Uplift your Linux systems programming skills with systemd and D-Bus

FOSDEM 2020, Go Devroom (2nd Feb, 2020)

Leonid Vasilyev, <vsleonid@gmail.com>

Agenda

- Scope of this talk
- What is D-Bus/What is systemd?
- How Linux distros use them?
- How to use D-Bus/systemd in Go?
- What interesting can be done with D-Bus/systemd?
- Is it worth it?

Scope

- Systems programming
  - *“Software that provides services for other (application) software” [wikipedia]*
- Go developer POV, not sysadmin
  - (Develop/Test/Debug cycle)
  - (NOT how to configure systemd/D-Bus, containers, etc.)
- Modern Linux
  - (Think most recent stable release of your Linux distro)

What is D-Bus?

- Freedesktop.org specification, started in 2003
  - Core Protocol: Types system / wire format / auth / introspection / properties
  - Message Bus: Naming / well known busses / message routing / standard interfaces
- Reference implementation: *libdbus*, *dbus-daemon*
  - Many alternative implementations of core protocol:
    - sd-bus (used by systemd)
    - godbus (Go native implementation)
  - Not that many of message bus:
    - dbus-broker

What is systemd?

- Started in 2010 as a SysVinit replacement, but expanded to much more
- Many mainstream Linux distros have it as a default
  - Even LFS (Linux From Scratch) has systemd version ;)
- Provides all API via D-Bus
  - Read src/core/dbus.c to understand what it provides exactly

Linux Session Setup

- Implemented by pam_systemd(8) and systemd-logind.service(8)
- We'll be using `--session` bus and `--user` systemd

```
$ systemd-cgls --unit user.slice
Unit user.slice (/user.slice):
└─user-1000.slice
 ├─user@1000.service
 │├─init.scope
 ││├─1248 /lib/systemd/systemd --user
 ││└─1249 (sd-pam)
 │└─dbus.service
 │ └─9733 /usr/bin/dbus-daemon --session --address=systemd:…
 └─session-3.scope
  ├─1246 sshd: vagrant [priv]
  ├─1324 sshd: vagrant@pts/0
  ├─1325 -bash
```

Linux Session Setup

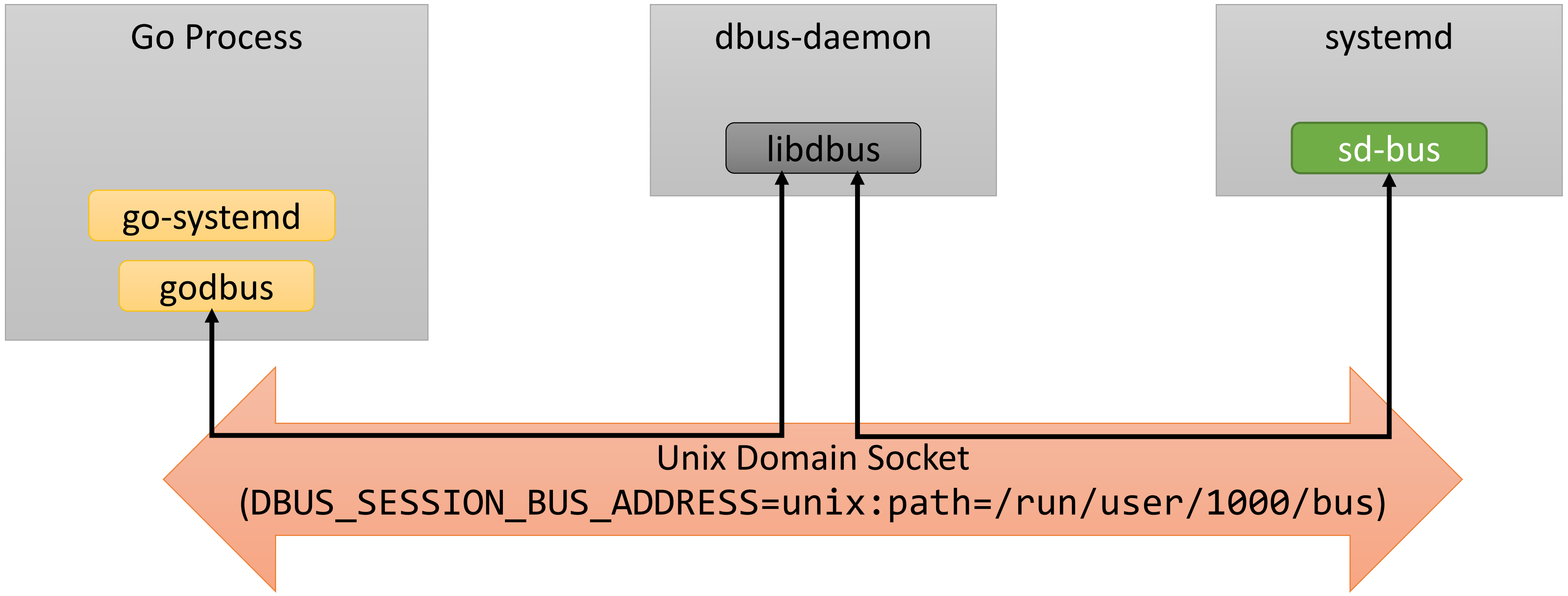
- No root required (aka “rootless”)

```
$ pstree -slap vagrant
sshd,1324
  └─bash,1325
      └─pstree,9755 -slap vagrant

systemd,1248 --user
  ├─(sd-pam),1249
  └─dbus-daemon,9733 --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
```

Go D-Bus/systemd Architecture

- godbus/dbus
- coreos/go-systemd



D-Bus: address format

A...	**is identified by**	**which looks like...**	**and is chosen by**
Bus	address	`unix:path=/var/run/dbus/sys_bus_socket`	system configuration
Connection	bus name	`:34-907` (unique) or `com.mycompany.TextEditor` (well-known)	D-Bus (unique) or the owning program (well-known)
Object	path	`/com/mycompany/TextFileManager`	the owning program
Interface	interface name	`org.freedesktop.Hal.Manager`	the owning program
Member	member name	`ListNames`	the owning program

* source: https://www.freedesktop.org/wiki/IntroductionToDBus/

D-Bus tools: **dbus-send**

```
$ dbus-send --session --print-reply --type=method_call
            --dest=org.freedesktop.DBus / org.freedesktop.DBus.ListNames

array [
  string "org.freedesktop.DBus"
  string "org.freedesktop.systemd1"
  string ":1.0"
  string ":1.9"
]
```

D-Bus tools: **`busctl`**

- Part of systemd
- Can do same stuff as dbus-send

```
$ busctl --user tree org.freedesktop.DBus
└─/org/freedesktop/Dbus
…

$ busctl --user introspect org.freedesktop.DBus /org/freedesktop/Dbus
NAME                                  TYPE      SIGNATURE RESULT/VALUE
org.freedesktop.DBus.Peer             interface -         -
.GetMachineId                         method    -         s
.Ping                                 method    -         -

org.freedesktop.DBus.Debug.Stats      interface -         -
.GetStats                             method    -         a{sv}
```

Godbus/bus: addressing

- Uses reflections heavily
- Easy to make it panic

```go
conn, err := dbus.SessionBus()
if err != nil {
    log.Fatalf("can't connect: %v", err)
}
defer conn.Close()

obj := conn.Object("org.freedesktop.DBus", "/")
call := obj.Call("org.freedesktop.DBus.ListNames", 0)

var result []string
if err := call.Store(&result); err != nil {
   log.Fatalf("can't complete the call: %v", err)
}
log.Printf("Call returned: %+v", result)
```

D-Bus message format

- Binary format
- Supports container types: structs, arrays, dict
- Extra: variant type, file descriptors(!)

```
yyyyuua(yv)

BYTE, BYTE, BYTE, BYTE, UINT32, UINT32, ARRAY of STRUCT of (BYTE,VARIANT)
```

Godbus/bus: **`Message`** type (header)

```
dbus.Message{
  Type: dbus.TypeMethodCall,

  Headers: map[dbus.HeaderField]dbus.Variant{
    dbus.FieldDestination: dbus.MakeVariant("org.freedesktop.Notifications"),
    dbus.FieldPath:        dbus.MakeVariant(dbus.ObjectPath("/org/freedesktop/Notifications")),
    dbus.FieldInterface:   dbus.MakeVariant("org.freedesktop.Notifications"),
    dbus.FieldMember:      dbus.MakeVariant("Notify"),
    dbus.FieldSignature:   dbus.MakeVariant(dbus.ParseSignatureMust("susssasa{sv}i")),
  },
…
```

Godbus/bus: **`Message`** type (body)

```
dbus.Message{
…
  Body: []interface{}{
    "app_name",
    uint32(0),
    "dialog-information",
    "Notification",
    "This is the body of a notification",
    []string{"ok", "Ok"},
    map[string]dbus.Variant{
      "sound-name": dbus.MakeVariant("dialog-information"),
    },
    int32(-1),
  },
}
```

D-Bus Introspection (XML)

- Introspection done via standard interface – `org.freedesktop.DBus.Introspectable`

```
<node>
  <interface name="org.freedesktop.DBus">
    <method name="Hello">
      <arg direction="out" type="s"/>
    </method>
        …
  <signal name="NameLost">
    <arg type="s"/>
  </signal>
        …
  <property name="Features" type="as" access="read">
    <annotation name="org.freedesktop.DBus.Property.EmitsChangedSignal" value="const"/>
  </property>
</node>
```

Godbus/bus: Exporting Objects (1)

- Server code needs to request a bus name
- Exporting object doesn’t not involve dbus-daemon

```
w := Worker{}

// Export object on the bus
conn.Export(w, "/", "com.github.lvsl.Worker")
conn.Export(introspect.Introspectable(intro), "/", "org.freedesktop.DBus.Introspectable")

// register on a bus
reply, err := conn.RequestName("com.github.lvsl.Worker", dbus.NameFlagDoNotQueue)
if err != nil {
  log.Fatalf("can't request a name: %v", err)
}

if reply != dbus.RequestNameReplyPrimaryOwner {
  log.Fatalf("name taken?")
}
```

Godbus/bus: Exporting Objects (2)

```
const intro = `
<node>
  <interface name="com.github.lvsl.Worker">
    <method name="DoWork">
      <arg direction="out" type="s"/>
    </method>
  </interface>` + introspect.IntrospectDataString + `</node>`

type Worker struct{}

func (w Worker) DoWork() (string, *dbus.Error) {
        token, err := uuid.NewRandom()
        if err != nil {
                return "", dbus.MakeFailedError(err)
        }

        // schedule some work here ...

        return token.String(), nil
}
```

D-Bus Signals

- Implement 1:N PubSub
- Async
- Must request to get messages first via Match Rules

D-Bus Best Practices

- Chrome OS D-Bus best practices
  - Avoid changing APIs/properties/complex object hierarchies
  - Use Protobuf for complex messages (?)
  - Don’t use dbus-daemon service activation
- How to Version D-Bus Interfaces
  - Version everything: service name, interface, object path

systemd

- Systemd operates with units (service, scope, etc.)
- Jobs are executed on units
- Units implement D-Bus interfaces
- Units have states
- Changing states emits D-Bus signals

```
$ busctl --user tree org.freedesktop.systemd1
$ busctl --user introspect org.freedesktop.systemd1 /org/freedesktop/systemd1
$ busctl --user introspect org.freedesktop.systemd1
                           /org/freedesktop/systemd1/unit/dbus_2eservice
```

Coreos/go-systemd: List Units

```
conn, err := dbus.NewUserConnection()
if err != nil {
  log.Fatalf("can't connect to --user systemd: %v", err)
}
defer conn.Close()

units, err := conn.ListUnits()
if err != nil {
  log.Fatalf("can't list units: %v", err)
}

log.Printf("Loadede units: %+v", units)
```

systemd: Creating a Transient Unit

- Transient unit created dynamically (not as files on disk)
- Similar to what `systemd-run --user` does

```
$ systemd-run --user env
Running as unit: run-r8f98f7c4d7214558996ed7612b3ba2f2.service

$ journalctl --user -u run-r8f98f7c4d7214558996ed7612b3ba2f2.service
```

Coreos/go-systemd: Transient unit

```
conn, err := dbus.NewUserConnection()
if err != nil {
  log.Fatalf("can't connect to --user systemd: %v", err)
}
defer conn.Close()

jobDone := make(chan string)
props := []dbus.Property{} // TODO: fill these in
jobid, err := conn.StartTransientUnit("coolunit.service", "fail", props, jobDone)
if err != nil {
  log.Fatalf("can't list units: %v", err)
}

log.Printf("Started job: %v", jobid)

status := <-jobDone

log.Printf("Job done: %+v", status)
```

Is it worth it?

Pros :

- Has stable API
- Is commonly available
- Is well understood
- Has Tools to dev/test/debug
- Deep integration with Linux
  - AppArmor, SELinux, UNIX permissions

Cons :

- Has some outdated semantics
- Has legacy features
- Has some outdated docs
- Dynamic typing