System-wide kdbus transport for DBus

Background

The main goal of the project is to replace DBus (main IPC system on Tizen platform) socket-based transport layer with kdbus (low-level, native kernel IPC transport) based transport.

One of the main requirements is to retain maximal possible compatibility with native DBus solution. In the optimal scenario existing applications should be able to work with the new transport without any modification while benefiting from improved performance, stability and security.

During brainstorming session some architecture proposals were developed and evaluated. We are aware that the solution being chosen for developed and evaluation is not a complete one. It may exist and work fine in Tizen environment while it is much too limited at the moment to be a generic solution.

Solution architecture

Primary assumption is that it is required and sufficient to enable new transport for applications using either libdbus (and other libraries built on top of it) or glib (libgio). Our solution includes modifications for both client libraries and DBus daemon. New transport type is “kdbus:”.

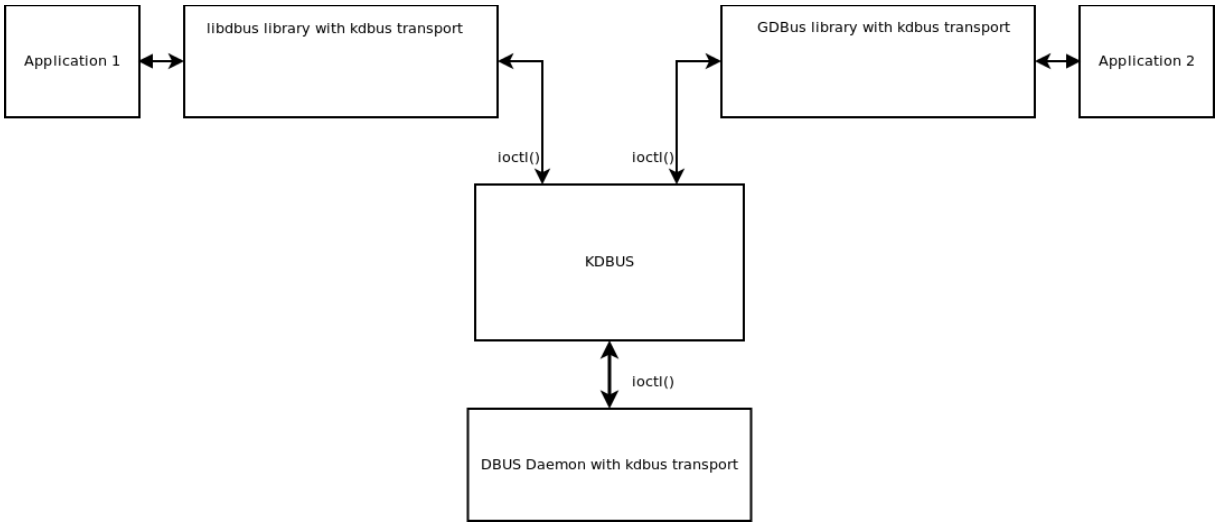
DBus daemon is as stateless as possible. Although it owns and holds a bus in kdbus and handles requests to *org.freedesktop.DBus* interface, yet the operations are translated to ioctl() and performed by kdbus and the required data (like registered names, etc.) is retrieved from kdbus.. Only the *Hello* message is intercepted and handled by transport layer. The same time daemon is not involved in message routing as this is handled directly in kdbus. The transport layer does a conversion between DBus and kdbus address formats.

Advantages:

- daemon remains on the bus and handles org.freedesktop.DBus calls,
- libraries modifications scope can be discussed with projects maintainers,
- no need to filter out org.freedesktop.DBus messages.

Disadvantages:

- every dbus library/wrapper need to be modified, which can be highly time consuming or even impossible (for generic solution),
- daemon executes calls on behalf of other processes,
- it is not possible to send *Hello* message on behalf of other process - it must be implemented (captured and handled) in each modified library.



Kdbus

We have tried to use a mainstream kdbus version, however for easiness of the solution development some minor changes were necessary. The same time bug fixes created during development are being delivered to the mainstream.

One important change which is kept local is an application of so called “compatibility package” which allows our kdbus to work with kernel 3.8.2 used on Tizen platform of our interest. A small patch (included in kdbus source tree) has to be applied to 3.8.2 kernel sources.

Another extension is a possibility to register well-known names as starters, which allows to autostart activatable services in the same way as it is in DBus – by simply sending regular message to them. In kdbus there is possibility to register connection as a starter, which seemed not to fit the needs, so the same mechanism was extened on names.

Known limitations

- Broadcast messages filtering (matching to recipients) is limited. Only keys “sender” and “interface” of match rules set by org.freedesktop.DBus.AddMatch method are used to create rules in kdbus. The rest of keys are ignored. This is because kdbus is only transport layer and in fact doesn’t know and doesn’t look into dbus messages being transported.
- Policies in kdbus are less sophisticated than in DBus and don’t reproduce a lot of DBus abilities. In kdbus it is only possible to set “allow” policies that grants owning, receiving and sending rights to user id, group id or everyone. As the well-known name acquiring is done through org.freedesktop.DBus interface implemented in dbus-daemon, DBus policies related to “own” rights are applied by daemon, but routing is done outside daemon so only simple kdbus polices can be implemented there.