Why from an RTOS to Linux?

> The requirements of your embedded system are growing rapidly, and many of them are not truly real-time.
Why from an RTOS to Linux?

- You need to support complex applications with requirements for network and peripheral connectivity (bluetooth, wi-fi, usb...).
Why Linux, from an RTOS?

> You will get an increasing amount of control and choice over practically everything, including
- operating system
- toolchain and programming languages
- middleware
- application frameworks.
Why Linux?

> No need to reinvent the wheel, you can reuse
  – existing hardware drivers
  – existing libraries
  – existing tools and applications.

> Some of these are available as industry-standard configurations, like GENIVI.
Reuse wisely!

> The amount of choice can be staggering, picking the right middleware and libraries requires time and analysis.

> It's important to be aware of the various licensing options of existing components.
Caveats of Linux

> Linux doesn't magically make your hardware do its own bringup.

> Linux doesn't magically make your board support package work.

> Linux doesn't magically make your BSP optimized.
Caveats of Linux

> The Linux distributions are different, be aware of packaging differences, init/upstart/systemd differences, choose a target distribution, and choose wisely.
Caveats of Linux

> Hardware-accelerated graphics require BSP work and often depend on closed-source components.

> Hardware-accelerated video requires BSP work, pipeline optimization and often depends on closed-source components.
When developing with an RTOS starts feeling painful,

- having to reinvent wheels that already exist for Linux.
- missing deadlines due to sheer complexity of such reinvented wheels.
- you have MMU-equipped hardware that is capable of running Linux.
When not?

> If you're running a strictly-real-time system, Linux may not be any easier.

> If you don't have an MMU, Linux might not be a good choice.

> Even if you can run Linux on a toaster, perhaps you shouldn't.
How?

> One reasonable option is doing a gradual port.
  – most of the architecture of the system remains as is, and is ported as directly as possible.

> Another, perhaps more time-consuming option (at least initially) is to fix the architecture first.
  – time-to-market will be longer initially.
"Fix the architecture"?

> Many legacy RTOS systems are so bare-bones that the middleware, or even applications, do memory/resource management and scheduling.

> Legacy systems also often do these things in an unfortunately system-dependent fashion.
"Fix the architecture"?

> Fixing the system dependencies is the first thing to do.

> Allowing an operating system to do the resource management and scheduling is the second
Ok, what about the gradual port?

> Don't change the module division and architecture of your RTOS application stack in the first phase.

> Port the application stack as directly as possible.

> Optimize when the basic port is done, change the architecture gradually to be more suitable for a "full OS" environment.
What about my RTOS?

> Don't "burn the platform".

> Porting things to Linux can be very easy. Maintaining architectural compatibility with a legacy system can be very hard.
Existing competences

- Driver/BSP competence transfers well, although the driver model takes some getting used to.
- Domain knowledge transfers well.
Existing competences

> Applications and middleware may require adopting a new thought model.

> The design approach is different when going from strict real-time to general-purpose programming.
Horror stories...

> We have seen applications "ported" from legacy RTOSes to Linux that consisted of a single binary handling **everything** in an embedded device, weighing in at 100 megabytes in executable size, doing custom memory management, custom threading and scheduling.
...and silver linings

> The aforementioned applications can be made to work for a quick time-to-market port.
> With further development, they can be made sane.
> The end result is an application stack that is easier to develop further and maintain.
Conclusions

> Adopting Linux can speed up development, and ease maintenance.
> Linux is not a free lunch.
> Going from an RTOS to Linux may be a one-way journey.
> Often the journey is well worth it.
SERIOUS ABOUT SOFTWARE