

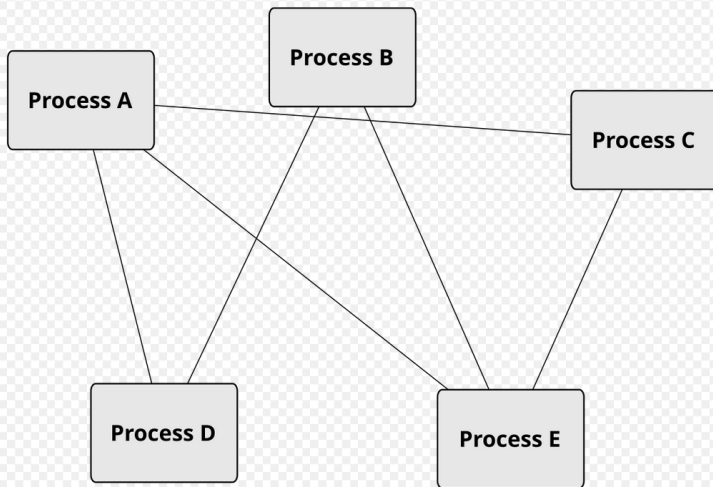
# Komunikacja międzyprocesowa z D-Busem

23 listopada 2024

# Plan

- Historia
- Teoria
- Praktyka

# Komunikacja międzyprocesowa



# Przykłady

- Pamięć współdzielona
- Sockety
- Pipe, named pipe

# Wysokopoziomowe rozwiązania

- CORBA, Common Object Request Broker Architecture
  - Skomplikowany, złożony standard
  - Transparentność lokacji
  - Kompatybilność
- DCOP, Desktop COmmunication Protocol
  - Część KDE...
  - ...do czasu D-Busa, KDE 4

Projekt z 2002 roku zarządzający między innymi:

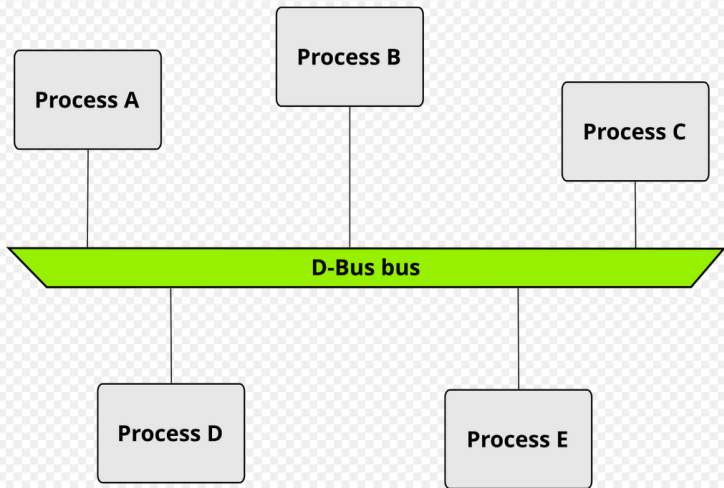
- PulseAudio
- systemd
- Wayland
- Mesa

Potrzeba zaimplementowania ustandaryzowanego, bezpiecznego IPC dla środowisk graficznych.

2002 - początek projektu

2006 - stabilna wersja

# Działanie





# Zalety

- Prosty
- Dostępność standardu w wielu językach programowania
- Security policy
- Abstrakcja połączenia
- Walidacja typów

# D-Bus bus

Demon do którego łączą się aplikacje. Przekierowuje wiadomości od aplikacji do innych aplikacji. Zaimplementowany w libdbus.

# System bus

Z poziomu użytkownika, osobna instancja dla każdego.

- GUI KDE, GNOME
- Aplikacje, Spotify, Firefox
- PulseAudio

Z poziomu systemu operacyjnego.

- Sieci, NetworkManager
- Urządzenia, UDisks, USB
- Uprawnienia, PolicyKit

# Połączenie serwisu

Następuje przy połączeniu do demona. Serwis ma przyznaną nazwę:

- Unikalna: :1-37, :1-42
- Well-known name: org.Cinammon,  
org.mpris.MediaPlayer2.spotify
- Znak : zarezerwowany

- Wymaga określenia bus name
- Ścieżka do obiektu jako nazwa:
  - /org/Cinnamon
  - /com/Test
  - /org/meks/Logger/1 org/meks/Logger/2
- Implementuje interfejsy
- Implementuje sygnały i metody

# Interfejsy

Zbiór metod, sygnałów oraz właściwości (properties).  
Implementowane przez obiekty. Coś jak abstract class.

# Proxy

Obiekt tworzony przez klienta jako reprezentacja zdalnego obiektu innego procesu. Niskopoziomowo tworzy wiadomość do serwisu z requestowanym wykonaniem metod i zwraca odpowiedź.

```
Proxy proxy = new Proxy(getBusConnection(),  
                        "/remote/object/path");  
Object returnValue = proxy.MethodName(arg1, arg2);
```



# Przykład

Address: unix:path=/run/user/1000/bus  
Name: org.mpris.MediaPlayer2.spotify  
Unique name: :1.255

Object path

▶ Match rules

▶ Statistics

▼ /org/mpris/MediaPlayer2

▼ Interfaces

- ▶ org.freedesktop.DBus.Introspectable
- ▶ org.freedesktop.DBus.Peer
- ▶ org.freedesktop.DBus.Properties
- ▶ org.mpris.MediaPlayer2
- ▼ org.mpris.MediaPlayer2.Player

▼ Methods

- Next () ⇌ ()
- OpenUri (String Uri) ⇌ ()
- Pause () ⇌ ()
- Play () ⇌ ()
- PlayPause () ⇌ ()
- Previous () ⇌ ()
- Seek (Int64 Offset) ⇌ ()
- SetPosition (Object Path TrackId, Int64 Position) ⇌ ()
- Stop () ⇌ ()

▼ Properties

- Boolean CanControl (read)
- Boolean CanGoNext (read)
- Boolean CanGoPrevious (read)
- Boolean CanPause (read)
- Boolean CanPlay (read)
- Boolean CanSeek (read)
- Boolean Shuffle (read / write)
- Dict of (String, Variant) Metadata (read)
- Double MaximumRate (read)
- Double MinimumRate (read)
- Double Rate (read / write)
- Double Volume (read / write)
- Int64 Position (read)
- String LoopStatus (read / write)
- String PlaybackStatus (read)

▼ Signals

- Seeked (Int64)

# System typów

Parę przykładów:

- `aai` - tablica tablic intów
- `a(ss)` - array structów z dwoma stringami
- `a{sd}` - mapa klucz-string value-double
- `v` - variant, dynamiczny typ
- `a{sv}` - mapa z wartościami o różnych typach

# Interfejsy standardowe

Interfejsy pozwalające badać właściwości serwisów. Między innymi:

- `org.freedesktop.DBus.Peer` - metody `Ping` i `GetMachinelid`
- `org.freedesktop.DBus.Introspectable` - reprezentacja obiektu w XML
- `org.freedesktop.DBus.Properties` - manipulacja właściwościami, odczyt wartości
- `org.freedesktop.DBus.ObjectManager`

# XML

```
<node>
  <interface name="org.meks.Logger">
    <method name="AddLog">
      <arg direction="in" type="s" name="log_str" />
      <arg direction="out" type="s" />
    </method>
    <signal name="CountChange">
      <arg direction="out" type="i" name="count" />
    </signal>
    <property name="Upper" type="b" access="readwrite"/>
    <property name="LogCount" type="i" access="read" />
    <property name="LogFileName" type="s" access="readwrite"/>
  </interface>
</node>
```

# Co możemy implementować?

Istnieje wiele interfejsów z możliwością implementacji. Przykładowe API do implementacji:

- Standard MPRIIS-odtwarzacze muzyki
- Więcej na: <https://specifications.freedesktop.org/>

- dbus-monitor - debugger do monitorowania wiadomości dbusa
- dbus-send - wysyłanie wiadomości do busy
- d-feet - graficzny debugger do sprawdzania serwisów i wysyłania wiadomości
-

# Implementacje, Bindingi

Freedesktop.org oferuje implementacje i bindingi protokołu w D-Bus. Najciekawsze:

- pydbus - wysokopoziomowa implementacja w pythonie, nie mylić z dbus library!
- zbus - implementacja w rust
- libdbus - niskopoziomowe API w C, nie polecane do pisania prostych aplikacji
- GDBus - implementacja w C, już lepiej w tym pisać. Ale kod lepiej wygenerować

- <https://en.wikipedia.org/wiki/D-Bus>
- <https://dbus.freedesktop.org/doc/dbus-tutorial.html>
- <https://www.freedesktop.org/wiki/IntroductionToDBus>
- <https://dbus.freedesktop.org/doc/dbus-python>
- <https://github.com/zyga/dbus-python/tree/master>