_infos[4])
            print(session.Id, " User:", session.Name, " Type:", session.Type,
                  " State:", session.State," Service:", session.Service)

In [ ]: session.Name
```

5.13.3 DBus-Signale

- sind asynchron eintreffende Nachrichten (Events) auf dem DBus
- werden mit einem Signal-Handler-Funktion bearbeitet
- Beispiel:

```
In [ ]: from gi.repository import GLib
        mainloop = GLib.MainLoop()

In [ ]: manager = bus.get(".login1")[".Manager"]
```

```
In [ ]: manager.onSessionNew = print
        GLib.timeout_add_seconds(30, mainloop.quit)
        mainloop.run()
        manager.onSessionNew = None
```

DBus-Signale ermöglichen es, auf externe Ereignisse zu reagieren.

5.14 Eigene DBus Services

Das Kind war brav und soll ein Bisschen zusätzliche Zeit bekommen. Was tun? - Konfigurationsdatei /etc/ctlimit.conf editieren - systemctl reload ctlimit.service - Vergessen die Verlängerung zurückzunehmen

Oder:

```
$ dbus-send --system --type=method_call --print-reply \
> --dest=de.kruis.ctlimit1 "/de/kruis/ctlimit1/User$(id -u kind)" \
> de.kruis.ctlimit1.User.IncreaseCurrentLimit \
> int32:1800
```

(Das lässt sich auch gut vom Smatphone aus machen.)

5.14.1 Beispiel: Echo-Server

```
In [ ]: from pydbus import SessionBus
        from gi.repository import GLib
        class EchoServer:
            """<node>
                <interface name='de.kruis.demo'>
                  <method name='Echo'>
                    <arg type='s' name='arg' direction='in'/>
                    <arg type='s' name='response' direction='out'/>
                  </method>
                  <method name='Quit'/>
                </interface>
              </node>"""
            def Echo(self, arg):
                print("Echo(%r)" % arg, flush=True)
                return arg
            def Quit(self):
                self.mainloop.quit()
            def run(self):
                self.mainloop = GLib.MainLoop()
                with SessionBus().publish("de.kruis.demo", self):
                    self.mainloop.run()

In [ ]: EchoServer().run()
```

5.14.2 System DBus Policy File

- `/etc/dbus-1/system.d/`*Service-Name*`.conf`
- Konfiguration der welche Nachrichten auf dem Bus erlaubt sind.
- Dokumentiert in dbus-daemon(1)
- Das KDE Projekt hat ein Tutorial

5.14.3 System DBus Policy File Beispiel

```
<busconfig>
  <!-- required -->
  <policy user="root">
    <allow own="de.kruis.ctlimit1"/>  <!-- service name, always required -->
    <!-- Allow a message to our service from a root user -->
    <allow send_destination="de.kruis.ctlimit1"
           send_interface="de.kruis.ctlimit1.User"
           send_member="SetCurrentLimit"/>
  </policy>

  <!-- optional: policy for a unix group -->
  <policy group="parents"> ... </policy>

  <!-- usually required: policy for other -->
  <policy context="default">
    <deny send_destination="de.kruis.ctlimit1"/>
    <allow send_destination="de.kruis.ctlimit1"
           send_interface="org.freedesktop.DBus.Introspectable"/>
    ...
    <!-- allow answering -->
    <allow receive_sender="de.kruis.ctlimit1"/>
    <!-- allow receiving signals and answers to method calls made by ctlimit -->
    <allow send_destination="de.kruis.ctlimit1" send_type="signal"/>
    <allow send_destination="de.kruis.ctlimit1" send_type="method_return"/>
    <allow send_destination="de.kruis.ctlimit1" send_type="error"/>
  </policy>
</busconfig>
```

5.15 Debugging Tips

- pdb ist nicht optimal
- Es gibt eine IDE Liste
- Ich verwende PyDev

Remote Debugging Fast immer wenn es in der IDE nicht mehr weitergeht hilft Remote Debugging. - Geeigneter Debugger Pydev.Debugger, enthalten in Eclipse Pydev, LiClipse und PyCharm - Modul pydevd verfügbar machen, z.B. `pip install pydevd` - An Debugger verbinden: `import pydevd; pydevd.settrace(...)`

DBus

- Beim Entwickeln / Coden möglichst den Session Bus verwenden
- Alternativ einen eigenen DBus starten: `dbus-run-session [--config-file FILENAME] PROGRAM [ARGUMENTS...]`
- Tools: dbus-monitor(1), dbus-send(1), d-feet(1)

5.15.1 Remote Debugging mit DBus starten

Über den System-DBus aufrufbare Methode

```
...
<method name='AttachPydevd'>
   <arg type='s' name='path' direction='in'/>
   <arg type='s' name='kwargs_json' direction='in'/>
</method>
...
```

```
def AttachPydevd(self, path, kwargs_json):
    if path and path not in sys.path:
        sys.path.insert(0, path)

    import pydevd
    kwargs = json.loads(kwargs_json or "{}")
    pydevd.settrace(**kwargs)
```

Aufruf

```
$ PYDEVD_DIR=$(find $(dirname $(which eclipse)) -type d -name pysrc)
$ sudo dbus-send --system --type=method_call \
> --dest=de.kruis.ctlimit1 "/de/kruis/ctlimit1" \
> de.kruis.ctlimit1.Daemon.AttachPydevd \
> string:"$PYDEVD_DIR" \
> string:'{"stderrToServer": true, "stdoutToServer": true}'
```

6 Fragen?

und Antworten