Boavizta - REX développement de Scaphandre pour Windows

Benoit Petit et Victorien Molle

bpetit@hubblo.org

biche@biche.re

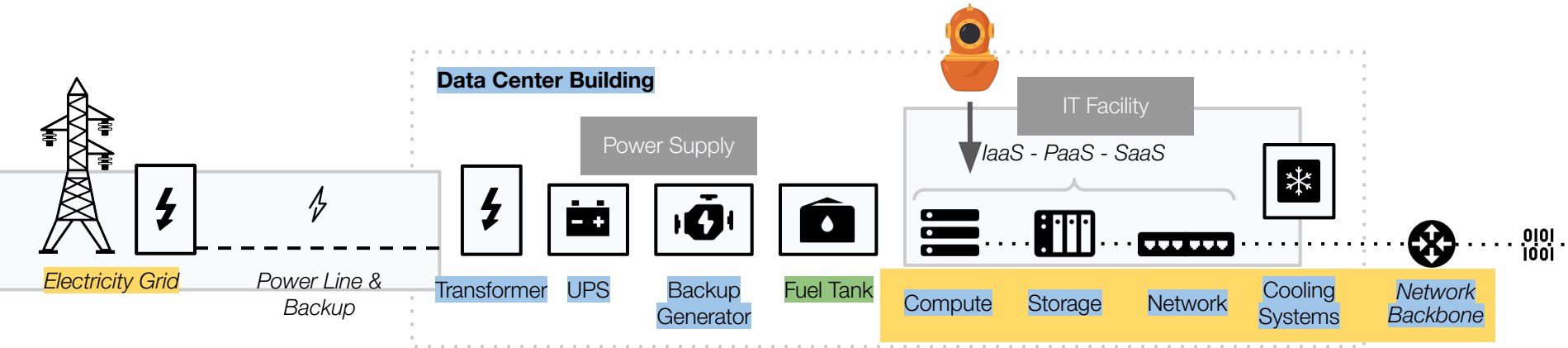
Scaphandre

Scope 1: Emissions directes : consommation d'énergies fossiles 🚗✈️

Scope 2: Emissions indirectes : consommation d'énergie finale ⚡

Scope 3: Autres émissions : fabrication, fournisseurs, ... 🏭

Scaphandre



Scope 1: Direct backup generator fuel consumption emissions

Scope 2: Electricity grid emission factors

Scope 3: Building and equipment embodied emissions from manufacturing

Scaphandre


Host power consumption
12 W
10 W
8 W
6 W
4 W
2 W
0 W
08:00
09:00
10:00
11:00
12:00
13:00
Process top consumers
awx-manage
2.9
docker-containe
0.56
dockerd
0.76
grafana-server
0.60
prometheus
1.3
qemu-system-x86
2.9
redis-server
0.50
Scaphandre Version
0.41
3h host energy consumpti...
40 Wh
35 Wh
30 Wh
25 Wh
20 Wh
1d host energy consumption
300 Wh
250 Wh
200 Wh
150 Wh
1w host energy consumption
2.2 kWh
2.0 kWh
1.8 kWh
1.6 kWh
1.4 kWh
1.2 kWh
1month host energy consumption over
4.5 kWh
4.0 kWh
3.5 kWh
3.0 kWh
2.5 kWh
1year host energy consumption over
55 kWh
50 kWh
45 kWh
40 kWh
35 kWh
30 kWh

Scaphandre

Data **visualization**

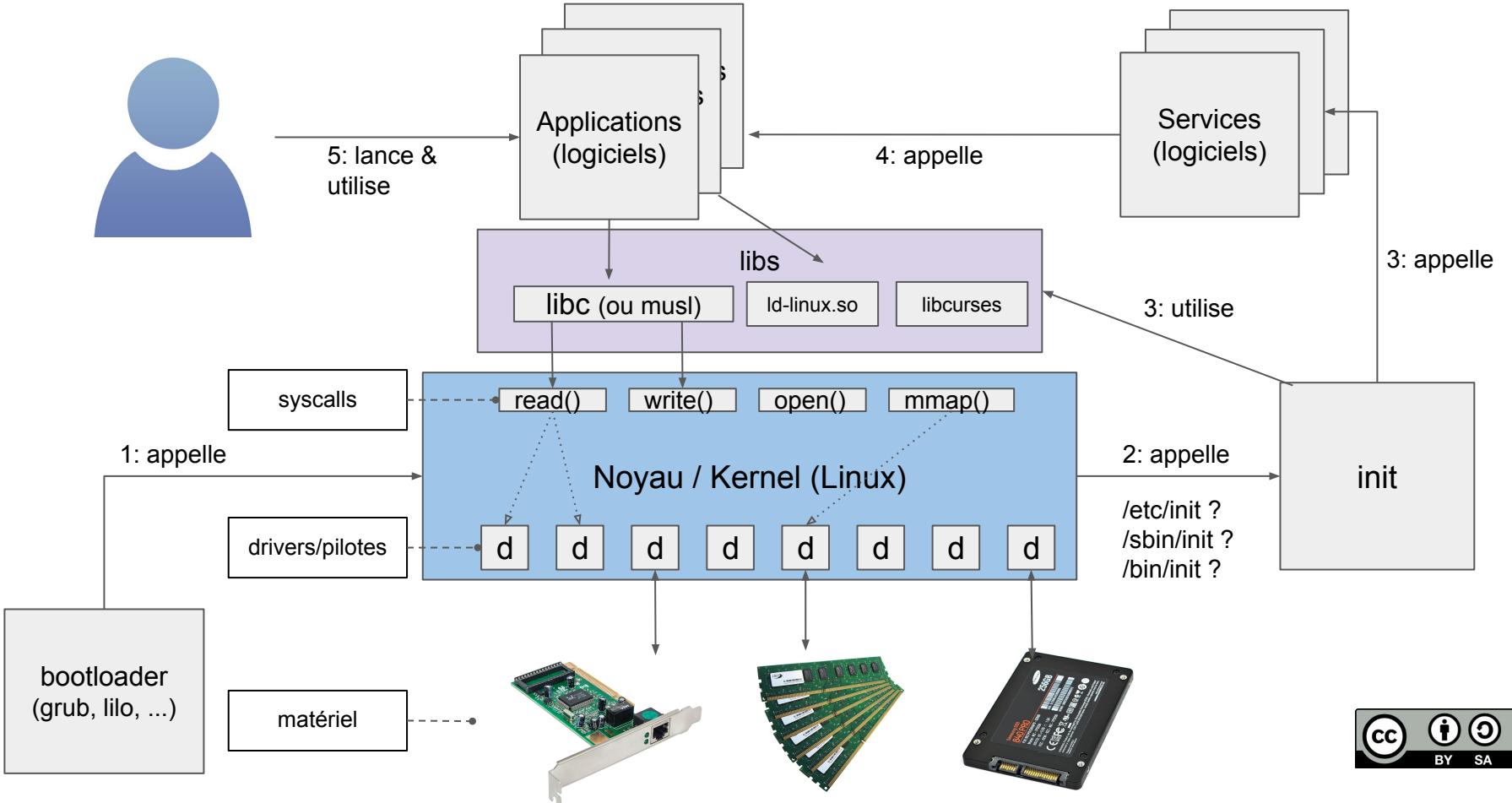
Data **aggregation** & access

Data **collection**

Scaphandre

Running
Average
Power
Limit
⚡

Qu’est ce qu’un système d’exploitation ?



RAPL

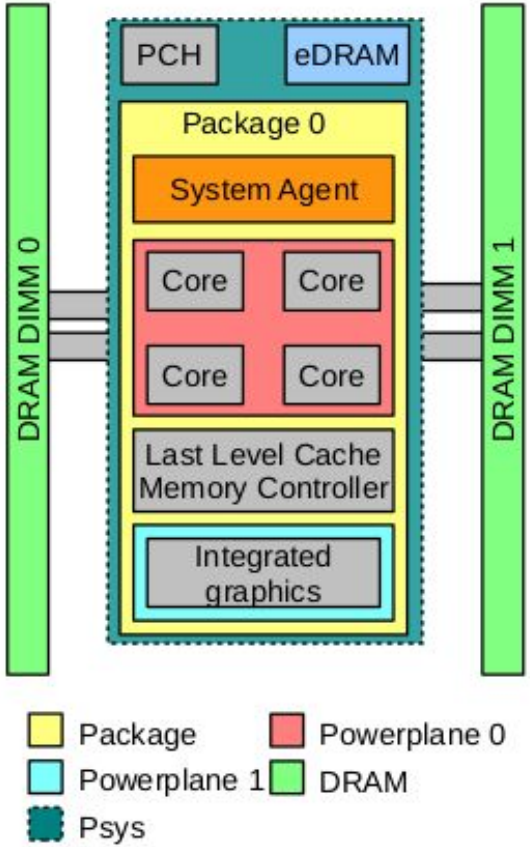


Fig. 1. Power domains supported by RAPL.

RAPL sous Linux

2 options

1. Par un module noyau ⇒ Power Caping Framework (“powercap”)
2. Par les MSR (Model Specific Register) ⇒ écrire un driver/pilote

Powercap

sudo cat /sys/class/powercap/intel-rapl:0:0/energy_uj
6120654252

```
tree /sys/class/powercap/
/sys/class/powercap/
├── intel-rapl -> ../../devices/virtual/powercap/intel-rapl
├── intel-rapl:0 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0
├── intel-rapl:0:0 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:0
├── intel-rapl:0:1 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:1
└── intel-rapl:0:2 -> ../../devices/virtual/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:2
```

Les MSR (intel x86)

```
15 const MSR_RAPL_POWER_UNIT: u16 = 0x606; //
16 const MSR_PKG_POWER_LIMIT: u16 = 0x610; // PKG RAPL Power Limit Control (R/W) See Section 14.7.3, Package RAPL Domain.
17 const MSR_PKG_ENERGY_STATUS: u16 = 0x611;
18 const MSR_PKG_POWER_INFO: u16 = 0x614;
19 const MSR_DRAM_ENERGY_STATUS: u16 = 0x619;
20 const MSR_PP0_ENERGY_STATUS: u16 = 0x639; //PP0 Energy Status (R/O) See Section 14.7.4, PP0/PP1 RAPL Domains.
21 const MSR_PP0_PERF_STATUS: u16 = 0x63b; // PP0 Performance Throttling Status (R/O) See Section 14.7.4, PP0/PP1 RAPL Domains.
22 const MSR_PP0_POLICY: u16 = 0x63a; //PP0 Balance Policy (R/W) See Section 14.7.4, PP0/PP1 RAPL Domains.
23 const MSR_PP0_POWER_LIMIT: u16 = 0x638; // PP0 RAPL Power Limit Control (R/W) See Section 14.7.4, PP0/PP1 RAPL Domains.
24 const MSR_PP1_ENERGY_STATUS: u16 = 0x641; // PP1 Energy Status (R/O) See Section 14.7.4, PP0/PP1 RAPL Domains.
25 const MSR_PP1_POLICY: u16 = 0x642; // PP1 Balance Policy (R/W) See Section 14.7.4, PP0/PP1 RAPL Domains.
26 const MSR_PP1_POWER_LIMIT: u16 = 0x640; // PP1 RAPL Power Limit Control (R/W) See Section 14.7.4, PP0/PP1 RAPL Domains.
```

Des registres dédiés à un usage spécifique.

Certains sont accessibles en écritures, d’autres seulement en lecture.

Pour AMD x86, fonctionnement similaire, mais les adresses sont différentes.

RAPL sous Windows

1 option

1. Par les MSR (Model Specific Register) ⇒ écrire un driver/pilote

Analyse de l’existant

Un travail de recherche effectué sur la base de deux logiciels :

- Intel Power Gadgets
- CPU-Z

Les deux logiciels révèlent des points communs :

- Emploient un driver (fichier .sys)
- Chargent le driver via un service Windows

Etape 1 : Création du service

La création de services sous Windows s'effectue de la façon suivante :

1. Ouverture du Service Control Manager
2. Création du service avec différents paramètres
   a. Un nom
   b. Un type de service (driver, programme utilisateur)
   c. Une condition de démarrage (au démarrage du système, à l'ouverture d'une session)
   d. Un chemin vers l'exécutable du service
3. Démarrage du service

Voir les API "*OpenSCManager*", "*CreateService*" et "*StartService*" dans la documentation Microsoft.

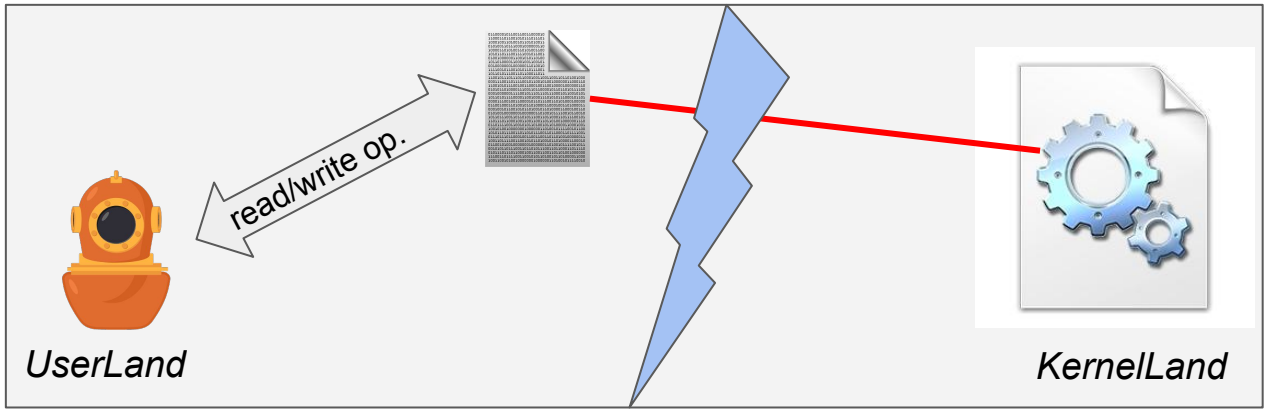
A cet instant, le driver peut être chargé dans le système et être utilisé.

Structure du driver - #1

Le driver consiste à récupérer des informations stockées dans les MSRs.

Ces informations sont demandées par l'espace utilisateur (Scaphandre) et sont renvoyées par l'espace noyau (ScaphandreDriver)

Pour cela, il faut pouvoir établir un canal de communication entre l'espace utilisateur et le noyau.



Structure du driver - #2

La communication s’établit via un fichier partagé par le driver et exposé à l’espace utilisateur.

Ce fichier représente une mémoire partagée dans lequel Scaphandre lit et écrit des données.

Néanmoins, écrire dans cette mémoire ne suffit pas à “réveiller” le driver, il faut lui signaler notre souhait d’échanger avec lui.

Deux versions ont été mises au point :

1. Un échange à sens unique : non retenue
2. Un échange bi-directionnel : retenue

Discuter avec un driver

Afin d'interagir avec un driver, il faut appeler une fonction Windows nommée *DeviceIoControl*

Cette fonction permet de passer des informations depuis l'espace utilisateur à un driver dans l'espace noyau.

Elle a notamment pour paramètres :

- Un Input/Output Control Code (IOCTL_CODE)
- Des données à envoyer (non obligatoire)
- Un espace alloué pour des données à recevoir (non obligatoire)

ScaphandreDrv - Version #1

La première version consistait à envoyer un IOCTL_CODE et attendre des données en retour.

L'IOCTL_CODE contenait l'index du registre spécifique au CPU, ainsi le driver pouvait aller lire dans le MSR souhaité et renvoyait le résultat à l'utilisateur.

Cette méthode était totalement fonctionnelle mais comportait les limitations suivantes :

- La valeur d'un IOCTL_CODE est restreinte par la liberté qu'a un développeur (Microsoft se réserve les 2048 premières valeurs, il en reste néanmoins 2047)
- Les index des registres spécifiques à RAPL étant dans la tranche des valeurs restreintes, une ré-association était nécessaire

```
#define AGENT_POWER_UNIT_CODE 0xBEB
case 0xBEB:
        msrResult = __readmsr(MSR_RAPL_POWER_UNIT); //  MSR_RAPL_POWER_UNIT = 0x606
        break;
```

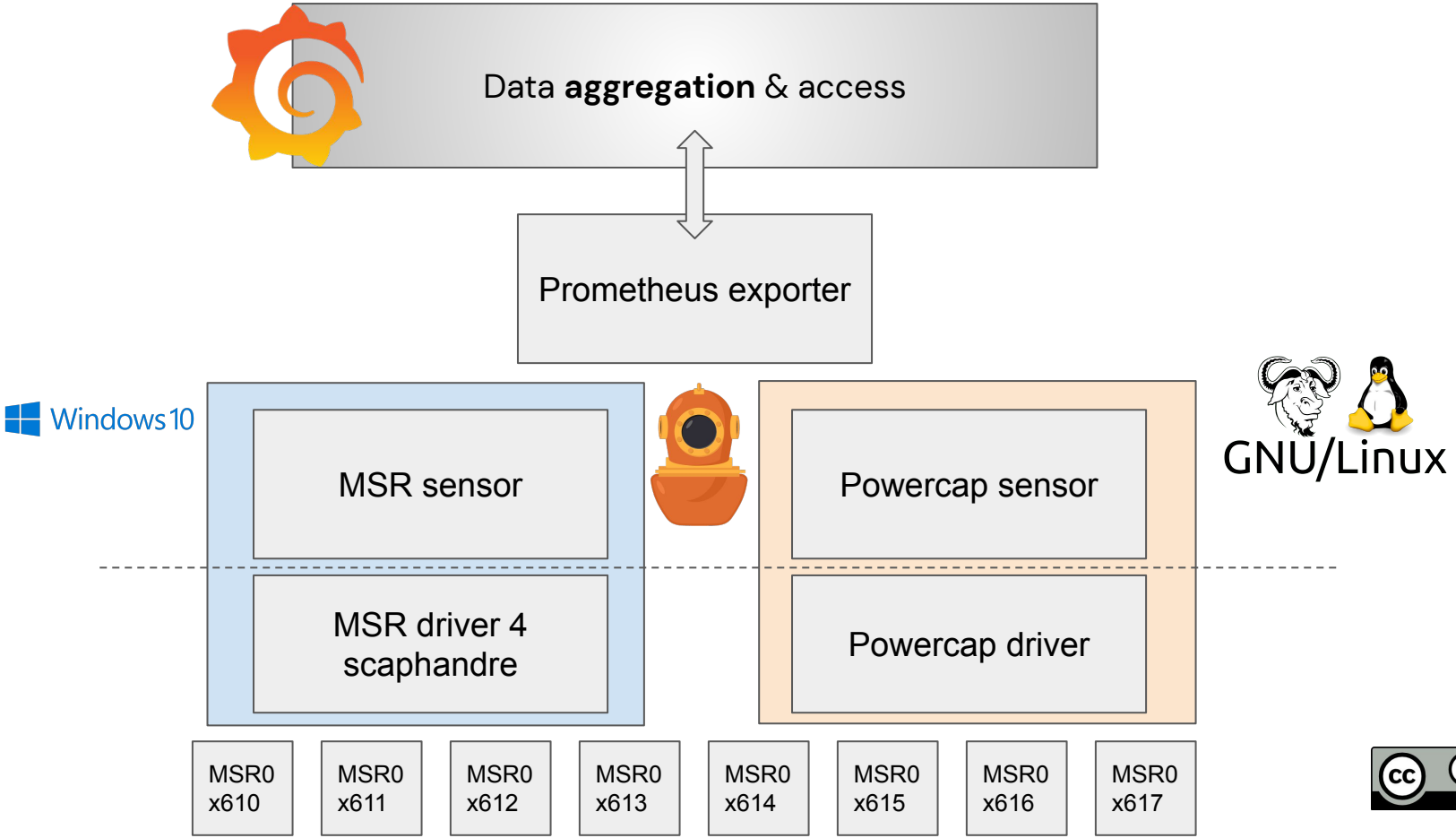
ScaphandreDrv - Version #2

La seconde version consiste à envoyer au driver l'index du registre spécifique à aller lire pour que le driver nous retourne la valeur de ce registre.

Les avantages sont :

- Toujours utiliser le même IOCTL_Code → plus de limitation
- Permettre à l'utilisateur de requêter n'importe quel index → plus besoin de faire de correspondance côté driver
- D'avoir une gestion optimale de la mémoire d'échange → l'échange des données se fait dans le deux sens

Scaphandre sous Windows



Next steps (en se basant sur ce que l’on a appris)

1. MacOS
2. Sensor basé sur les MSR pour Linux
   (lorsqu’il n’est pas possible d’utiliser powercap)

bpetit@hubblo.org

biche@biche.re