



Zephyr Project: O RTOS de código aberto da The Linux Foundation

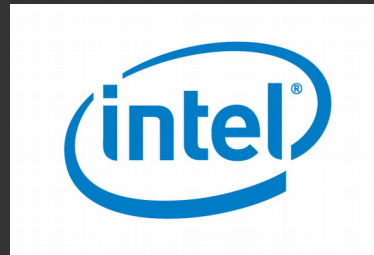
Felipe Neves – Venturus Instituto de Tecnologia

felipe.neves@venturus.org.br
ryukokki.felipe@gmail.com



Breve histórico

- Tudo começa com Intel e Wind River:



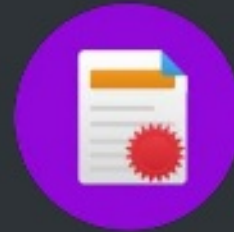
- ...em seguida entra a the Linux Foundation:



Dentre os RTOS/Linux existentes, por que o Zephyr?



Strategic Investment



Permissively Licensed



Modular



True Open Source
Development and
Governance



Zephyr, por dentro do kernel



Zephyr: Por dentro do kernel

- Flexível em termos de configuração;
- Política de execução preemptiva;
- Suporte a execução cooperativa;
- Recursos definidos em tempo de compilação.



Zephyr: Por dentro do kernel

```
.config - Zephyr Kernel Configuration

Zephyr Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features.
Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded
<M> module < > module capable

Architecture (ARM architecture) --->
[ ] Simple system fatal error handler
[*] Use generated IRQ tables
[*]   Generate an interrupt vector table
[*]   Generate a software ISR table
ARM SoC Selection (Kinetis K6x Series MCU) --->
ARM Options --->
Board Selection (Freescale FRDM-K64F) --->
Board Options ----
General Kernel Options --->
Device Drivers --->
Compile and Link Features --->
Debugging Options --->
System Monitoring Options --->
Boot Options --->
Cryptography ----
C Library --->
Additional libraries --->
File System --->
[ ] Bluetooth support ----
Disk --->
Networking --->
Logging Options --->

+ (+)

<Select> < Exit > < Help > < Save > < Load >
```



Zephyr: Por dentro do kernel

- Como todo bom RTOS oferece serviços básicos:
- Gerenciador de Threads;
- Sincronização e IPC;
- Gerenciador de interrupções;
- Nativamente oferece serviços de gerenciamento de energia.



Acessando o hardware, device drivers



Device drivers:

- O Zephyr oferece um modelo comum a todas as arquiteturas para acesso aos dispositivos;
- Não utiliza POSIX-sycalls, ex: `open()`, `write()`, `ioctl()`;
- O suporte vem crescendo com os releases, figurando desde periféricos da SoC escolhida, até circuitos integrados externos;



Device drivers: Representação genérica

```
typedef int (*subsystem_do_this_t)(struct device *device);
typedef void (*subsystem_do_that_t)(struct device *device);

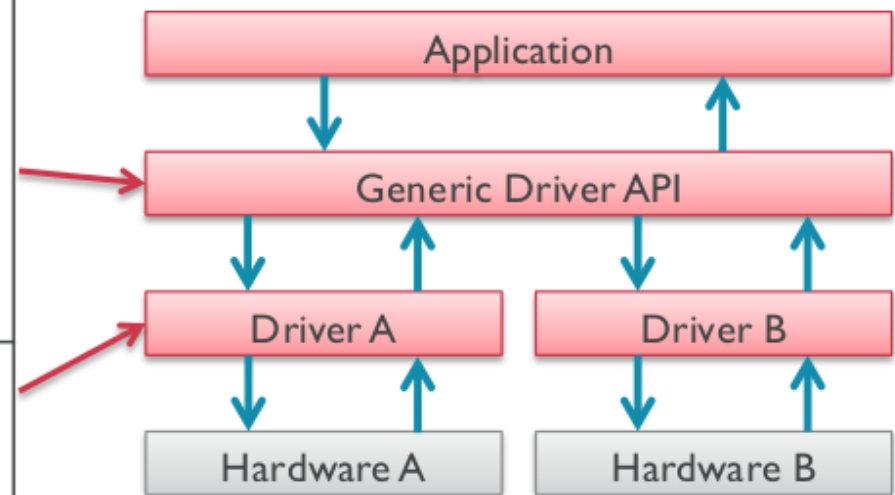
struct subsystem_api {
    subsystem_do_this_t do_this;
    subsystem_do_that_t do_that;
};

static inline int subsystem_do_this(struct device *device)
{
    struct subsystem_api *api;

    api = (struct subsystem_api *)device->driver_api;
    return api->do_this(device);
}

static int my_driver_do_this(struct device *device)
{
    ...
}

static struct subsystem_api my_driver_api_funcs = {
    .do_this = my_driver_do_this,
    .do_that = my_driver_do_that
};
```



Subsistemas do Zephyr

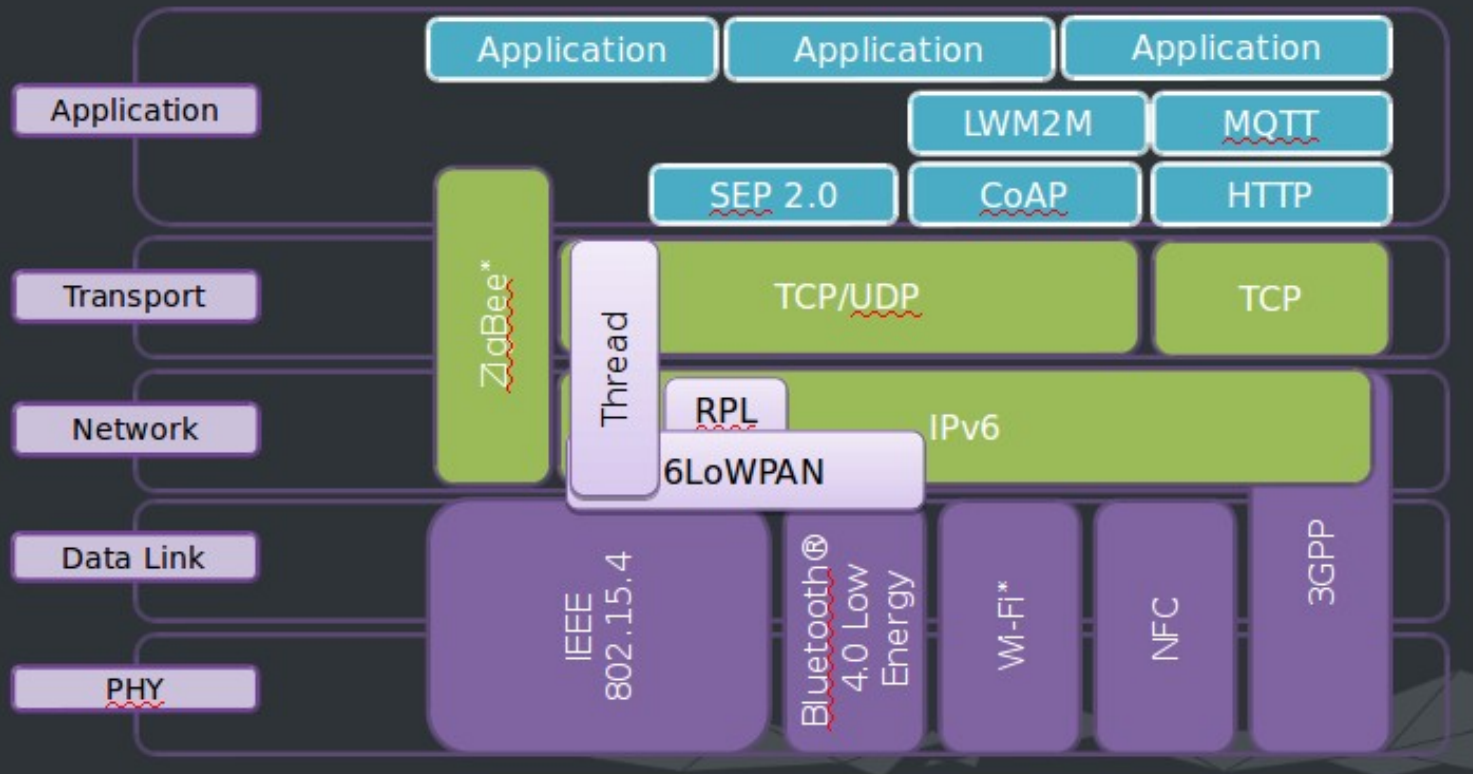


Zephyr subsys: Indo além do óbvio

- O Zephyr foi concebido para ser um RTOS voltado para dispositivos de borda;
- Ou seja componentes tipicamente utilizados em IoT figuram no chamado subsys do Zephyr;
- Além disso ferramentas úteis como console e shell podem ser habilitados para uso da aplicação.



Zephyr subsys: Indo além do óbvio



Mais informações?

- O Website do projeto possui toda documentação para tirar dúvidas: <http://docs.zephyrproject.org/>
- A lista de placas e arquiteturas suportadas pode ser checada aqui:
<http://docs.zephyrproject.org/boards/boards.html>



Obrigado!

