Interface for Heterogeneous Kernels (IHK) Specifications
Version 1.7.1-0.7

Masamichi Takagi, Balazs Gerofi, Tomoki Shirasawa, Gou Nakamura
and Yutaka Ishikawa

Monday 18th January, 2021

Contents

List of Figures

Chapter 1

IHK外部仕様

1.1 概要

Interface for Heterogeneous Kernels (IHK) is a low-level software infrastructure, which enables partitioning node resources and the management of lightweight kernels on subsets of the resources. This section introduces the basic architecture of IHK and gives a brief overview of its main components. An overview of the IHK architecture is shown in Figure 1.1.

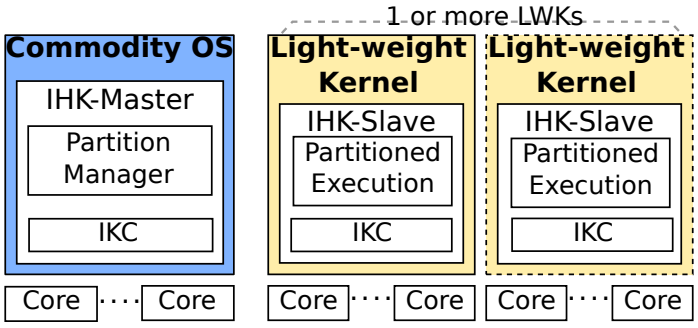

Figure 1.1: **Architectural overview of IHK components.**

IHK categorizes kernels in two types: a master kernel and the slave kernels (i.e., lightweight kernels). The master kernel is a kernel that is booted in the node first through the normal booting process, for example, booted from BIOS or UEFI, and is typically a commodity operating system, it is Linux in the rest of this document. Slave kernels are kernels that are booted from the master kernel. IHK's components in the master and slave kernels are called IHK-master and IHK-slave[1], respectively.

Resource partitioning, the management and bootstrapping of lightweight slave kernels are implemented in IHK-master, while support for executing over a partition of resources is implemented in IHK-slave. A low-level communication facility called IHK-IKC is present both in IHK-master and IHK-Slave.

[1]The terms "IHK-slave" and "co-kernel" are used interchangably.

1.1.1 管理者向け機能

This section discusses the functionalities and components of IHK-master. The resource partitioning mechanism provided by the implementation in the Linux kernel is also explained.

IHK-master consists of two types of modules. *IHK-master core* provides the basic IHK framework and management infrastructure. It is required for registering/removing the so called *IHK-master drivers* (discussed below) and provides administration interface through device files and ioctl() APIs for:

- Managing devices.
- Managing OS kernel instances.

In particular, the IHK-master core module enables in-kernel interfaces (by means of exporting a set of IHK specific Linux kernel functions) which allow registration and deregistration of IHK-master drivers.

IHK-master drivers represent resources, such as CPU cores of an SMP chip along with the physical memory of the given node or PCI-Express attached co-processors. Specifically, the current IHK implementation in Linux provides one type of IHK-master drivers:

- *IHK-SMP x86:* Represents a virtual device that enables partitioning CPU cores of an x86 (Xeon) SMP chip as well as the physical memory attached to the node among OS instances.

Note, that neither the IHK-master core module, nor the IHK-SMP x86 drivers require any modifications to the Linux kernel.

IHK-master drivers support the abstraction of *IHK devices*, which essentially represent resources. On top of IHK devices one can create *IHK OS instances* and use the framework to assign a set of the underlying resources to the particular OS instance. As we mentioned earlier, IHK exposes its management interface via device files which in turn can be controlled with specific command line tools. Figure 1.2 shows the execution steps of an IHK device registration and the creation of an OS instance for x86 (Xeon) SMP chip.

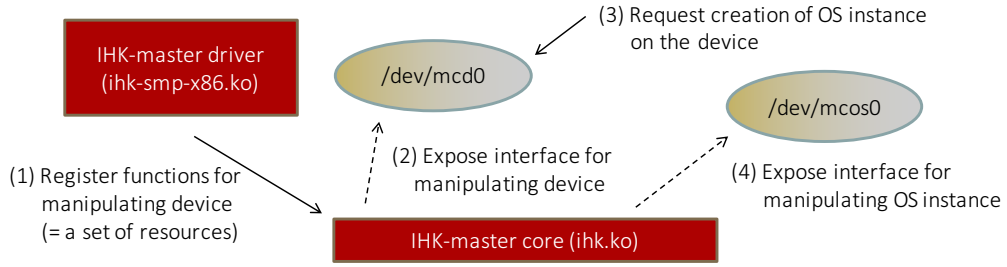

Figure 1.2: **Steps of IHK-master driver registration and device file creation.**

The initial state of the figure is right after the IHK-master core module (`ihk.ko`) has been loaded. The following four steps are then highlighted:

1. Load an IHK-master driver module (`ihk-smp-x86.ko`), which automatically registers itself into the IHK framework.
2. The IHK framework creates the `/dev/mcd0` device file, which will represent the resources accessible through the inserted IHK master driver.

3. Use the IHK tools (e.g. `ihkconfig` command or `ihk_config` functions) to request creation of an OS instance, which in turn does an ioctl() call on the specified IHK device.

4. The IHK framework creates `/dev/mcos0` device file, which represents an OS instance on top of IHK device `/dev/mcd0`.

Note the index `0` in the file names of `/dev/mcd0` and `/dev/mcos0`. The IHK framework allows registration of multiple IHK-master drivers as well as the creation of multiple OS instances over a specific IHK device. The index of the corresponding device file is assigned by the framework automatically.

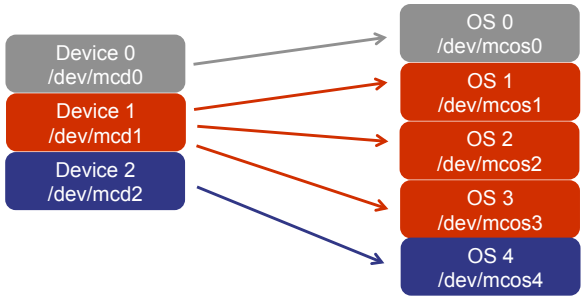

Figure 1.3: **Relation between IHK devices and OS instances.**

To emphasize the relation between OS instances and IHK devices see Figure 1.3. As shown, `/dev/mcd1` has multiple OS instances on top of it.

`x86_64` アーキテクチャのシステムでの資源管理と OS 管理のステップは以下の通り。

1. コアドライバ `ihk.ko` と、`ihk.ko` の開示するインターフェイス経由でシステム依存機能を提供するドライバ `ihk_smp_x86.ko` を `insmod` する。

2. IHK が CPU 資源およびメモリ資源を Linux から獲得する。この操作を資源の予約と呼ぶ。

3. IHK が OS インスタンスを作成する。

4. IHK が OS インスタンスに CPU 資源を割り当てる。

5. IHK が OS インスタンスにメモリ資源を割り当てる。

6. IHK が OS インスタンスにカーネルイメージをロードし、起動する。

7. IHK が OS 状態監視のためのファイルディスクリプタを OS インスタンスから取得する。

8. IHK が McKernel にプロセスを起動する。

9. IHK が必要に応じて、OS インスタンスの統計情報の取得、OS インスタンスの一時停止、OS インスタンスのメモリダンプの採取を行う。

10. IHK が OS インスタンスのカーネルをシャットダウンする。

11. IHK が OS インスタンスに割り当てられた CPU 資源およびメモリ資源を IHK に戻す。この操作を資源の解放と呼ぶ。

12. IHK が OS インスタンスを破棄する。

13. IHK が CPU 資源およびメモリ資源を Linux に戻す。この操作を資源の解放と呼ぶ。

IHK は運用ソフトウェアがこれらの操作を行えるようにするコマンド群およびライブラリを提供する。

カーネルモジュール、コマンド、ライブラリの場所は以下の通り。SMP プロセッサ向け、x86_64 アーキ向けのファイルを記載する。なお、IHK のインストールディレクトリを<ihk_install>とする。

ファイル	説明
<ihk_install>/kmod/ihk.ko	IHK-master core
<ihk_install>/kmod/ihk-smp-x86.ko	IHK-master driver
<ihk_install>/include/libihk.h	資源管理関数および OS 管理関数のヘッダファイル
<ihk_install>/lib/libihk.so	資源管理関数および OS 管理関数の共有オブジェクト
<ihk_install>/sbin/ihkconfig	資源管理コマンド
<ihk_install>/sbin/ihkosctl	OS 管理コマンド
<ihk_install>/sbin/ihkmond	カーネルメッセージの syslog プロトコルによる/dev/log への転送と、ハングアップの監視とを行うデーモン

コマンドおよびライブラリのソースコードの場所は以下の通り。なお、IHK のソースディレクトリを<ihk_src>とする。

ファイル	説明
<ihk_src>/linux/user/ihklib.h.in	ヘッダファイル
<ihk_src>/linux/user/ihklib.c	資源管理関数および OS 管理関数の実装
<ihk_src>/linux/user/ihkconfig.c	資源管理コマンドの実装
<ihk_src>/linux/user/ihkosctl.c	OS 管理コマンドの実装

1.1.2 LWK 向け機能

IHK は LWK に以下の機能を提供する。

- IHK により割り当てられた資源情報の取得
- カーネル引数の IHK からの取得
- カーネルメッセージバッファアドレスの IHK への通知
- Linux との通信（Inter Kernel Communication, IKC）機能

上記ライブラリのソースコードの場所は以下の通り。SMP プロセッサ向け、x86_64 アーキ向けのファイルを記載する。なお、IHK のソースディレクトリを<ihk_src>、LWK のソースディレクトリを<lwk_src>とする。

ファイル	説明
<ihk_src>/cokernel/smp/x86/	LWK 向け機能関数定義（LWK 非依存・メニーコア構成依存・アーキ依存部）
<ihk_src>/cokernel/smp/x86/include	LWK 向け機能ヘッダファイル（LWK 非依存・メニーコア構成依存・アーキ依存部）
<ihk_src>/ikc/include/ikc/ihk.h	ヘッダファイル（IKC 関連、LWK 非依存・アーキ非依存部）
<ihk_src>/linux/include/ihk/	ヘッダファイル（Linux ドライバ向けインターフェイス、LWK 非依存・アーキ非依存部）
<lwk_src>/lib/include/ihk/	ヘッダファイル（LWK 依存・アーキ非依存部）
<lwk_src>/arch/x86/kernel/include/ihk/	ヘッダファイル（LWK 依存・アーキ依存部）

1.1.3 Linux ドライバ向け機能

IHK は Linux に以下の機能を提供する。

- 制御レジスタへのアクセス

上記ライブラリのファイル構成は以下の通り。なお、IHK のソースディレクトリを<ihk_src>とする。

ファイル	説明
<ihk_src>/linux/include/ihk/ihk_host_driver.h	ヘッダファイル

1.2 関数仕様

以下の関数や第 1.3 節で説明するコマンドを並列で呼び出す場合は、呼び出し元で排他制御を行うなどして IHK や LWK の一貫性を担保する必要がある。

1.2.1 管理者向け資源管理機能

1.2.1.1 CPU 予約

書式

```
int ihk_reserve_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された IHK デバイスに対して、cpus，num_cpus で指定された CPU を予約する。cpus には Linux での CPU 番号の配列のアドレスを指定し、num_cpus には配列のサイズを指定する。呼び出し元が cpus の領域を用意する。

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EFAULT	cpus にアクセスできない
-EINVAL	不正なパラメータ

1.2.1.2 予約済 CPU 数取得

書式

```
int ihk_get_num_reserved_cpus(int index)
```

説明

index で指定された IHK デバイスに予約されている CPU の数を返す。

戻り値

0 以上	CPU 数
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.3 予約済 CPU 情報取得

書式

```
int ihk_query_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された IHK デバイスに予約されている CPU の番号列を cpus で指定された配列に格納する。 num_cpus には配列のサイズを指定する。呼び出し元が cpus の領域を用意する。

利用方法は以下の通り。

1. ihk_get_num_reserved_cpus() を用いて CPU 数を取得する。
2. 取得した CPU 数のサイズを持った整数配列の領域を呼び出し元に確保する。
3. ihk_query_cpu() に配列のアドレスとサイズを渡し、CPU の番号列を取得する。

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.4 CPU 解放

書式

```
int ihk_release_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された IHK デバイスに予約されている CPU のうち、cpus，num_cpus で指定されたものを解放する。cpus には Linux での CPU 番号の配列を指定し、num_cpus には配列のサイズを指定する。呼び出し元が cpus の領域を用意する。

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.5 メモリ領域予約動作制御

書式

```
int ihk_reserve_mem_conf(int index, int key, void *value)
```

説明

index で指定された IHK デバイスに対する ihk_reserve_mem() の動作を key と value のペアで指定したものに変更する。value は値へのポインタで指定する。key と value のペアの意味は以下のように定義される。

IHK_RESERVE_MEM_BALANCED_{ENABLE,BEST_EFFORT,VARIANCE_LIMIT}

IHK_RESERVE_MEM_BALANCED_ENABLE（型は int、デフォルトは 0）が非ゼロの場合は、NUMA ノードごとの予約サイズが NUMA ノード間でなるべく均等になるように予約する。目的は、NUMA ノードごとのメモリ空き容量に NUMA ノード間でばらつきがあり、またそれらの空き容量が事前にわからないようなシステムで、合計予約サイズをより大きくすることである。ステップは以下の通り。

1. IHK_RESERVE_MEM_BALANCED_BEST_EFFORT（型は int、デフォルトは 0）が 0 の場合は、ihk_reserve_mem() で指定したサイズの NUMA ノードに渡る合計値（以下、ihk_reserve_mem() 指定合計サイズと呼ぶ）を予約サイズとする。予約時点の空きメモリ量の NUMA ノードに渡る合計に IHK_RESERVE_MEM_MAX_SIZE_RATIO_ALL を乗じたもの（以下、調整後空き容量と呼ぶ）がこのサイズ未満の場合は、-ENOMEM を返す。非ゼロの場合は、調整後空き容量と、ihk_reserve_mem() 指定合計サイズのうち小さい方の値を予約サイズとする。

2. NUMA ノードに渡る合計サイズがこの予約サイズになるように、また NUMA ノードごとの予約サイズが NUMA ノード間でなるべく均等になるように各 NUMA ノードの予約サイズを決定する。この予約サイズの NUMA ノードに渡る平均からの差の絶対値の平均に対する割合が、IHK_RESERVE_MEM_BALANCED_VARIANCE_LIMIT で指定した値（型は int、単位は%）を超えた場合は-ENOMEM を返す。なお、各 NUMA ノードの予約サイズは IHK_RESERVE_MEM_BALANCED_VARIANCE_LIMIT で指定した値に影響を受けない。

IHK_RESERVE_MEM_MIN_CHUNK_SIZE

予約処理は、あるサイズの物理連続領域（ページ単位）を繰り返し Linux に要求する、ということを、大きいサイズから始めてサイズを小さくしながら繰り返す。このサイズの下限値を指定された値にする。デフォルト設定はページサイズである。

このパラメタの目的は、Linux による空き領域の分断化が激しい状況においてメモリ予約処理時間を抑えることである。上記の状況で予約処理時間が長くなるのは、小さいサイズでの物理連続領域が大量に存在するので、小さいサイズでの要求回数が非常に大きくなるためである。

IHK_RESERVE_MEM_MAX_SIZE_RATIO_ALL

ihk_reserve_mem() でサイズに-1 を指定した場合と IHK_RESERVE_MEM_BALANCED_ENABLE に非ゼロを指定した場合に用いられる予約サイズを、予約時点で測定した空き容量に指定した値を乗じたものにする。なお、ゼロ以下の値または 98 より大きい値を設定しようとすると-EINVAL を返す。また、デフォルト設定は 98%である。

目的は、Linux による空き領域の分断化が激しい状況においてメモリ予約処理時間を抑えること、また予約時に Linux のプロセスのメモリ要求が満たされない状況にならないようにすることである。

IHK_RESERVE_MEM_TIMEOUT

予約処理は、あるサイズの物理連続領域を繰り返し Linux に要求する、ということを、大きいサイズから始めてサイズを小さくしながら繰り返す。あるサイズでの Linux への繰り返し要求の処理時間が指定時間（単位は秒）を超えた場合に予約を打ち切る。デフォルト設定は 30 秒である。

このパラメタの目的は、IHK_RESERVE_MEM_MAX_SIZE_RATIO_ALL と同じく、Linux による空き領域の分断化が激しい状況においてメモリ予約処理時間を抑えることである。

戻り値

0	正常終了
-EINVAL	不正な key 値

1.2.1.6 メモリ領域予約

書式

```
int ihk_reserve_mem(int index, struct ihk_mem_chunk *mem_chunks,
  int num_mem_chunks)
```

説明

index で指定された IHK デバイスに対して、mem_chunks, num_mem_chunks で指定されたメモリ領域を予約する。mem_chunks にはメモリ領域情報の配列を指定し、num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。要求サイズは 4 MiB の整数倍である必要がある。また、予約サイズは NUMA ノードごと最大 4 MiB 上振

れする可能性がある。なお、NUMA ノード 0 については Linux にメモリを残すために空き容量の 95%以上の予約を試みない。

ihk_mem_chunk は以下のように定義される。

```
typedef struct {
    unsigned long size;   // 要求サイズを指定する。-1 が指定された場合、
                          // 可能な限り多くのメモリを予約する。
    int numa_node_number; // NUMA ノード番号を指定する。
} ihk_mem_chunk;
```

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	チャンク数が不正、または NUMA ノード番号が不正、またはサイズが 4MiB の整数倍でない
-ENOMEM	メモリ不足

1.2.1.7 予約済メモリ領域数取得

書式

```
int ihk_get_num_reserved_mem_chunks(int index)
```

説明

index で指定された IHK デバイスに予約されているメモリ領域の数を返す。

戻り値

0 以上	メモリ領域数
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.8 予約済メモリ領域情報取得

書式

```
int ihk_query_mem(int index, struct ihk_mem_chunk *mem_chunks,
 int num_mem_chunks)
```

説明

index で指定された IHK デバイスに予約されているメモリ領域の情報を mem_chunks で指定された配列に格納する。 num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.9 メモリ領域解放

書式

```
int ihk_release_mem(int index, struct ihk_mem_chunk *mem_chunks,
 int num_mem_chunks)
```

説明

index で指定された IHK デバイスに予約されているメモリ領域のうち、mem_chunks, num_mem_chunks で指定されたものを解放する。mem_chunks にはメモリ領域情報の配列を指定し、num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。一連のチャンクの解放の途中で失敗した場合、それまでに解放したチャンクは解放されたままとなる。mem_chunks の要素の size フィールドに-1 を指定した場合、全ての NUMA ノードについて予約されたメモリ領域の全てを解放する。

戻り値

0	正常終了
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.10 OS インスタンス作成

書式

```
int ihk_create_os(int index)
```

説明

index で指定された IHK デバイス上に IHK での OS 表現の実体である OS インスタンスを作成し、そのインデックスを返す。

戻り値

0 以上	生成された OS インスタンスのインデックス
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.11 OS インスタンス数取得

書式

```
int ihk_get_num_os_instances(int index)
```

説明

index で指定された IHK デバイスの OS インスタンス数を返す。

戻り値

0 以上	OS インスタンス数
-ENOENT	指定した IHK デバイスが存在しない
-EINVAL	不正なパラメータ

1.2.1.12 OS インスタンス一覧取得

書式

```
int ihk_get_os_instances(int index, int *indices, int num_os_instances)
```

説明

index で指定された IHK デバイスの OS インスタンスのインデックス列を indices で指定された配列に格納する。num_os_instances には OS インスタンス数を指定する。呼び出し元が indices を用意する。

なお、ポスト京では OS インスタンスは 1 つのみ生成するため、本関数で OS インスタンスの存在を確認した後は、OS インデックス 0 を固定的に使用してよい。

戻り値

0	正常終了
-EINVAL	num_os_instances に指定した値が実際の OS インスタンス数と一致しない

1.2.1.13 OS インスタンス削除

書式

```
int ihk_destroy_os(int dev_index, int os_index)
```

説明

dev_index で指定された IHK デバイスの os_index で指定された OS インスタンスを削除する。当該 OS インスタンスに割り当てられた資源は解放される。なお、この関数は OS インスタンスが異常状態にあってもブロックしたりエラーを返したりすることはない。

本関数は OS インスタンスの状態を変更するので、/dev/mcos<os_index>を排他的にオープンする必要がある。ihkmond（第 1.3.2.18 節参照）のような、当該デバイスファイルを定期的かつ短時間オープンするプロセスとの衝突を防ぐため、本関数は内部でオープンのリトライを行う。

戻り値

0	正常に終了した。
-ENOENT	指定された IHK デバイスまたは OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	/dev/mcos<os_index>をオープンしているプロセスが存在する。なお、オープンする可能性があるのは mcexec、ihkmond、IHK の関数、IHK のコマンドである。

1.2.2 管理者向け OS 管理機能

1.2.2.1 CPU 割当

書式

```
int ihk_os_assign_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された OS インスタンスに対し、IHK デバイスに予約されている CPU のうち cpus，num_cpus で指定されたものを割り当てる。cpus には Linux での CPU の番号配列のアドレスを指定し、num_cpus には配列のサイズを指定する。呼び出し元が cpus の領域を用意する。なお、LWK によっては cpus での CPU 番号の順番が意味を持つ。例えば、McKernel では cpus で指定した順番に CPU 番号が振り直される。この呼び出しは特権ユーザのみ実行できる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	OS インスタンスがブート済みである

1.2.2.2 割当済 CPU 数取得

書式

```
int ihk_os_get_num_assigned_cpus(int index)
```

説明

index で指定された OS インスタンスに割り当てられている CPU の数を返す。

戻り値

0 以上	CPU 数
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.3 割当済 CPU 情報取得

書式

```
int ihk_os_query_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された OS インスタンスに割り当てられている CPU の番号列を cpus で指定された配列に格納する。 num_cpus には配列のサイズを指定する。

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.4 CPU 解放

書式

```
int ihk_os_release_cpu(int index, int *cpus, int num_cpus)
```

説明

index で指定された OS インスタンスに割り当てられている CPU のうち cpus，num_cpus で指定されたものを解放する。cpus には Linux での CPU 番号の配列を指定し、num_cpus には配列のサイズを指定する。呼び出し元が cpus の領域を用意する。この呼び出しは特権ユーザのみ実行できる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.5 IKC map 設定

書式

```
int ihk_os_set_ikc_map(int index, struct ihk_ikc_cpu_map *map, int num_cpus)
```

説明

index で指定された OS インスタンスの IKC map を map に設定する。num_cpus には OS インスタンスに割り当てられた CPU 数を指定する。呼び出し元が map の領域を用意する。なお、この呼び出しは特権ユーザのみ実行できる。

IKC map とは LWK CPU とその IKC メッセージ送信先 CPU の対応関係のことである。IKC map は struct ihk_ikc_cpu_map の配列で表現する。struct ihk_ikc_cpu_map は以下のように定義される。

```
struct ihk_ikc_cpu_map {
   int src_cpu; /* LWK CPU */;
   int dst_cpu; /* IKC メッセージ送信先 CPU */
};
```

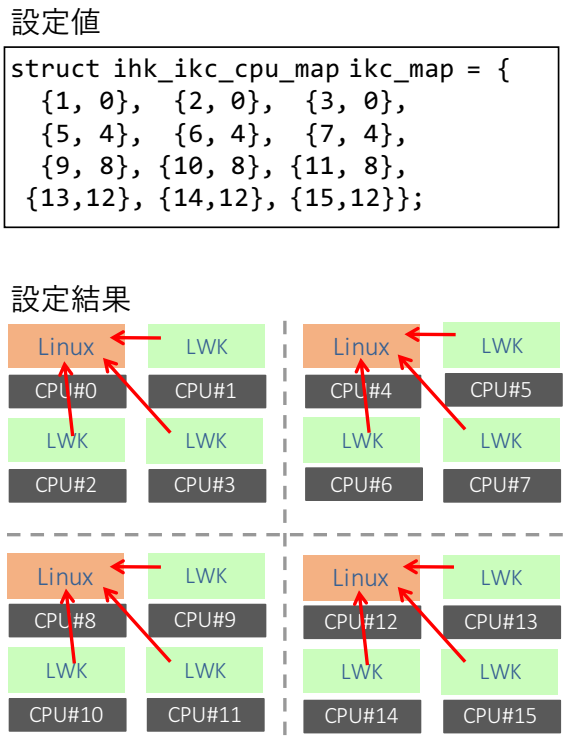

Figure 1.4: ihk_os_set_ikc_map() の例

IKC map の目的は、IKC 通信の送信先 CPU を複数にした上で、送信の際に物理的に近い CPU に送るようにすることにより、IKC 通信の遅延を削減することである。ihk_os_set_ikc_map() の例を図 1.4 に示す。この例では、プロセッサを 4 つの区画に分け、IKC 通信がそれぞれの区画内で閉じるように設定している。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	OS インスタンスがブート済みである

1.2.2.6 IKC map 取得

書式

```
int ihk_os_get_ikc_map(int index, struct ihk_ikc_cpu_map *map, int num_cpus)
```

説明

index で指定された OS インスタンスの IKC map を map で指定された配列に格納する。num_cpus には配列のサイズを指定する。呼び出し元が map の領域を用意する。なお、この呼び出しは特権ユーザのみ実行できる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.7 メモリ割当

書式

```
int ihk_os_assign_mem(int index, struct ihk_mem_chunk *mem_chunks,
 int num_mem_chunks)
```

説明

index で指定された OS インスタンスに対し、IHK デバイスに予約されているメモリ領域のうち mem_chunks，num_mem_chunks で指定されたものを割り当てる。mem_chunks にはメモリ領域情報の配列を指定し、num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。この呼び出しは特権ユーザのみ実行できる。mem_chunks の要素の size フィールドに-1 を指定した場合、指定された NUMA ノードについて予約されたメモリ領域の全てを割り当てる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	OS インスタンスがブート済みである

1.2.2.8 割当済メモリ領域数取得

書式

```
int ihk_os_get_num_assigned_mem_chunks(int index)
```

説明

index で指定された OS インスタンスに割り当てられているメモリ領域の数を返す。

戻り値

0 以上	メモリ領域数
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.9 割当済メモリ領域情報取得

書式

```
int ihk_os_query_mem(int index, struct ihk_mem_chunks *mem_chunks,
  int num_mem_chunks)
```

説明

index で指定された OS インスタンスに割り当てられているメモリ領域の情報を mem_chunks に格納する。 num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.10 メモリ領域解放

書式

```
int ihk_os_release_mem(int index, struct ihk_mem_chunks *mem_chunks,
  int num_mem_chunks)
```

説明

index で指定された OS インスタンスに割り当てられているメモリ領域のうち mem_chunks, num_mem_chunks で指定されたものを解放する。mem_chunks にはメモリ領域情報の配列を指定し、num_mem_chunks には配列のサイズを指定する。呼び出し元が mem_chunks の領域を用意する。一連のチャンクの解放の途中で失敗した場合、それまでに解放したチャンクは解放されたままとなる。この呼び出しは特権ユーザのみ実行できる。mem_chunks の要素の size フィールドに-1 を指定した場合、全ての NUMA ノードについて割り当てられたメモリ領域の全てを解放する。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	OS インスタンスがブート済みである

1.2.2.11 監視用 eventfd 取得

書式

```
int ihk_os_get_eventfd(int index, int type)
```

説明

index で指定された OS インスタンスでの type で指定したイベント発生を通知する eventfd を取得する。この呼び出しは特権ユーザのみ実行できる。type の取りうる値は以下の通り。

type の値	説明
0	カーネルおよびユーザのメモリ使用量が（IHK によって LWK に割り当てられた量− 2MiB）を超えた際に通知する。
2	OS がハングアップした際または PANIC を起こした際に通知する。

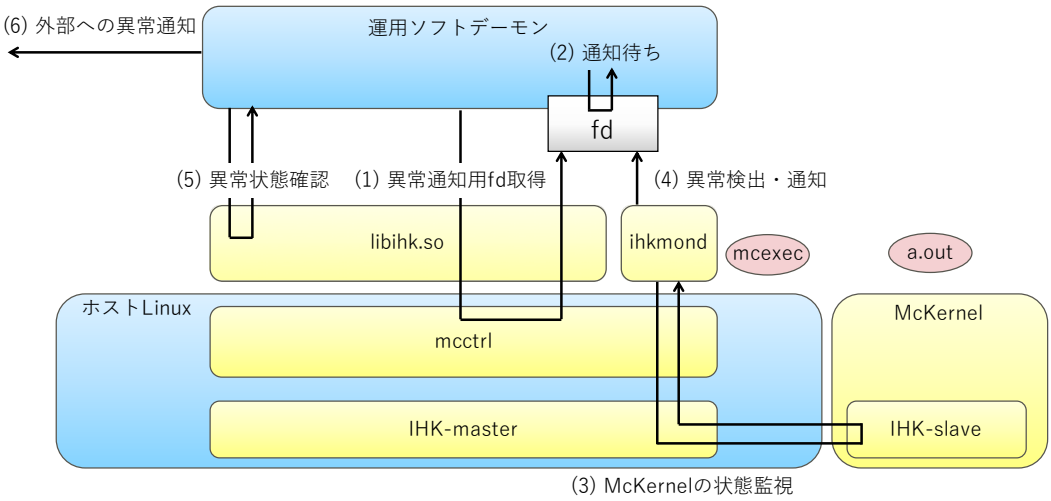

Figure 1.5: OS 状態監視フロー

図 1.5 を用いて LWK として McKernel が動作している場合の OS 状態監視のフローを説明する。`mcctrl` は McKernel で用いられるカーネルモジュールである。

1. 運用ソフトデーモンがジョブ実行開始時に `ihk_os_get_eventfd()` により監視イベント通知用 fd を取得する。（図の（１））
2. 運用ソフトデーモンは `epoll()` などで上記 fd 経由の通知を待つ。（図の（２））
3. 監視デーモン `ihkmond` が McKernel の状態を監視する。（図の（３））なお、McKernel の状態監視機能の詳細は”McKernel Specifications” に記載する。
4. 監視デーモン `ihkmond` が異常を検出し、上記 fd 経由でジョブ運用ソフトデーモンに異常を通知する。（図の（４））
5. 運用ソフトデーモンは通知を受けると `ihk_os_get_status()` により McKernel の状態を取得し、実際に異常状態にあることを確認する。（図の（５））
6. 運用ソフトデーモンは確認が取れると外部に異常を通知する。（図の（６））

戻り値

0 以上の値	`eventfd` のファイルデスクリプタ
`-EPERM`	操作に対する権限がない
`-ENOENT`	指定された OS インスタンスが存在しない
`-EINVAL`	不正なパラメータ

1.2.2.12 カーネルロード

書式

```
int ihk_os_load(int index, char *image)
```

説明

index で指定された OS インスタンスに image で指定されたファイル名のカーネルイメージをロードする。なお、ポスト京では OS インスタンスは 1 つのみ生成するため、OS インデックスは 0 を固定的に指定してよい。この呼び出しは特権ユーザのみ実行できる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	イメージがロードできない、割当メモリまたは割当 CPU が不足している
-EBUSY	OS インスタンスがブート済みである

1.2.2.13 カーネル引数設定

書式

```
int ihk_os_kargs(int index, char *kargs)
```

説明

index で指定された OS インスタンスに kargs に格納されているカーネル引数を渡す。この呼び出しは特権ユーザのみ実行できる。なお、hidos の文字列を含まない場合は-EINVAL を返す。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ
-EBUSY	OS インスタンスがブート済みである
-EFAULT	kargs にアクセスできない

1.2.2.14 設定リストによる OS インスタンスの作成と設定

書式

```
int ihk_create_os_str(int dev_index, int *os_index,
  const char *env_p, int num_env, const char *kernel_image,
  const char *default_kargs, char *err_msg);
```

説明

dev_index で指定された IHK デバイスに対し、envp と num_env で指定された設定リストに従って、OS インスタンスの作成と設定とを行う。本関数は特権ユーザのみが呼び出せる。

作成と設定のステップは以下の通り。

1. 資源を予約する。なお、資源が既に予約されていた場合は全ての資源を解放してから予約を行う。
2. OS インスタンスを作成する。インデックスは os_index に格納される。
3. OS インスタンスに予約した資源を割り当てる
4. LWK から Linux へのメッセージの経路を設定する
5. kernel_image で指定されたカーネルイメージをロードする
6. OS インスタンスのカーネル引数を設定する。なお、envp に指定がなかった場合は、default_kargs に格納された内容を用いる。

envp の内容は、設定を表す num_env 個の文字列が NULL 文字で結合されたものである。各設定は"KEY=VAL"の形式を持つ。設定可能な項目は以下の通り。なお、「必須」と記された項目がない場合は、本関数は-EINVAL を返す。また、これ以外の設定は無視される。

設定項目	設定内容
IHK_CPUS=<cpus> （必須）	<cpus>に指定された CPU を McKernel に割り当てる。<cpus>の書式は第 1.3.1.1 節に記載する。
IHK_RESERVE_MEM_BALANCED_ENABLE=(0|1) IHK_RESERVE_MEM_BALANCED_BEST_EFFORT=(0|1) IHK_RESERVE_MEM_BALANCED_VARIANCE_LIMIT=<limit_in_%> IHK_RESERVE_MEM_MIN_CHUNK_SIZE=<size> IHK_RESERVE_MEM_MAX_SIZE_RATIO_ALL=<cap_in_%> IHK_RESERVE_MEM_MEM_TIMEOUT=<timeout_in_second>	それぞれ、第 1.2.1.5 節に記載のメモリ予約の動作設定について、左辺の Key に対する値を右辺の値に設定する。
IHK_MEM=<mems> （必須）	<mems>に指定されたメモリを McKernel に割り当てる。<mems>の書式は第 1.3.1.4 節に記載する。
IHK_IKC_MAP=<ikc_map>	LWK から Linux へのメッセージの経路を<ikc_map>に指定されたものに設定する。<ikc_map>の書式は第 1.3.2.4 節に記載する。
IHK_KARGS=<kargs>	<kargs>をカーネル引数として LWK に渡す。

本関数は、エラー発生時は、OS インスタンスは全て削除された状態で、また CPU およびメモリは全て解放された状態で復帰する。本関数内での IHK インターフェイス関数の呼び出しでエラーが発生した場合は、err_msg にエラーメッセージが書き込まれる。書き込まれる内容には、呼び出し元のソースコードの名前と行番号、エラーを起こした関数の名前が含まれる。なお、本関数の呼び出し元が err_msg の領域を用意する。

戻り値

0	成功した
-EINVAL	envp の内容が適切でなかった。具体的には、必須の設定がなかった、あるいは設定値が不正であった。
-ENOMEM	メモリ不足が発生した
-EPERM	IHK デバイスを表すデバイスファイルまたは OS インスタンスを表すデバイスファイルにアクセスできなかった
-ENOENT	IHK デバイスまたは OS インスタンスが存在しなかった
-EFAULT	関数内部で使用する一時バッファにアクセスできなかった

1.2.2.15 ブート

書式

```
int ihk_os_boot(int index)
```

説明

index で指定された OS インスタンスのカーネルをブートする。この呼び出しは特権ユーザのみ実行できる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.16 シャットダウン

書式

```
int ihk_os_shutdown(int index)
```

説明

index で指定された OS インスタンスのカーネルをシャットダウンする。当該 OS インスタンスに割り当てられた資源は解放される。この関数は OS の状態が IHK_STATUS_INACTIVE に遷移したことを確認せずに復帰する。操作完了は ihk_os_get_status で確認できる。この呼び出しは特権ユーザのみ実行できる。

戻り値

0	シャットダウンに成功、または既にシャットダウン済
-EPERM	操作に対する権限がない
-ENOENT	指定された OS インスタンスが存在しない
-EBUSY	OS インスタンスが既にシャットダウン中である
-EINVAL	OS インスタンスがブート前である

1.2.2.17 OS 状態取得

書式

```
int ihk_os_get_status(int index)
```

説明

index で指定された OS インスタンスの状態を返す。
OS の状態は enum ihklib_os_status で表される。enum ihklib_os_status は以下のように定義される。

```
enum ihklib_os_status {
    IHK_STATUS_INACTIVE, // 起動前
    IHK_STATUS_BOOTING,  // 起動中
    IHK_STATUS_RUNNING,  // 起動後
    IHK_STATUS_SHUTDOWN, // シャットダウン中
    IHK_STATUS_PANIC,    // PANIC
    IHK_STATUS_HUNGUP,   // ハングアップ
    IHK_STATUS_FREEZING, // 一時停止状態へ移行中
    IHK_STATUS_FROZEN,   // 一時停止状態
};
```

戻り値

0 以上	OS 状態
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.18 カーネルメッセージサイズ取得

書式

```
ssize_t ihk_os_get_kmsg_size(int index)
```

説明

index で指定された OS インスタンスのカーネルメッセージ用バッファのサイズを返す。

戻り値

正の値	カーネルメッセージのサイズ
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.19 カーネルメッセージ取得

書式

```
int ihk_os_kmsg(int index, char *kmsg, size_t size_kmsg)
```

説明

index で指定された OS インスタンスのカーネルメッセージを kmsg にコピーする。size_kmsg の値は ihk_os_get_kmsg_size が返す値と等しい必要がある。なお、呼び出し元が kmsg の領域を用意する。

戻り値

0 以上の値	コピーしたバイト数
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.20 カーネルメッセージクリア

書式

```
int ihk_os_clear_kmsg(int index)
```

説明

index で指定された OS インスタンスのカーネルメッセージをクリアする。

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.21 NUMA ノード数取得

書式

```
int ihk_os_get_num_numa_nodes(int index)
```

説明

index で指定された OS インスタンスが利用可能な NUMA ノードの数を返す。

戻り値

1 以上	NUMA ノード数
0	エラー

1.2.2.22 空きメモリ量取得

書式

```
int ihk_os_query_free_mem(int index, unsigned long *memfree, int num_numa_nodes)
```

説明

index で指定された OS インスタンスの NUMA ノードごとの空きメモリ量を memfree で指定された配列に格納する。 num_numa_nodes には配列のサイズを指定する。呼び出し元が memfree の領域を用意する。

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.23 ページサイズ種数取得

書式

```
int ihk_os_get_num_pagesizes(int index)
```

説明

index で指定された OS インスタンスのページサイズ種数を返す。ihk_os_get_pagesizes() と組み合わせることでページサイズの表を取得できる。

戻り値

1 以上	ページサイズ種数
0	エラー

1.2.2.24 ページサイズ取得

書式

```
int ihk_os_get_pagesizes(int index, long *pgsizes, int num_pgsizes)
```

説明

index で指定された OS インスタンスのページサイズ表を pgsizes で指定された配列に格納する。num_pgsizes には配列のサイズを指定する。呼び出し元が pgsizes の領域を用意する。なお、ページサイズ表には、LWK で利用できるページサイズ以外のページサイズが含まれることがある。

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.25 統計情報取得

書式

```
int ihk_os_getrusage(int index, struct ihk_os_rusage *rusage)
```

説明

index で指定された OS インスタンスの呼び出し時点での統計情報を rusage に格納する。呼び出し元が rusage の領域を用意する。

struct ihk_os_rusage 型は以下のように定義される。なお、cpuacct_usage_percpu のインデックスは LWK での CPU 番号である。

```
struct ihk_os_rusage {
    unsigned long memory_stat_rss[IHK_MAX_NUM_PGSIZES];
    /* ユーザのページサイズごとの anonymous ページ使用量現在値（バイト単位） */
    unsigned long memory_stat_mapped_file[IHK_MAX_NUM_PGSIZES];
    /* ユーザのページサイズごとの file-backed ページ使用量現在値（バイト単位） */
    unsigned long memory_max_usage;
    /* ユーザのメモリ使用量最大値（バイト単位） */
    unsigned long memory_kmem_usage;
    /* カーネルのメモリ使用量現在値（バイト単位） */
    unsigned long memory_kmem_max_usage;
    /* カーネルのメモリ使用量最大値（バイト単位） */
    unsigned long memory_numa_stat[IHK_MAX_NUM_NUMA_NODES];
    /* NUMA ごとのユーザのメモリ使用量現在値（バイト単位） */
    unsigned long cpuacct_stat_system;
    /* システム時間（USER_HZ 単位） */
    unsigned long cpuacct_stat_user;
    /* ユーザ時間（USER_HZ 単位） */
    unsigned long cpuacct_usage;
    /* ユーザの CPU 時間（ナノ秒単位） */
    unsigned long cpuacct_usage_percpu[IHK_MAX_NUM_CPUS];
    /* コアごとのユーザの CPU 時間（ナノ秒単位）*/
    int num_threads;
    /* スレッド数現在値 */
    int max_num_threads;
    /* スレッド数最大値 */
};
```

memory_stat_rss および memory_stat_mapped_file のインデックスはサイズによるページ種であり、以下のように定義される。

```
enum ihk_os_pgsize {
        IHK_OS_PGSIZE_4KB,
        IHK_OS_PGSIZE_64KB,
        IHK_OS_PGSIZE_2MB,
        IHK_OS_PGSIZE_32MB,
        IHK_OS_PGSIZE_1GB,
        IHK_OS_PGSIZE_16GB,
        IHK_OS_PGSIZE_512MB,
        IHK_OS_PGSIZE_4TB,
        IHK_MAX_NUM_PGSIZES
};
```

戻り値

0	正常終了
-ENOENT	指定された OS インスタンスが存在しない
-EINVAL	不正なパラメータ

1.2.2.26 CPU PA 情報採取イベント登録

書式

```
int ihk_os_setperfevent(int index, struct ihk_perf_event_attr attr[], int n)
```

説明

index で指定された OS インスタンスにおいて attr，n で指定したイベントを収集する設定を行う。n はイベント種数で、ハードウェアが備える PA カウンタ数以下の値を指定する。呼び出し元が attr の領域を用意する。この関数は特権ユーザのみ呼び出せる。

ihk_perf_event_attr は以下のように定義される。

```
struct ihk_perf_event_attr{
    unsigned long config;       // ハードウェアで規定されるイベント番号
    unsigned disabled:1;        // 無効設定
    unsigned pinned:1;          // 常に収集対象とする
    unsigned exclude_user:1;    // ユーザモードで発生したイベントを計上しない
    unsigned exclude_kernel:1; // カーネルモードで発生したイベントを計上しない
    unsigned exclude_hv:1;      // hypervisor モードで発生したイベントを計上しない
    unsigned exclude_idle:1;    // idle 状態のイベントを計上しない
};
```

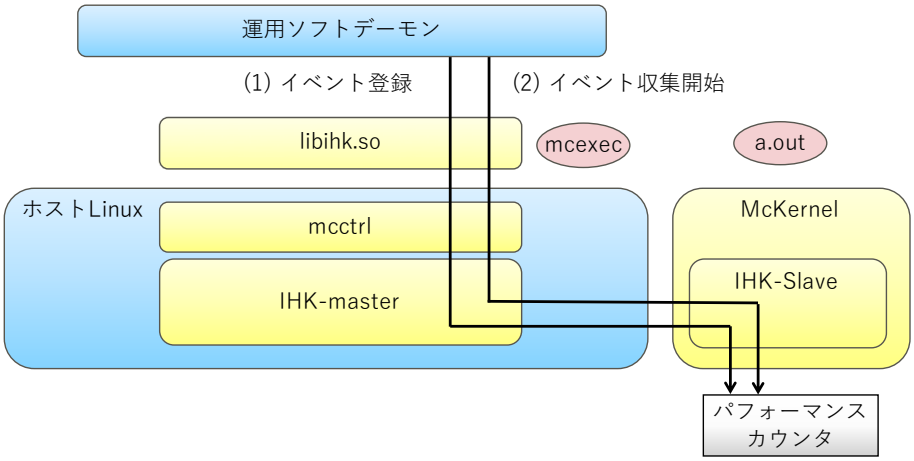

Figure 1.6: CPU PA 情報の収集開始のフロー

CPU PA 情報の収集は運用ソフトデーモンによって行われる。運用ソフトデーモンはジョブプロセス開始直前に収集を開始し、ジョブプロセス終了直後に収集を停止し値を回収する。図 1.6 を用いて LWK として McKernel が動作している際の収集開始のフローを説明する。

1. 運用ソフトデーモンが ihk_os_setperfevent() を用いて取得する CPU PA 情報（イベント）の設定を行う。（図の（1））
2. 運用ソフトデーモンが ihk_os_perfctl() を用いてイベント収集を開始する。（図の（2））

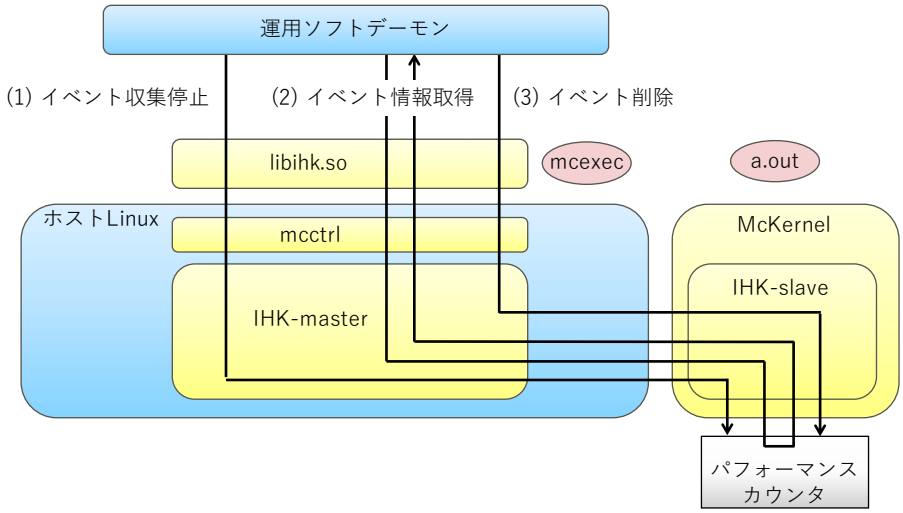

Figure 1.7: CPU PA 情報の収集停止と値回収のフロー

図 1.7 を用いて収集停止と値回収のフローを説明する。

1. 運用ソフトデーモンが `ihk_os_perfctl()` を用いてイベント収集を停止する。(図の (1))
2. 運用ソフトデーモンが `ihk_os_getperfevent()` を用いて CPU PA 情報の取得 (値の読み出し) を行う。(図の (2))
3. 運用ソフトデーモンが `ihk_os_perfctl()` を用いてイベントを削除する。(図の (3))

戻り値

0 または正の値	正常終了。登録に成功したイベント数を返す。
`-EPERM`	操作に対する権限がない
`-ENOENT`	指定された OS インスタンスが存在しない
`-EINVAL`	不正なパラメタ、または OS インスタンスが起動していない

1.2.2.27 CPU PA 情報収集開始停止

書式

```
int ihk_os_perfctl(int index, int comm)
```

説明

`index` で指定された OS インスタンスに対して `comm` で指定するサブコマンドを用いて PA イベント収集の制御を行う。この関数は特権ユーザのみ呼び出せる。

サブコマンドには以下の値を指定する。

値 (マクロ)	コマンドの意味	運用ソフトでの使用タイミング
PERF_EVENT_ENABLE	PA イベント収集開始	ジョブ開始時に使用
PERF_EVENT_DISABLE	PA イベント収集停止	ジョブ終了時に使用
PERF_EVENT_DESTROY	PA イベント削除	ジョブ終了時に使用

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	index で指定された OS インスタンスが存在しない
-EINVAL	不正なパラメタ

1.2.2.28 CPU PA 情報取得

書式

```
int ihk_os_getperfevent(int index, unsigned long *counter, int n)
```

説明

index で指定された OS インスタンスのイベント発生回数を要素数 n の配列 counter に格納する。n はイベント種数で、ihk_os_setperfevent の戻り値、すなわち登録に成功したイベント種数を指定する。呼び出し元が counter の領域を用意する。この関数は特権ユーザのみ呼び出せる。

戻り値

0	正常終了
-ENOENT	index で指定された OS インスタンスが存在しない
-EINVAL	不正なパラメタ

1.2.2.29 全 CPU 一時停止

書式

```
int ihk_os_freeze(unsigned long *os_set, int n)
```

説明

os_set で指定された OS インスタンスについて、全 CPU の一時停止状態への遷移を開始して即座に復帰する。対象 OS インスタンスの状態は、1 以上 CPU 数未満の数の CPU が一時停止状態へ遷移した時に IHK_STATUS_FREEZING に遷移し、全 CPU が一時停止状態へ遷移した時に IHK_STATUS_FROZEN に遷移する。操作が一定時間で完了しないケースは ihk_os_get_status() を用いて検出することができる。この場合は ihk_os_thaw() を用いて遷移をキャンセルすることができる。os_set は長さ n のビット列を指すポインタで、LSB から数えて第 i 番目のビットが 1 の場合は OS インデックスが i である OS インスタンスが操作の対象となる。なお、この関数は特権ユーザのみ呼び出せる。

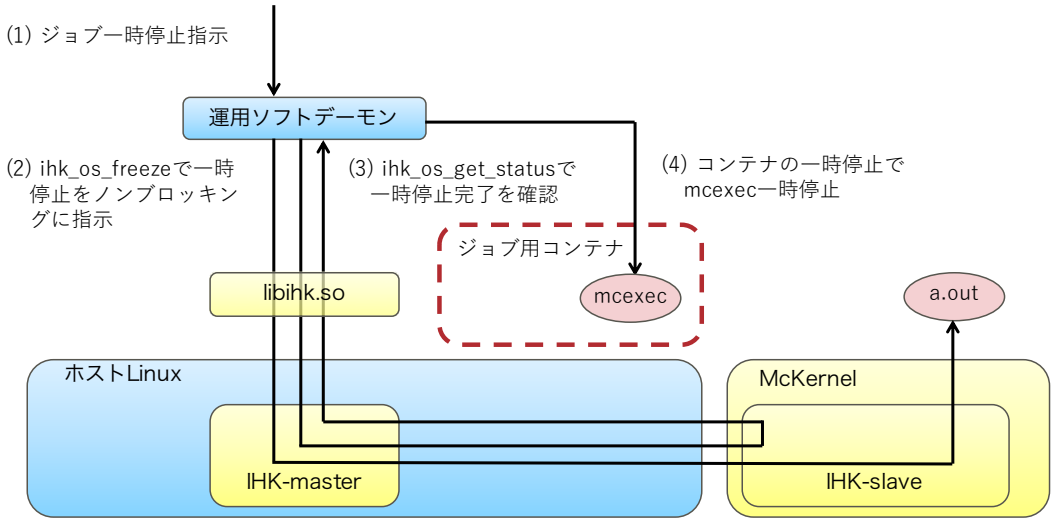

Figure 1.8: 全 CPU 一時停止のフロー

全 CPU 一時停止の動作フローを図 1.8 を用いて説明する。

1. 運用ソフトがノードの運用ソフトデーモンにジョブの一時停止を指示する。（図の（１））
2. 運用ソフトデーモンが `ihk_os_freeze()` で McKernel に全 CPU の一時停止をノンブロッキングに指示する。（図の（２））
3. 運用ソフトデーモンが `ihk_os_get_status()` で全 CPU の一時停止の完了を確認する。（図の（３））
4. 運用ソフトデーモンがコンテナの状態を変えることで mcexec (proxy process) を一時停止状態にする。（図の（４））

戻り値

0	正常終了
-EINVAL	OS インスタンスのステータスが IHK_STATUS_RUNNING、IHK_STATUS_FREEZING、IHK_STATUS_FROZEN 以外
-EBUSY	OS インスタンスのステータスが IHK_STATUS_FREEZING または IHK_STATUS_FROZEN
-EPERM	操作に対する権限がない
-ENOENT	index が示す OS インスタンスは存在しない

1.2.2.30 全 CPU 一時停止からの復帰

書式

```
int ihk_os_thaw(unsigned long *os_set, int n)
```

説明

os_set で指定された OS インスタンスについて、一時停止状態にあるか、一時停止状態へ遷移しつつある CPU を元の状態に戻す。また、OS の状態を IHK_STATUS_RUNNING にする。os_set は長さ n のビット列を指すポインタで、LSB から数えて第 i 番目のビットが 1 の場合は OS インデックスが i である OS インスタンスが操作の対象となる。なお、この関数は特権ユーザのみ呼び出せる。

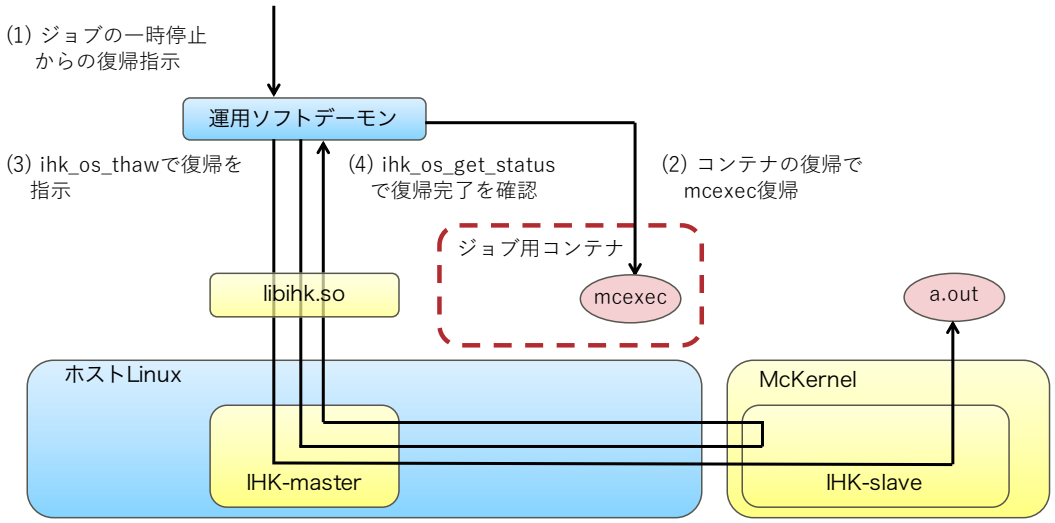

Figure 1.9: 全 CPU の一時停止からの復帰のフロー

全 CPU の一時停止からの復帰の動作フローを図 1.9 を用いて説明する。

1. 運用ソフトがノードの運用ソフトデーモンにジョブの一時停止からの復帰を指示する。（図の（１））

2. 運用ソフトデーモンがコンテナの状態を変えることで mcexec (proxy process) を一時停止状態から復帰させる。（図の（２））

3. 運用ソフトデーモンが ihk_os_thaw() で McKernel に全 CPU の一時停止からの復帰を指示する。（図の（３））

4. 運用ソフトデーモンが ihk_os_get_status() で全 CPU の一時停止からの復帰完了を確認する。（図の（４））

戻り値

0	正常終了
-EINVAL	OS インスタンスのステータスが IHK_STATUS_FREEZING、IHK_STATUS_FROZEN 以外
-EPERM	操作に対する権限がない
-ENOENT	index が示す OS インスタンスは存在しない

1.2.2.31 メモリダンプ採取

書式

```
int ihk_os_makedumpfile(int index, char *dump_file, int dump_level, int interactive)
```

説明

index で指定された OS インスタンスについて、dump_level で指定されたメモリ領域を dump_file で指定したファイルに出力する。dump_level の指定方法は以下の通り。

0	IHK が OS インスタンスに割り当てたメモリ領域を出力する。
24	カーネルが使用しているメモリ領域を出力する。

interactive が 1 の場合は、interactive mode 向けのファイルを出力する。このモードでは、ダンプ解析ツールはデバッグ対象マシンのメモリを直接参照して解析を行う。なお、この関数は特権ユーザのみ呼び出せる。

戻り値

0	正常終了
-EPERM	操作に対する権限がない
-ENOENT	dump_file の値が NULL、または dump_file が長さ 0 の文字列を指している、または dump_file に含まれるディレクトリが存在しない
-EACCESS	dump_file で指定したファイルについて、ディレクトリは存在するがファイルが作成できない
-EEXIST	dump_file で指定したファイルが既に存在する
-EINVAL	不正なパラメタ。index が負の場合を含む。
-ENODEV	index で指定される OS インスタンスが存在しない
-EPERM	index で指定される OS インスタンスにアクセスできない

1.2.3 LWK 向け OS 初期化機能

1.2.3.1 Get Number of NUMA Nodes

Synopsis

```
int ihk_mc_get_nr_numa_nodes();
```

Description

This function returns the number of NUMA nodes assigned by IHK.

Return Value

> 0	The number of the NUMA nodes

1.2.3.2 Get NUMA Node Information

Synopsis

```
int ihk_mc_get_numa_node(int id, int *linux_numa_id, int *type);
```

id	**input**	NUMA id
linux_numa_id	**output**	Linux NUMA id
type	**output**	Memory type

Description

The host Linux NUMA id and the memory type of the NUMA node specified by `id` is stored to `linux_numa_id` and `type`, respectively. Each of the values is not stored when the corresponding pointer is NULL.

Return Value

0	Success
-1	`id` is not valid

1.2.3.3 Get NUMA id

Synopsis

```
int ihk_mc_get_numa_id();
```

Description

Returns NUMA id of the CPU on which the caller is running on.

Return Value

≥ 0	NUMA id

1.2.3.4 Get Distance between NUMA Nodes

Synopsis

```
int ihk_mc_get_numa_distance(int i, int j);
```

Description

Returns the distance between the NUMA nodes specified by `i` and `j`. The distance matrix could be the same as Linux.

Return Value

≥ 0	The distance between the NUMA nodes

1.2.3.5 Get Number of Memory Chunks

Synopsis

```
int ihk_mc_get_nr_memory_chunks();
```

Description

This function returns the number of physical memory chunks assigned by IHK.

Return Value

≥ 0	The number of memory chunks

1.2.3.6 Get Memory Chunk Information

Synopsis

```
int ihk_mc_get_memory_chunk(int id, unsigned long *start, unsigned
            long *end, int *numa_id);
```

Description

The start physical address, end physical address and the NUMA id are stored to `start`, `end`, `numa_id`, respectively. Each of the values is not stored when the corresponding pointer is NULL.

Return Value

0	Success
-1	`id` is not valid

1.2.3.7 Get Number of Cores

Synopsis

```
int ihk_mc_get_nr_cores();
```

Description

This function returns the number of CPU cores assigned by IHK.

Return Value

≥ 0	The number of CPU cores

1.2.3.8 Get Core Information

Synopsis

```
int ihk_mc_get_core(int id, unsigned long *linux_core_id, unsigned
          long *hw_id, int *numa_id);
```

Description

The host Linux CPU id, the hardware id and the LWK NUMA id of the CPU core specified by `id` are stored to `linux_core_id`, `hw_id`, `numa_id`, respectively. Each of the values is not stored when the corresponding pointer is NULL. The hardware id corresponds to the hardware APIC id in `x86_64` architecture.

Return Value

0	Success
-1	`id` is not valid

1.2.3.9 Get IKC Destination CPU

Synopsis

```
int ihk_mc_get_ikc_cpu(int id);
```

Description

This function returns the Linux id of the CPU to which the CPU specified by `id` sends IKC messages.

Return Value

≥ 0	The Linux id of the IKC destination CPU

1.2.3.10 Get Kernel Arguments

Synopsis

```
char *ihk_get_kargs();
```

Description

This function returns the pointer to the buffer containing the kernel arguments given by the IHK master.

Return Value

> 0	The pointer to the kernel argument string

1.2.3.11 Get Information of Kernel Message Buffer

Synopsis

```
int ihk_get_kmsg_buf(unsigned long *addr, unsigned long *size);
```

Description

The physical address and the size of the kernel message buffer are stored to `addr` and `size`, respectively. This function is supposed to be called when initializing LWK.

Return Value

0	Success

1.2.3.12 Boot a Core

Synopsis

```
void ihk_mc_boot_cpu(int cpu_id, unsigned long pc);
```

Description

This function makes the CPU specified by `cpu_id` (Physical APIC CPU ID) start execution from the virtual address specified by `pc`.

1.2.4 LWK 向け Inter-Kernel Communication (IKC) 機能

1.2.4.1 Initialize Master Channel on the IHK-master side

Synopsis

```
int ihk_ikc_master_init(ihk_os_t os);
```

Description

This function is called by Linux and initializes the master channel connected to the OS specified by `os`. The master channel is present at boot time and used for creating / destroying more channels. The created channel is called regular channel and used for the communication. LWK sends a connection request to Linux through the master channel to create a regular channel.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.2 Initialize Master Channel on the IHK-slave side

Synopsis

```
void ihk_ikc_master_init(void);
```

Description

This function is called by LWK and initializes the master channel connected to Linux.

1.2.4.3 Listen to Connection Requests

Synopsis

```
int ihk_ikc_listen_port(ihk_os_t os, struct ihk_ikc_listen_param *param);
```

Description

This function makes the master channel listen to the remote OS specified by `os` to create a channel with the parameters of `param`.

`struct ihk_ikc_listen_param` specifies parameters for a channel to be created and defined as follows.

```
struct ihk_ikc_listen_param {
    int (*handler)(struct ihk_ikc_channel_info *);
    int port;
    int pkt_size;
    int queue_size;
    int magic;
    int recv_cpu;
};
```

`handler`	A function called when accepting an incoming connection request
`port`	Port number
`pkt_size`	Packet size
`queue_size`	Queue size
`magic`	Magic number for identification of the communication initiator
`recv_cpu`	CPU ID of the listener

An IHK user must set the first four fields before passing it to `ihk_ikc_listen_port`. The IHK user must define function which is set to `handler` field of this structure. `handler` is called when accepting an incoming connection request and it is expected to set `packet_handler` field of the argument. The value of the field is then copied to `handler` field of `ihk_ikc_channel_desc` and becomes the call-back function which is called when detecting an arrival of a packet. This accept-time call-back mechanism is used to create a table which is indexed by a CPU ID and returns the channel bound to the CPU.

`ihk_ikc_channel_info` is an intermediate object used by the accept-time call-back function to pass the packet-arrival-time call-back function to the channel as described above and is defined as follows.

```
struct ihk_ikc_channel_info {
    struct ihk_ikc_channel_desc *channel;
    ihk_ikc_ph_t packet_handler;
};
```

`channel` is only used internally. `packet_handler` is a pointer to the packet-arrival-time call-back function and is set by the accept-time call-back function.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.4 Send a Connection Request

Synopsis

```
int ihk_ikc_connect(ihk_os_t os, struct ihk_ikc_connect_param *p);
```

Description

This function sends a connection request to the remote OS specified by `os` via the master channel to create a regular channel with the parameters of `p`. The created channel is stored to `p->channel`. The receiver side detects the arrival of a packet either by calling non-blocking receive function or by notification (IRQ) and call-back mechanism.

`ihk_ikc_connect_param` specifies the parameters for the channel to be created and is defined as follows.

```
struct ihk_ikc_connect_param {
    int port;
    int pkt_size;
    int queue_size;
    int magic;
    ihk_ikc_ph_t                handler;
    struct ihk_ikc_channel_desc *channel;
};
```

`port`	Port number
`pkt_size`	Packet size
`queue_size`	Queue size
`magic`	Magic number for identification of the communication initiator
`handler`	Packet handler called when calling `ihk_ikc_recv_handler`
`channel`	Channel descriptor which is set when connected

An IHK user must set `port, pkt_size, queue_size, magic, handler` fields. `channel` field is set to the descriptor of the channel.

`ihk_ikc_channel_desc` is an opaque type represeinting an IKC channel.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.5 Register a Call-Back Function for Receive Events

Synopsis

```
int ihk_ikc_recv_handler(struct ihk_ikc_channel_desc *channel, ihk_ikc_ph_t
          h, void *harg, int opt);
```

Description

This function registers to the channel specified by `channel` a call-back function specified by `h` and an argument passed to it specified by `harg`. The call-back function is called when a packet arrives. The call-back function handles multiple packets that have arrived and performs only one notification action (e.g. sends an interrupt to the sender side). `NO_COPY` bit of `opt` should be set to zero when the packet is accessed by the code outside the handler.

`ihk_ikc_ph_t` represents the call-back function which is called when detecting an arrival of an incoming packet and is defined as follows.

```
typedef int (*ihk_ikc_ph_t)(struct ihk_ikc_channel_desc *, void *, void *);
```

It takes the descriptor of IKC channel as the first argument, the address of the incoming packet as the second argument and `harg` passed by `ihk_ikc_recv_handler` as the third argument.

`harg` supports the use case where an IHK user can bind an abstracted channel structure used in the IHK user module to the IKC channel so that the handler can identify the abstracted channel through which the packet has arrived. A reverse search table which returns the abstracted channel given the IKC channel ID is needed if `harg` is not passed down to the call-back function.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.6 Send a Packet

Synopsis

```
int ihk_ikc_send(struct ihk_ikc_channel_desc *channel, void *p, int opt);
```

Description

This function sends a packet specified by `p` through a regular channel specified by `channel`. It performs a notification action to the receiver side (e.g. sends an interrupt) when `IKC_NO_NOTIFY` bit of `opt` is zero. It is safe to overwrite memory area pointed by `p` after calling `ihk_ikc_send` because the packet is memory-copied before sending. It is the IHK user's responsibility to perform flow control.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.7 Disconnect a Channel

Synopsis

```
int ihk_ikc_disconnect(struct ihk_ikc_channel_desc *c);
```

Description

This function disconnects a regular channel specified by `c`.

Return Value

0	Success
$\neq 0$	Error number

1.2.4.8 Destroy a Channel

Synopsis

```
void ihk_ikc_destroy_channel(struct ihk_ikc_channel_desc *c);
```

Description

This function destroys the master channel or a regular channel specified by `c`.

1.2.5 Linux ドライバ向け機能

1.2.5.1 制御レジスタリード

書式

```
int ihk_os_read_cpu_register(ihk_os_t os, int cpu, struct ihk_os_cpu_register
*desc)
```

説明

`os` で指定する OS インスタンスの `cpu` で指定する CPU の `desc` で指定する制御レジスタ値を `desc->val` へ非同期で読み込む。完了は `desc->sync` のゼロ以外の値への変化で検知できる。なお、`cpu` には LWK での番号を指定する。また、呼び出し元が `desc` の領域を用意する。
`struct ihk_os_cpu_register` は以下のように定義される。

```
struct ihk_os_cpu_register {
    unsigned long addr;
    /* メモリマップの制御レジスタのアドレス。アーキテクチャ固有の値
       をそのまま用いる。*/
    unsigned long addr_ext;
    /* CPU の制御レジスタ番号。アーキテクチャ固有の値をそのまま用いる。*/
    unsigned long val;
    /* ihk_os_write_cpu_register() : 制御レジスタに書き込む値
       ihk_os_read_cpu_register() : 制御レジスタ値の記録先 */
    atomic_t sync;
    /* 制御レジスタへの操作完了を示す。0 は未完了を意味し、0 以外は完了を
       意味する。*/
};
```

利用のステップを図 1.10 を用いて説明する。

1. LWK 上で動作するライブラリが LWK からのオフロード経由で Linux ドライバにレジスタ操作を指示する場合は、`ihk_get_request_os_cpu()` を用いてオフロード元 OS インスタンスと CPU 番号を取得する。こうすることで、操作先の OS インスタンス偽装を防ぐ。（図の（1））

2. Linux ドライバが操作完了を示す変数を未完了 (0) に設定してから、`ihk_os_read_cpu_register()` または `ihk_os_write_cpu_register()` でレジスタを非同期に操作する。（図の（2））

3. IHK または LWK が上記変数の値を変化させることにより操作完了を Linux ドライバに通知する。（図の（３））

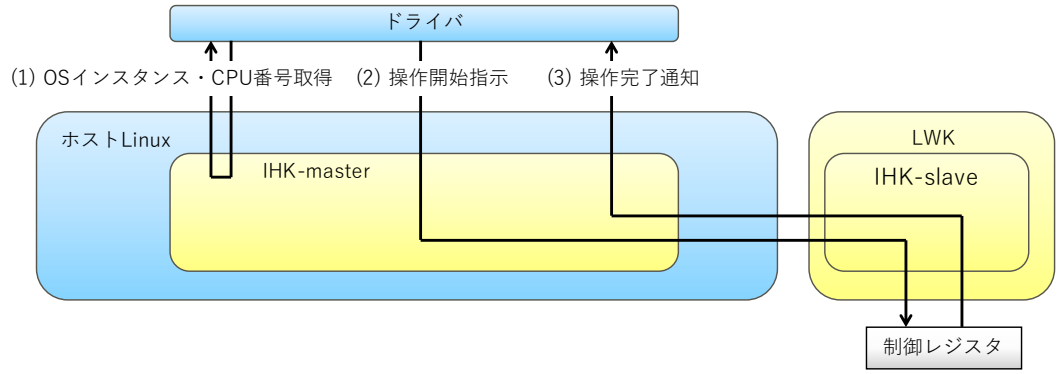

Figure 1.10: 制御レジスタの操作ステップ

戻り値

0	正常終了
-EINVAL	os にアクセスできない、または cpu が LWK に割り当てられている CPU ではない、または指定されたアドレスまたは番号のレジスタは存在しない
-EFAULT	desc にアクセスできない

1.2.5.2 制御レジスタライト

書式

```
int ihk_os_write_cpu_register(ihk_os_t os, int cpu, struct ihk_os_cpu_register
*desc)
```

説明

os で指定する OS インスタンスの cpu で指定する CPU の desc で指定する制御レジスタへ desc->val で指定する値を非同期で書き込む。完了は desc->sync のゼロ以外の値への変化で検知できる。 なお、cpu には LWK での番号を指定する。

戻り値

0	正常終了
-EINVAL	os にアクセスできない、または cpu が LWK に割り当てられている CPU ではない、または指定されたアドレスまたは番号のレジスタは存在しない
-EFAULT	desc にアクセスできない

1.2.5.3 オフロード元 OS インスタンス取得

書式

```
int ihk_get_request_os_cpu(ihk_os_t *os, int *cpu)
```

説明

システムコール移譲などのオフロード経由で本関数の呼び出しを行った場合、オフロード元 LWK の OS インスタンスを os に、CPU 番号を cpu に返す。なお、cpu は McKernel での番号である。

戻り値

0	正常終了
-EINVAL	オフロードにより当該呼び出しにいたっていない
-EFAULT	os または cpu にアクセスできない

1.3 コマンド・デーモン仕様

1.3.1 管理者向け資源管理機能

1.3.1.1 Reserve CPUs

Synopsis

```
ihkconfig <dev index> reserve cpu <CPU id list>
```

Description

This command reserves specific CPU cores for the IHK framework. <dev index> identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module, and <CPU id list> is the following format: <CPU logical id>,...,<CPU logical id> or <CPU logical id> - <CPU logical id> (must be a positive range in ascending order) or a mixture of the two: <CPU logical id>,...,<CPU logical id> - <CPU logical id>. CPU logical ID begins at 0 and the maximum value is "number of CPUs in system - 1". An actual example of usage would be:

```
$ ihkconfig 0 reserve cpu 24-31
```

The reserve operation may be executed multiple times adding CPU logical ID cores as required.

Exit Status

0	Success
Other than 0	Failure

1.3.1.2 Query CPUs

Synopsis

```
ihkconfig <dev index> query cpu
```

Description

This command queries which CPU cores the IHK framework has reserved. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module.

The command returns the list of CPUs in the same format as the above reservation command.

Exit Status

0	Success
Other than 0	Failure

1.3.1.3 Release CPUs

Synopsis

```
ihkconfig <dev index> release cpu <CPU id list>
```

Description

This command releases the specific CPU cores from the IHK framework. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module, and `<CPU id list>` is the following format: `<CPU logical id>`,...,`<CPU logical id>` or `<CPU logical id>` - `<CPU logical id>` (must be a positive range in ascending order) or a mixture of the two: `<CPU logical id>`,...,`<CPU logical id>` - `<CPU logical id>`. CPU logical ID begins at 0 and the maximum value is "number of CPUs in system - 1". An actual example of usage would be:

```
$ ihkconfig 0 release cpu 24-31
```

The release operation may be executed multiple times removing CPU logical ID cores from IHK as required.

Exit Status

0	Success
Other than 0	Failure

1.3.1.4 Reserve Memory

Synopsis

```
ihkconfig <dev index> reserve mem <memory description>
```

Description

This command reserves memory for the IHK framework. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module. You can specify the size and the NUMA nodes with the `<memory description>` argument by using the following format:

```
(<size>[<unit>]|ALL)[@<NUMA-id>][,(<size>[<unit>]|ALL)[@<NUMA-id>]...]
```

where `<size>` is the number of bytes requested, optionally followed by a unit (`M`, `G` and `T` are available, meaning MiB, GiB and TiB, respectively). Moreover, the optional `@` symbol that can be followed by a decimal number denotes the targeted NUMA node, where the default NUMA node is 0. Specifying `ALL` in the size field means request for best effort maximum.

Here is an example which allocates 2 Gigabytes from NUMA node 1:

```
$ ihkconfig 0 reserve mem 2G@1
```

The reserve operation may be executed multiple times adding physical memory as required. The operation may fail in case the system wide available memory is less than the amount requested. IHK reserves the memory area of the requested size using the following alorithm. We denote by s the requested size and by t the total size of the reserved memory-chunks.

1. Find the largest memory chunk with the size of less than or equal to $s - t$ and reserve it.
2. Repeat the above step until t becomes equals to s.

Exit Status

0	Success
Other than 0	Failure

1.3.1.5 Query Memory

Synopsis

```
ihkconfig <dev index> query mem
```

Description

This command queries the amount of the memory that the IHK framework has reserved and has not been assigned to an OS instance. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module.

The command returns the list of memory regions in the same format as the above reservation command.

Exit Status

0	Success
Other than 0	Failure

1.3.1.6 Release Memory

Synopsis

```
ihkconfig <dev index> release mem <memory list>
```

Description

This command releases memory from the IHK framework. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module. The `<memory list>` takes the same format as the above reserve command. `all` means to release all of the reserved memory. An actual example of usage would be:

```
$ ihkconfig 0 release mem 1G@1
```

The release operation may be executed multiple times freeing physical memory as required. The operation may fail in case the IHK reserved memory is less than the amount requested.

Exit Status

0	Success
Other than 0	Failure

1.3.1.7 Create OS instance

Synopsis

```
ihkconfig <dev index> create
```

Description

This command creates an OS instance over the specific IHK device. `<dev index>` identifies the IHK device file that appears as the result of the insertion of the IHK-master driver module. An actual example of usage would be:

```
$ ihkconfig 0 create
```

Unless an error occurs, the command returns an index `X` which will denote the specific OS device file with path name of `/dev/mcosX`.

Exit Status

0	Success
Other than 0	Failure

1.3.1.8 Destroy OS instance

Synopsis

```
ihkconfig <dev index> destroy <os index>
```

Description

This command destroys the OS instance specified by `<os index>` residing on the IHK device specified by `<dev index>`. The resources assigned to the OS instance are released before destroying it. An actual example of usage would be:

```
$ ihkconfig 0 destroy 2
```

Destroying an operating system instance requires that all internal IHK structures associated with the OS are not being used and the operation may fail otherwise. Internal IHK resources may be used by the *mcexec* process and thus terminating those processes before destroying an OS instance is required.

Exit Status

0	Success
Other than 0	Failure

1.3.1.9 OS インスタンス一覧取得

書式

```
ihkconfig <dev index> get os_instances
```

説明

IHK デバイス`/dev/mcd<dev index>`上に存在する OS インスタンスの OS インデックスを以下の形式で出力する。

```
<os_index>[,<os_index>...]
```

なお、ポスト京では OS インスタンスは 1 つのみ生成するため、本関数で OS インスタンスの存在を確認した後は、OS インデックスには 0 を固定的に指定してよい。

エラー時出力

文字列	意味
Error: Invalid argument	不正なパラメータ

Exit Status

0	正常終了
0 以外	エラー

1.3.2 管理者向け OS 管理機能

`ihkosctl` is responsible of providing a simple interface for interacting with IHK OS instance device files, i.e., those named as `/dev/mcosX`.

1.3.2.1 Assign CPUs

Synopsis

```
ihkosctl <os index> assign cpu <CPU id list>
```

Description

This operation assigns CPU cores to an OS instance. `<os index>` identifies the OS index that has been returned by the OS creation operation, and `<CPU id list>` is the following format: `<CPU logical id>`,...,`<CPU logical id>` or `<CPU logical id>` - `<CPU logical id>` (must be a positive range in ascending order) or a mixture of the two: `<CPU logical id>`,...,`<CPU logical id>` - `<CPU logical id>`. CPU logical ID begins at 0 and the maximum value is "number of CPUs in system - 1". Note that only CPU logical IDs which have been reserved for the IHK framework are available. An actual example of usage would be:

```
$ ihkosctl 0 assign cpu 2-8
```

In which example, CPU cores 2, 3, 4, 5, 6, 7, 8 are assigned to OS instance 0. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.2 Query CPUs

Synopsis

```
ihkosctl <os index> query cpu
```

Description

This command queries the CPUs that are assigned to the OS instance specified by `<os index>`. The command returns the list of CPUs in the same format as the above assign command.

Exit Status

0	Success
Other than 0	Failure

1.3.2.3 Release CPUs

Synopsis

```
ihkosctl <os index> release cpu <CPU id list>
```

Description

This command releases the CPUs specified by `<CPU id list>` that are assigned to the OS instance specified by `<os index>`. The `<CPU id list>` takes the same format as the above assign command. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.4 Set IKC Map

Synopsis

```
ihkosctl <os index> set ikc_map <IKC map>
```

Description

This command sets up the IKC mapping between LWK CPUs and Linux CPU. `<os index>` identifies the OS index that has been returned by the OS creation operation, and `<IKC map>` has the following format: `<CPU list>:<CPU logical id>[+<CPU list>:<CPU logical id>...]`. Refer to Section 1.3.2.1 for the format of `<CPU list>`. Each `<CPU list>:<CPU logical id>` denotes the McKernel CPUs denoted by `<CPU list>` send IKC messages to the Linux CPU denoted by `<CPU logical id>`.

An actual example of usage would be:

```
$ ihkosctl 0 ikc_map 1-3:0+5-7:4+9-11:8+13-15:12
```

In this example, McKernel CPUs 1, 2, 3 send IKC messages to Linux CPU 0 and McKernel CPU 5, 6, 7 to Linux CPU 4 and so on. Only privileged user can perform this operation.

See Section **??** for the detail of the IKC mapping.

Exit Status

0	Success
Other than 0	Failure

1.3.2.5 Get IKC Map

Synopsis

```
ihkosctl <os index> get ikc_map
```

Description

This command prints out the IKC mapping between LWK CPUs and Linux CPU of the OS instance specified by `<os index>`. The output format representing the IKC mapping is explained in Section 1.3.2.4.

Error output

String	Meaning
Error: OS instance not found	The OS instance specified does not exit
Error: Invalid argument	Invalid parameter

Exit Status

0	Success
Other than 0	Failure

1.3.2.6 Assign Memory

Synopsis

```
ihkosctl <os index> assign mem <memory list>
```

Description

This command allocates physical memory to an OS instance. `<os index>` identifies the OS index that has been returned by the OS creation operation, the IHK OS instance's index that has been returned as the result of the creation operation, and `<memory list>` is given in the following format: `X[M|G|T][@P][,Y[M|G|T][@Q]..]|all`, where `X` is a decimal number denoting the number of bytes requested, unless one of the standard metric prefixes is attached (i.e., `M` as Mega, `G` as Giga, or `T` as Terra), in which case it stands for the specified metric. Moreover, the optional `@` symbol that can be followed by a decimal number denotes the targeted NUMA node, where the default NUMA node is 0. `all` means request for all of the reserved memory. Note that only memory which have been reserved for the IHK framework is available. An actual example of usage would be:

```
$ ihkosctl 0 assign mem 1G@0,1G@1
```

In which example, 1 GB of memory from NUMA node 0 and 1 GB from NUMA node 1 are assigned to OS instance 0. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.7 Query Memory

Synopsis

```
ihkosctl <os index> query mem
```

Description

This command queries the memory areas that are assigned to the OS instance specified by `<os index>`. The command returns the memory list in the same format as the above assign command.

Exit Status

0	Success
Other than 0	Failure

1.3.2.8 Release Memory

Synopsis

```
ihkosctl <os index> release mem <memory list>
```

Description

This command releases the memory areas specified by `<memory list>` that are assigned to the OS instance specified by `<os index>`. The `<memory list>` takes the same format as the above assign command. `all` means to release all of the assigned memory. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.9 Load Kernel Image

Synopsis

```
ihkosctl <os index> load <filename>
```

Description

This command loads a specific kernel image into an OS instance. `<os index>` identifies the OS index that has been returned by the OS creation operation, `<filename>` specifies the path to the kernel image intended to be loaded for the OS instance. An actual example of usage would be:

```
$ ihkosctl 0 load /home/example/lwk/kernel.elf.img
```

In which example, /home/example/lwk/kernel.elf.img is loaded. As mentioned earlier, an IHK compatible kernel image is a standard ELF binary linked against the IHK-slave provided library so that it can interact with the other components in the system. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.10 Set Kernel Arguments

Synopsis

```
ihkosctl <os index> kargs <kernel arguments>
```

Description

This command assigns kernel command line parameters to an OS instance, which will be passed to the kernel during boot. `<os index>` identifies the OS index that has been returned after the OS creation operation and `<kernel arguments>` is a list of comma separated values. An actual example of usage would be:

```
$ ihkosctl 0 kargs foo=bar,foo2=bar2
```

In which example, foo=bar and foo2=bar2 are the boot time arguments. Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.11 Boot Kernel

Synopsis

```
ihkosctl <os index> boot
```

Description

This command instructs the OS instance to boot the kernel image specified earlier. `<os index>` identifies the OS index that has been returned after the OS creation operation. An actual example of usage would be:

```
$ ihkosctl 0 boot
```

Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.12 Query Free Memory

Synopsis

```
ihkosctl <os index> query_free_mem
```

Description

This command queries the amounts of free memory areas that are assigned to the OS instance specified by `<os index>`. The command returns the memory list in the same format as `ihkosctl (assign mem)` command.

Exit Status

0	Success
Other than 0	Failure

1.3.2.13 Display Kernel Message

Synopsis

```
ihkosctl <os index> kmsg
```

Description

This command obtains the kernel message buffer from the OS instance. `<os index>` identifies the OS index that has been returned after the OS creation operation. An actual example of usage would be:

```
$ ihkosctl 0 kmsg
```

Exit Status

0	Success
Other than 0	Failure

1.3.2.14 Clear Kernel Message

Synopsis

```
ihkosctl <os index> clear_kmsg
```

Description

This command clears the kernel message buffer of the OS instance. `<os index>` identifies the OS index that has been returned after the OS creation operation. An actual example of usage would be:

```
$ ihkosctl 0 clear_kmsg
```

Exit Status

0	Success
Other than 0	Failure

1.3.2.15 Shutdown Kernel

Synopsis

```
ihkosctl <os index> shutdown
```

Description

This command shuts down the OS instance specified by `<os index>`. The resources assigned to the OS instance are released before shutting it down. An actual example of usage would be:

```
$ ihkosctl 0 shutdown
```

Only privileged user can perform this operation.

Exit Status

0	Success
Other than 0	Failure

1.3.2.16 OS 状態取得

書式

```
ihkosctl <os_index> get status
```

説明

<os_index>で指定された OS インスタンスの OS 状態を出力する。各 OS 状態に対応する文字列と意味は以下の通り。

文字列	意味
INACTIVE	起動前
BOOTING	起動中
RUNNING	起動後、停止前
SHUTDOWN	シャットダウン中
PANIC	PANIC
HUNGUP	ハングアップ
FREEZING	一時停止状態へ移行中
FROZEN	一時停止状態

エラー時出力

文字列	意味
Error: OS instance not found	指定された OS インスタンスが存在しない
Error: Invalid argument	不正なパラメタ

Exit Status

0	正常終了
0 以外	エラー

1.3.2.17　メモリダンプ採取

書式

```
ihkosctl <os_index> dump [-d <dump_level>] [<file_name>] [--interactive|-i]
```

オプション

-d <dump_level>	ダンプ対象とするメモリ領域の種類を<level>に設定する。設定可能な値は以下の通り。 0: IHK が McKernel に割り当てたメモリ領域を出力する。 24: カーネルが使用しているメモリ領域を出力する。 指定がなかった場合は 0 が用いられる。
<file_name>	出力先ファイル名。指定がなかった場合は mcdump_YYYYmmddHHMMSS が用いられる。
--interactive|-i	Interactive mode 向けのファイルを出力する。このモードでは、ダンプ解析ツールはデバッグ対象マシンのメモリを直接参照して解析を行う。

説明

<os_index>で指定された OS インスタンスの<dump_level>で指定されたメモリ領域を<file_name>で指定されたファイルに出力する。なお、このコマンドは特権ユーザのみ実行できる。

エラー時出力

文字列	意味
Error: No such file or directory	〈ダンプファイル名〉に含まれるディレクトリが存在しない。
Error: Permission denied	〈ダンプファイル名〉で指定したファイルについて、ディレクトリは存在するがファイルが作成できない。
Error: File exists	〈ダンプファイル名〉で指定したファイルが既に存在する。
Error: Invalid argument	不正なパラメタ。
Error: OS instance not found	〈OS インデックス〉で指定される OS インスタンスが存在しない。
Error: Operation not permitted	〈OS インデックス〉で指定される OS インスタンスにアクセスできない。

Exit Status

0	正常終了
0 以外	エラー

1.3.2.18　カーネルメッセージリダイレクト・ハングアップ検知デーモン

書式

```
ihkmond [-k <redirect_kmsg>] [-i <mon_interval>] [-f <facility>]
```

オプション

`-k <redirect_kmsg>`	カーネルメッセージの/dev/log へのリダイレクト有無を指定する。0 が指定された場合はリダイレクトを行わず、0 以外が指定された場合はリダイレクトを行う。指定がない場合はリダイレクトを行う。
`-i <mon_interval>`	ハングアップ検知のために OS 状態を確認する時間間隔を秒単位で指定する。-1 が指定された場合はハングアップ検知を行わない。指定がない場合は 600 秒が用いられる。
`-f <facility>`	syslog プロトコルの facility を指定する。指定がない場合は `LOG_LOCAL6` を用いる。

説明

カーネルメッセージを取得し `syslog()` を用いて`/dev/log` に書き込む。syslog プロトコルの facility は`<facility>`に設定される。また、`<mon_interval>`秒ごとに `ioctl()` の `IHK_OS_DETECT_HUNGUP` サブコマンドを用いて OS 状態を確認する。2 回連続して、通常時間がかからない処理であって、かつカーネルの処理の実行中であることが確認された場合はハングアップと判断する。そして、運用ソフトが `ihk_os_get_eventfd()` で `eventfd` を取得している場合はそれに対して報告する。なお、本デーモンは任意のタイミングで起動してよい。これは、本デーモンは OS インスタンスの作成を検知して動作を開始するためである。

戻り値

0	正常終了
0 以外	エラー

Chapter 2

LWK起動

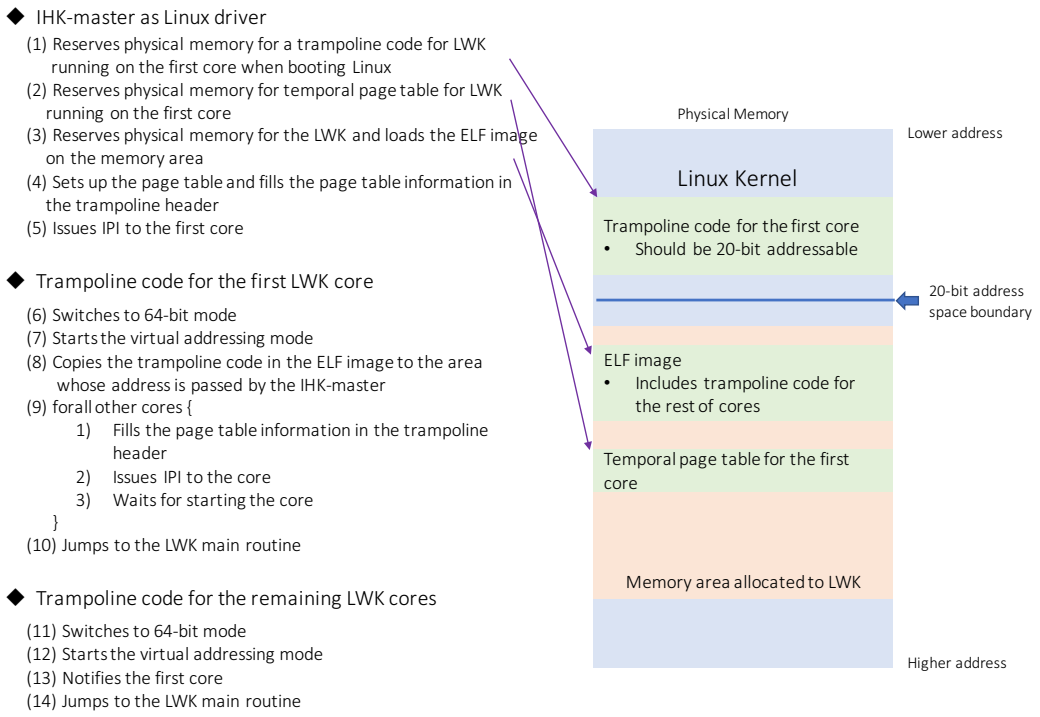

Figure 2.1: **Boot sequence of cores for LWK.**

Fig. 2.1 explains the steps for Linux to boot an LWK using IHK. All of these are performed by IHK. Two particular details deserve further discussion. First, the trampoline code must fit in 20-bit address space because an IPI is used to make the first LWK core jump to the trampoline code and the current x86 restriction for the address field in the IPI demands 20-bit address representation. Second, the location of the temporal page table must fit in 32-bit address space because the control register (CR3) has 32-bit width when a CPU core is in 32-bit mode in the early phase of the trampoline execution.

When the IHK-slave passes the control to the LWK main routine, it is given the physical address of the kernel arguments as the first argument and the physical address of the kernel text as the second argument. IHK allocates a dedicated page as stack area and the stack pointer is set to that page. Fig. 2.2 shows the memory map set at the time of entering

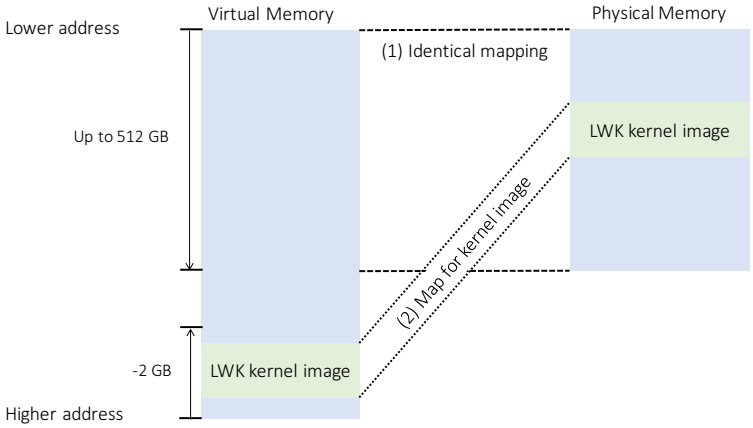

Figure 2.2: **Memory map when the LWK core enters LWK main routine.**

the LWK main routine. The virtual address range of `[ffff ffff 8000 0000, ffff ffff ffff ffff]` points the LWK kernel image in physical address space and the virtual address range of `[0000 0000 0000 0000, 0000 ff80 0000 0000]` defines an identical mapping to the same physical address range. LWK developers are recommended to create their own memory mapping based on this mapping.