FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

ジョブ運用ソフトウェア
APIユーザーズガイド Power API編

J2UL-2545-01Z0(02)
2022年3月

まえがき

本書の目的

本書では、Technical Computing Suiteのジョブ運用ソフトウェアが提供するPower APIの使用方法について説明します。

本書の読者

本書は、ジョブの運用・管理をする管理者と、Power APIを利用するエンドユーザが対象です。本書を読むためには、以下の知識が必要です。

- Linux に関する基本的な知識
- 「ジョブ運用ソフトウェア エンドユーザ向けガイド」による、ジョブの利用方法の知識
- Sandia Power APIの仕様に関する知識

本書の構成

本書は、次の構成になっています。

第1章 Power APIとは

Power APIの概要について説明します。

第2章 Power APIの利用方法

Power APIプログラムの作成方法、コンパイル方法、および実行方法について説明します。

第3章 FXサーバまたはPRIMERGYサーバで構成されるシステムにおける固有事項

本システムにおけるPower APIの固有事項について説明します。

付録A ジョブ運用ソフトウェアで利用可能な関数

ジョブ運用ソフトウェア上における、Power API関数のサポート範囲について説明します。

付録B サンプルプログラム

Power APIを利用したサンプルプログラムを記載しています。

本書の表記について

単位の表現

本書では、単位を表現する際の接頭語は以下のとおりです。基本的にディスクサイズは 10 のべき乗、メモリサイズは 2 のべき乗で表現します。コマンドの表示や入力時の指定の際には注意してください。

接頭語	値	接頭語	値
K (kilo)	10^3	Ki (kibi)	2^{10}
M (mega)	10^6	Mi (mebi)	2^{20}
G (giga)	10^9	Gi (gibi)	2^{30}
T (tera)	10^{12}	Ti (tebi)	2^{40}
P (peta)	10^{15}	Pi (pebi)	2^{50}

機種名の表現

本書では 富士通製CPU A64FXを搭載した計算機を「FXサーバ」、FUJITSU server PRIMERGYを「PRIMERGYサーバ」(または単に「PRIMERGY」)と表記します。

本書で説明する機能の一部には、対象機種によって仕様に差があります。このような機能の説明では、以下のように対象機種を略称で表記します。

[FX]: FXサーバを対象にした機能です。
[PG]: PRIMERGYサーバを対象にした機能です。

コマンドのパス名の表記

操作例で、ディレクトリ /bin、/usr/bin、/sbin、または /usr/sbin 配下にあるコマンドについては絶対パスで示していない場合があります。

マニュアル内のアイコンについて

本書では、以下のアイコンを使用しています。

注意

特に注意が必要な事項を説明しています。必ずお読みください。

参照

詳細な情報が書かれている参照先を示しています。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

商標

- Linux® は米国およびそのほかの国における Linus Torvalds の登録商標です。
- そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。

出版年月および版数

版数	マニュアルコード
2022年3月 第1.2版	J2UL-2545-01Z0(02)
2020年6月 第1.1版	J2UL-2545-01Z0(01)
2020年2月 初版	J2UL-2545-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
FXサーバの電力量に関する説明を追加しました。	3.2.2	第1.2版
Power APIの電力計測・制御の対象を修正しました。	第1章	第1.1版
パワーノブの上限および下限値の問い合わせ関数を修正しました。	2.1.4, 3.2.3	
クロスコンパイラーのコマンド名を修正しました。	2.2, B.1	
ノードにより存在しないObjectを取得した場合の動作を明記しました。	3.2.1	
Attributeごとに異なる設定値・取得値のデータ型を明記しました。	3.2.2, 3.2.3, 3.3.2	
サンプルプログラムを修正しました。	B.4	

目　次

第1章 Power APIとは

Sandia Power API(Power Application Programming Interface)とは、Sandia National Laboratoriesが提唱した電力計測・制御をするためのライブラリインターフェースを規定したものです。
Sandia Power APIは、HPC(High Performance Computing)システムで必要とされる計測・制御機能を備えています。

ジョブ運用管理ソフトウェアにおけるPower APIは、FXサーバまたはPRIMERGYサーバの計算ノード上で利用できます。エンドユーザは、Power APIを利用するプログラムをジョブとして実行することで、特別な権限は必要とせずに以下に示す操作ができます。

表1.1 Power APIで実施可能な操作

機種	計算ノード内の電力計測	計算ノード内の電力制御
FXサーバ	○	○
PRIMERGYサーバ	○	×

○:実施可能、×:実施不可

Power APIはC言語で書かれたアプリケーションプログラムから利用できます。また、ジョブ運用ソフトウェアでは、FortranからもPower APIを利用でき、そのためのサンプルプログラムをパッケージに同梱しています。以降、Power APIを利用するプログラムのことを、Power APIプログラムと呼びます。

本システムで利用できるPower APIはSandia National Laboratoriesで公開している仕様書の2.0版をベースにしたもので、以下ができます。

- エンドユーザは、"付録A ジョブ運用ソフトウェアで利用可能な関数"に示した関数を利用できます。
- 自プロセスが割り当たっている計算ノードに対して、電力計測・制御ができます。

参照

- Sandia Power APIの詳細は、Sandia National LaboratoriesのWebページを参照してください。

  http://powerapi.sandia.gov/

- サンプルプログラムの詳細については、"付録B サンプルプログラム"を参照してください。

注意

ジョブ実行環境のKVMモードでは、Power APIは使用できません。KVMモードの詳細については、「ジョブ運用ソフトウェア エンドユーザ向けガイド」を参照してください。

第2章 Power APIの利用方法

ここでは、ジョブからPower APIを利用する方法について説明しています。

FXサーバまたはPRIMERGYサーバで構成されるシステムでは、以下のステップでPower APIを利用します。

1. Power APIを使用したプログラムを作成する。
2. Power APIプログラムをコンパイルする。
3. 作成したプログラムをジョブとして投入し、実行する。

以降では、各ステップの詳細について説明します。

2.1 Power APIプログラムの作成方法

ここでは、エンドユーザが単一のハードウェアに対して電力計測・制御ができるようなPower APIプログラムの作成方法について説明します。

参照

本システム固有の事項については、"第3章 FXサーバまたはPRIMERGYサーバで構成されるシステムにおける固有事項 "を参照してください。

Power APIの仕様の詳細は、Sandia National LaboratoriesのWebページを参照してください。

http://powerapi.sandia.gov/

2.1.1 Power APIプログラムの流れ

Power APIプログラムの基本的なステップは、以下のようになります。

1. 初期化をする。
2. 電力計測・制御の対象となるObjectを取得する。

   Power APIでは、CPUやメモリのようなハードウェアのことをObjectと呼びます。Objectの取得の詳細は"2.1.3 Objectの取得"を参照してください。
3. 対象となるプログラム区間に対して、電力計測・制御をする。
4. Power APIでは、Objectで実施可能な電力計測・制御の種類をAttributeで表現します。エンドユーザは、ObjectとAttributeを指定することで、対象となるハードウェアに対し電力計測や制御ができます。電力計測や制御の詳細は"2.1.4 電力計測・制御"を参照してください。
5. 終了処理をする。

電力の計測プログラムを上のステップに従って記述すると、以下のようになります。

```
#include <stdio.h>
#include "pwr.h"


int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    double energy0 = 0.0;
    double energy1 = 0.0;
    PWR_Time ts0 = 0;
    PWR_Time ts1 = 0;
    double ave_power = 0.0;

    // 1. Power API の初期化
    PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
```

```
    // 2. Objectの取得(ここでは計算ノード全体を示すObjectを取得)
    PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);

    // 3. 計測の対象とするプログラム区間の始点、終点で、電力量を取得し、
    //    それらからプログラム区間の平均電力を求める
    //    (ここではAttributeとして、PWR_ATTR_ENERGYを指定)
    PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy0, &ts0);

    ・・・・・・// 任意のプログラム区間

    PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy1, &ts1);

    ・・・・・・// 得られた2つの電力量energy0とenergy1から平均電力を求める。

    // 4. Power APIの終了処理
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

以降の節では、各ステップについて説明します。

2.1.2 初期化

エンドユーザは、初期化関数PWR_CntxtInit()を呼び出すことで、Power APIを初期化します。

初期化関数の引数には以下を指定します。

- PWR_CntxtType *type*

  利用するPower APIの機能を指定するための識別子です。

- PWR_Role *role*

  Power API利用者の権限を示します。指定する権限によって、PWR_CntxtTypeで指定された機能の中で利用できる範囲が異なります。

参照

本システムで指定可能なPWR_CntxtType、およびPWR_Roleについては、"3.1 初期化関数について"を参照してください。

2.1.3 Objectの取得

Sandia Power APIでは、電力計測・制御の対象となるハードウェアのことをObjectと呼びます。Power APIが対象とするシステムは、Objectのツリー構造により表現されます。エンドユーザは、PWR_CntxtGetObjByName関数を利用することで、ユニーク名からObjectを取得できます。

表2.1 ユニーク名からObjectを取得する関数

関数	説明
PWR_CntxtGetObjByName	指定したユニーク名のObjectを取得する

参照

本システムのObject、およびObjectツリーについては、"3.2.1 FXサーバのObjectツリー構造の表現"および"3.3.1 PRIMERGYサーバのObjectツリー構造の表現"を参照してください。

2.1.4 電力計測・制御

エンドユーザは、ObjectとAttributeを指定することで、電力計測・制御ができます。

電力の計測・制御には、以下の関数を利用します。

表2.2 Attributeの取得・設定関数

関数	説明
PWR_ObjAttrGetValue	Objectに対して、単一のAttributeの値を取得します。
PWR_ObjAttrSetValue	Objectに対して、単一のAttributeの値を設定します。

電力の計測をしたい場合は、PWR_ATTR_ENERGYをPWR_ObjAttrGetValue関数に指定します。電力制御として周波数を変更したい場合は、PWR_ATTR_FREQをPWR_ObjAttrSetValue関数に指定します。また、現在設定されている周波数を取得したい場合は、PWR_ATTR_FREQをPWR_ObjAttrGetValue関数に指定します。

注意

FXサーバでは、消費電力を抑えるための機能であるBoB単位の電力キャッピング機能によって、CPU周波数の変更が制限される状態になることがあります(条件は管理者が設定します)。この状態では、Power APIでCPU周波数は変更できず、関数はエラーコードPWR_RET_FAILUREを返します。
ただし、設定可能な周波数の上限と下限(PWR_ObjAttrGetMeta関数で取得するAttributeのPWR_MD_MAXとPWR_MD_MIN)は、この制限が動作する前の値が得られます。PWR_ObjAttrGetMeta関数によるこれらの値の取得方法は、Sandia National LaboratoriesのWebページを参照してください。

参照

指定可能なAttributeは対象のObjectによって異なります。本システムにおけるObjectと指定可能なAttributeの対応関係については、計算ノードの機種に応じて"3.2 FXサーバにおける電力制御・計測"および"3.3 PRIMERGYサーバにおける電力制御・計測"を参照してください。

Power APIでは、複数のObjectおよびAttributeに対して一括して設定・取得を実施する関数もあります。詳細は、Sandia National LaboratoriesのWebページを参照してください。

2.1.5 終了処理

エンドユーザは、以下の関数を利用することで、Power APIを終了します。

表2.3 終了関数

関数	説明
PWR_CntxtDestroy	Power APIを終了します。

2.2 コンパイル方法

Power APIプログラムの実行ファイルを作成するためには、ログインノード上でPower APIプログラムをコンパイルする必要があります。コンパイルの際にはPower APIのヘッダーファイルを使用し、Power APIライブラリとリンクする必要があります。

Power APIのヘッダーファイルはpwr.h、ライブラリはlibpwr.soです。
Power APIヘッダーファイルとPower APIライブラリの配置場所は以下のとおりです。Power APIを実行する計算ノードの機種によって、ヘッダーファイル、ライブラリの配置場所が異なります。

表2.4 ヘッダーファイル、ライブラリの配置場所

ファイルの種類	配置場所
FXサーバ向けPower APIヘッダーファイル	/opt/FJSVtcs/pwrm/aarch64/include/
PRIMERGYサーバ向けPower APIヘッダーファイル	/opt/FJSVtcs/pwrm/x86_64/include/
FXサーバ向けPower APIライブラリ	/opt/FJSVtcs/pwrm/aarch64/lib64/
PRIMERGYサーバ向けPower APIライブラリ	/opt/FJSVtcs/pwrm/x86_64/lib64/

以下に、クロスコンパイラー(コマンド名をfccpxとします)を使用して、FXサーバ向けにPower APIプログラム sample.cをコンパイルする例を示します。

```
$ fccpx sample.c -L /opt/FJSVtcs/pwrm/aarch64/lib64 -I /opt/FJSVtcs/pwrm/aarch64/include -lpwr
```

2.3 実行方法

エンドユーザは、Power APIプログラムを実行するジョブスクリプトを記述し、ジョブを投入します。

例えば、Power APIプログラムa.outを実行するジョブスクリプトjob.shは、以下のように記述します。

```
#!/bin/bash
#PJM -L node=1

./a.out
```

ジョブは以下のように投入します。

```
$ pjsub job.sh
```

参照

ジョブ運用ソフトウェアのジョブスクリプトの記述方法、およびジョブの投入方法の詳細については、「ジョブ運用ソフトウェア エンドユーザ向けガイド」を参照してください。

第3章 FXサーバまたはPRIMERGYサーバで構成されるシステムにおける固有事項

ここでは、FXサーバまたはPRIMERGYサーバで構成されるシステムにおけるPower APIの固有事項について説明します。

3.1 初期化関数について

ここでは、Power APIプログラムの初期化をするPWR_CntxtInit関数の引数に指定可能なPWR_CntxtType、およびPWR_Roleについて説明します。

引数PWR_CntxtType *type*には、以下が指定できます。

表3.1 指定可能なPWR_CntxtType

PWR_CntxtType	説明
PWR_CNTXT_DEFAULT	Power API標準機能が利用可能
PWR_CNTXT_FX1000 [FX]	FXサーバ向けに拡張された機能が利用可能

本システムにおけるPower APIでは、Power APIのAttributeを拡張し、FXサーバ固有の電力計測・制御に対応したAttributeを独自定義しています。エンドユーザは、初期化時にPWR_CNTXT_FX1000を指定することで、FXサーバ固有のAttributeが利用可能になります。

引数PWR_Role *role*には、以下が指定できます。

表3.2 指定可能なPWR_Role

PWR_Role	説明
PWR_ROLE_APP	アプリケーション利用者を意味する

3.2 FXサーバにおける電力制御・計測

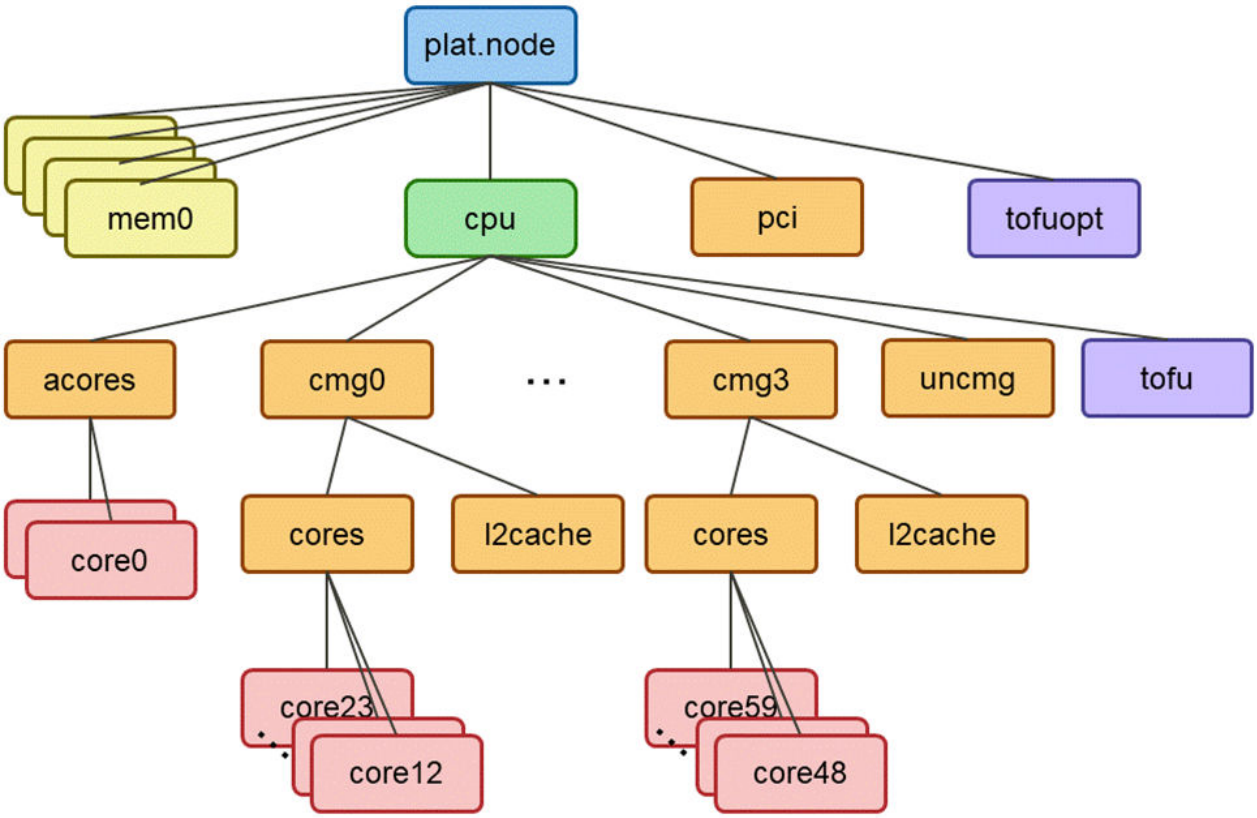
ここでは、FXサーバのObjectのツリー構造、FXサーバで実施可能な電力計測および電力制御について説明します。

3.2.1 FXサーバのObjectツリー構造の表現

本システムにおけるPower APIでは、自プロセスが動作する計算ノードのObjectのみ取得できます。それ以外の計算ノードのObjectは、取得できません。

自プロセスが動作する計算ノードは、以下の図のようなツリー構造により表現されます。各節点は、それぞれ、Objectを示しています。

図3.1 FXサーバのObjectツリー



上図で同色で示されているObjectは、同じPWR_ObjTypeであることを示しています。

各Objectの詳細は、以下のとおりです。

表3.3 FXサーバにおけるObject

ユニーク名	PWR_ObjType	説明
plat.node	PWR_OBJ_NODE	ノード全体
plat.node.cpu	PWR_OBJ_SOCKET	CPU Socket
plat.node.mem*N* (*N* = 0～3)	PWR_OBJ_MEM	メモリ
plat.node.pci	PWR_OBJ_POWER_PLANE	PCI express
plat.node.tofuopt	PWR_OBJ_NIC	光モジュール
plat.node.cpu.uncmg	PWR_OBJ_POWER_PLANE	CPU内のアシスタントコア群、CMG、Tofuを除いた部分
plat.node.cpu.acores (※) McKernelモードについては、注意を参照	PWR_OBJ_POWER_PLANE	アシスタントコア群
plat.node.cpu.cmg*N* (*N* = 0～3)	PWR_OBJ_POWER_PLANE	CMG
plat.node.cpu.tofu	PWR_OBJ_NIC	Tofu
plat.node.cpu.acores.core*L* (計算ノード: *L* = 0、1 計算ノード兼I/Oノード: *L* = 0～3) (※) McKernelモードについては、注意を参照	PWR_OBJ_CORE	アシスタントコア
plat.node.cpu.cmg*N*.cores	PWR_OBJ_POWER_PLANE	CMG内の計算コア群

ユニーク名	PWR_ObjType	説明
plat.node.cpu.cmg*N*.cores.l2cache	PWR_OBJ_POWER_PLANE	L2キャッシュ
plat.node.cpu.cmg*N*.cores.core*M* *N*に対し、*M*は以下の範囲をとります。 *N* = 0のとき、*M* = 12～23 *N* = 1のとき、*M* = 24～35 *N* = 2のとき、*M* = 36～47 *N* = 3のとき、*M* = 48～59 (※) McKernelモードについては、注意を参照	PWR_OBJ_CORE	計算コア

Power APIで利用するObjectのユニーク名は、ツリーのルート(plat.node)を起点として、ツリーを辿っていった節点の名前を"."で区切って並べていくことで表現することに注意してください。

例えば、図3.1 FXサーバのObjectツリーでcmg0の下にあるcoresのObjectをユニーク名で表現した場合は、plat.node.cmg0.coresとなります。

注意

- ユニーク名plat.node.cpu.acores.core*L*のL=2または3に該当するObjectは計算ノード兼I/Oノードだけ存在し、計算ノードには存在しません。計算ノードに対してPWR_CntxtGetObjByName 関数でこれらのObjectを指定した場合、エラーを返します。
- McKernelモードでは、アシスタントコアおよびアシスタントコア群のObjectは取得できません。
  McKernelモードでは、計算コアのObjectのユニーク名(plat.node.cpu.cmg*N*.cores.core*M*)の*N*、*M*は以下のとおりです。
  - *N* = 0のとき、*M* = 0～11
  - *N* = 1のとき、*M* = 12～23
  - *N* = 2のとき、*M* = 24～35
  - *N* = 3のとき、*M* = 36～47

  McKernelモードの詳細については、「ジョブ運用ソフトウェア エンドユーザ向けガイド」を参照してください。

3.2.2 FXサーバの電力計測

FXサーバでは、以下の2種類の電力量(単位はJ)を取得できます。

電力量	説明
推定電力量 (IDEAL)	CPUの命令発行数などに基づいて推定した電力量のことです。計算ノードの個体差による電力量のばらつきを考慮しない値であり、同じジョブであれば、推定電力量は同じになります。 ハードウェアにより算出され、1ミリ秒ごとに更新されます。
実測電力量 (MEASURED)	ジョブが実際に消費した電力量のことです。計算ノードの個体差による電力量のばらつきが含まれる値であり、同じジョブでも実行されるノードや処理するデータパターンの違いに伴い実測電力量は異なります。 ハードウェアによりノード内の電力計測素子から収集され、5ミリ秒ごとに更新されます。

これらの電力量に対応するPower APIのAttribute、そのデータ型、および計測可能なObjectは、以下の表のとおりです。

表3.4 電力量に対応するAttributeおよび計測可能なObject

電力量	Attribute(データ型)	計測可能なObject
推定電力量	PWR_ATTR_ENERGY (double)	ノード全体 CMG内の計算コア群 L2キャッシュ メモリ Tofu CPU内のアシスタントコア群、CMG、Tofuを除いた部分

電力量	Attribute(データ型)	計測可能なObject
		アシスタントコア 光モジュール PCI Express
実測電力量	PWR_ATTR_MEASURED_ENERGY (double)	ノード全体

注意

- FXサーバにおける「ノード全体」の推定電力量は、以下の計算式で求めています。

  「CMG内の計算コア群」+「L2キャッシュ」+「メモリ」+「Tofu」+「CPU内のアシスタントコア群、CMG、Tofuを除いた部分」

  「アシスタントコア」はジョブ以外のプロセスでも使用します。また、「光モジュール」および「PCI Express」の推定電力量はジョブが割り当たったノードに依存して変化します。このため、これら3つの推定電力量はノード全体の推定電力量には計上されません。
- 「ノード全体」の実測電力量は、計算ノードと計算ノード兼I/O ノードによって構成が異なる「アシスタントコア」、「光モジュール」、「PCI Express」も計上されます。
- 同一処理を異なるノードで実行する場合、「ノード全体」の推定電力量はノードによらず同じであるのに対し、実測電力量はノードによって差があります。「ノード全体」の2種類の電力量を比較すると±40%程度の差がある場合があります。
- PWR_ATTR_MEASURED_ENERGY は、FX サーバでの拡張Attribute です。初期化時に引数PWR_CntxtType *type*にPWR_CNTXT_FX1000を指定した場合にのみ、計測可能となります。

3.2.3 FXサーバの電力制御

FXサーバでは、以下のパワーノブ機能を使った電力制御ができます。

表3.5 FXサーバで利用可能なパワーノブ機能

パワーノブ機能	説明
周波数変更	CPUの周波数を制御します。
メモリアクセス制限	メモリアクセスコントローラーとメモリ間のバス使用率を制御します。
命令発行制限	コアが同時に処理する命令数を制御します。
EXA only	汎用レジスタを使用する命令が利用できるコアのパイプ数を制御します。
エコモード、FLA only状態	FLA onlyは、FPとSIMDレジスタが利用できるコアのパイプ数を制御します。エコモードは、FLA only有効時の電力削減効果を増やすための機能です。
Retention状態	コア上でプロセスが動作していない場合に、より低電力状態(Retention状態)に遷移するかどうかを設定します。

パワーノブ機能に対応するPower APIのAttribute、そのデータ型、利用可能なObject、および設定可能な値は以下のとおりです。

表3.6 パワーノブ機能に対応するAttributeと利用可能なObject、および設定可能な値

パワーノブ機能	Attribute(データ型)	利用可能なObject	設定可能な値
周波数変更	PWR_ATTR_FREQ (double)	CPU Socket	Hz単位で設定します。 2200000000 2000000000 1600000000 ハードウェアで設定可能な値は記載された値と違う場合があります。
メモリアクセス制限	PWR_ATTR_THROTTLING_STATE (uint64_t)	メモリ	0: バス使用率100% 1: バス使用率90% 2: バス使用率80%

パワーノブ機能	Attribute(データ型)	利用可能なObject	設定可能な値
			3: バス使用率70% 4: バス使用率60% 5: バス使用率50% 6: バス使用率40% 7: バス使用率30% 8: バス使用率20% 9: バス使用率10%
命令発行数制限	PWR_ATTR_ISSUE_STATE (uint64_t)	計算コア	0: 4命令 1: 2命令
EXA only	PWR_ATTR_EX_PIPE_STATE (uint64_t)	計算コア	0: パイプA、パイプBを利用 1: パイプAのみ利用
エコモード、 FLA only状態	PWR_ATTR_ECO_STATE (uint64_t)	計算コア	0: エコモード無効、FLA only無効 1: エコモード無効、FLA only有効 2: エコモード有効、FLA only有効
Retention状態	PWR_ATTR_RETENTION_STATE (uint64_t)	計算コア	0: Retention状態に遷移しない 1: Retention状態に遷移する

エンドユーザが電力制御で設定可能な範囲は、管理者の設定により制限されます。電力制御で設定可能な値の範囲を知るためには、以下の方法があります。

- PWR_ObjAttrGetMeta関数に取得対象のAttributeとMetadataを指定して問い合わせます。Attributeに設定可能な下限値を取得するためにはMetadataにPWR_MD_MINを、設定可能な上限値を取得するためにはPWR_MD_MAXを指定してください。値はAttributeに応じて"表3.6 パワーノブ機能に対応するAttributeと利用可能なObject、および設定可能な値"に示したデータ型として返却されます。

参照

PWR_ObjAttrGetMeta関数の仕様の詳細は、Sandia National LaboratoriesのWebページを参照してください。

http://powerapi.sandia.gov/

注意

PWR_ATTR_THROTTLING_STATE、PWR_ATTR_ISSUE_STATE、PWR_ATTR_EX_PIPE_STATE、PWR_ATTR_ECO_STATE、およびPWR_ATTR_RETENTION_STATEは、FXサーバでの拡張Attributeです。初期化時に引数PWR_CntxtType *type*にPWR_CNTXT_FX1000を指定した場合にのみ、制御可能となります。

3.3 PRIMERGYサーバにおける電力制御・計測

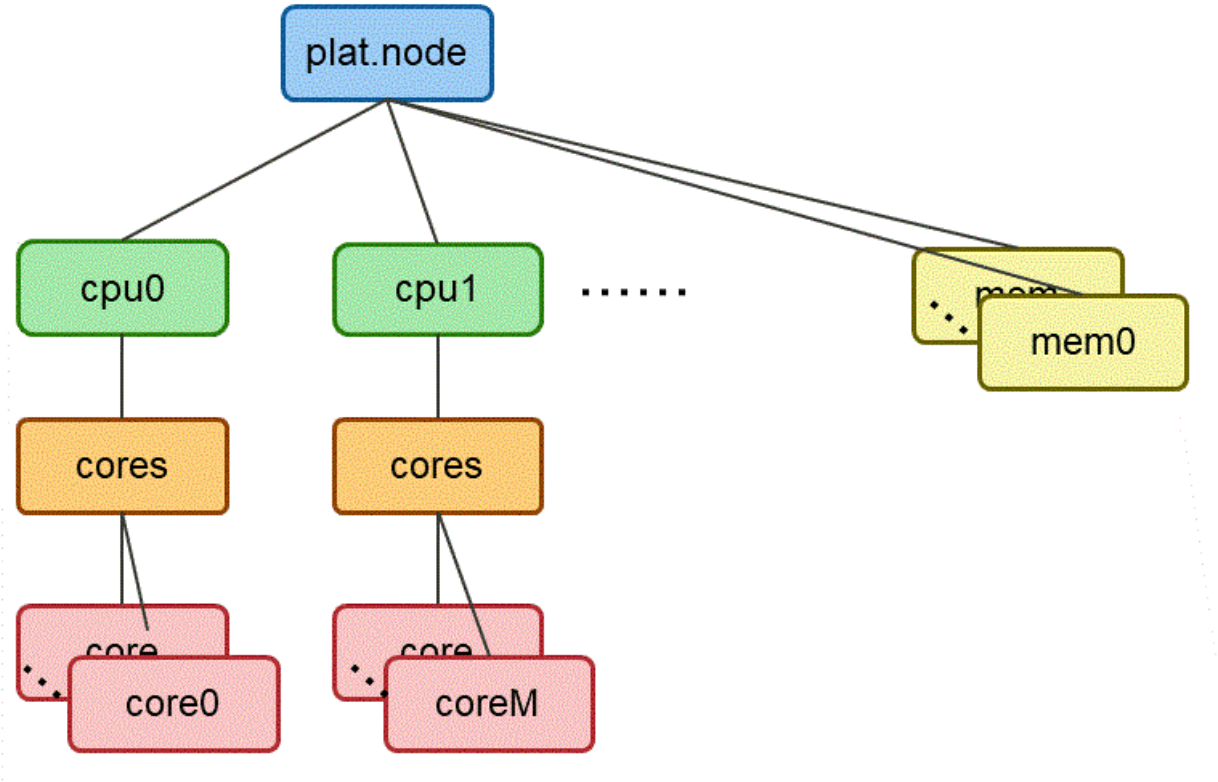
ここでは、PRIMERGYサーバのObjectのツリー構造、PRIMERGYサーバで実施可能な電力計測および電力制御について説明します。

3.3.1 PRIMERGYサーバのObjectツリー構造の表現

本システムにおけるPower APIでは、自プロセスが動作する計算ノードのObjectのみ取得できます。それ以外の計算ノードのObjectは、取得できません。

自プロセスが動作する計算ノードは、以下の図のようなツリー構造により表現されます。各節点は、それぞれObjectを示しています。

図3.2 PRIMERGYサーバのObjectツリー



上図で同色で示されているObjectは、同じPWR_ObjTypeであることを示しています。

各Objectの詳細は、以下のとおりです。

表3.7 PRIMERGYサーバにおけるObject

ユニーク名	PWR_ObjType	説明
plat.node	PWR_OBJ_NODE	ノード全体
plat.node.cpu*N* (*N*はハードウェアに依存)	PWR_OBJ_SOCKET	CPU Socket
plat.node.mem*N* (*N*はハードウェアに依存)	PWR_OBJ_MEM	メモリ
plat.node.cpu*N*.cores (*N*はハードウェアに依存)	PWR_OBJ_POWER_PLANE	CPUコア群
plat.node.cpu*N*.cores.core*M* (*N*、*M*はハードウェアに依存)	PWR_OBJ_CORE	CPUコア

Power APIで利用するObjectのユニーク名は、ツリーのルート(plat.node)を起点として、ツリーを辿っていった節点の名前を"."で区切って並べていくことで表現することに注意してください。

例えば、図3.2 PRIMERGYサーバのObjectツリーでcpu0の下にあるcoresのObjectをユニーク名で表現した場合は、plat.node.cpu0.coresとなります。

3.3.2 PRIMERGYサーバの電力計測

PRIMERGYサーバでは、以下の電力量(単位はJ)を取得できます。

電力量	説明
実測電力量	実測された電力量のことです。実測電力量を利用することで、ジョブが実際に消費した電力を得られます。

これらの電力量に対応するPower APIのAttribute、そのデータ型、および計測可能なObjectは、以下の表のとおりです。

表3.8 電力量に対応するAttributeおよび計測可能なObject

電力量	Attribute(データ型)	計測可能なObject
実測電力量	PWR_ATTR_ENERGY (double)	CPU Socket メモリ CPUコア群

注意

電力量計測可能なObjectは、PRIMERGYサーバに搭載されているCPUによって異なります。電力量計測時にエンドユーザが電力量計測不可のObjectを指定した場合は、Power APIの関数はエラーを返します。詳細は、Sandia National LaboratoriesのWebページを参照してください。

3.3.3 PRIMERGYサーバの電力制御

PRIMERGYサーバでは、ジョブからの電力制御はできません。

3.4 統計情報

Sandia Power APIでは、統計情報を取得するための関数が規定されています。本システムでは、PWR_ATTR_ENERGYが利用可能なObjectについて、電力(PWR_ATTR_POWER)の統計情報を取得できます。

Power APIで取得可能な統計情報の種類と対応するAttributeは、以下の表のとおりです。電力量の変化から電力を定期に算出し、統計処理を行います。

表3.9 取得可能な統計情報の種類

項目	Attribute
電力	PWR_ATTR_POWER

参照

- PWR_ATTR_ENERGYが利用可能なObjectについては、計算ノードの機種に応じて、"3.2.2 FXサーバの電力計測" または "3.3.2 PRIMERGYサーバの電力計測" を参照してください。
- 統計情報および統計情報を取得する関数の詳細は、Sandia National LaboratoriesのWebページを参照してください。
- また、"B.4 統計情報取得のプログラム例" に統計情報の関数のサンプルプログラムが記載されていますので、そちらも参考にしてください。

付録A　ジョブ運用ソフトウェアで利用可能な関数

ジョブ運用ソフトウェアが提供するPower APIでは、エンドユーザは以下の関数を利用できます。

表A.1 エンドユーザが利用可能な関数一覧

関数名	利用可否	説明
PWR_CntxtInit	○	Power APIの初期化関数。
PWR_CntxtDestroy	○	Power APIの終了関数。
PWR_CntxtGetEntryPoint	○	Objectツリーのrootのobjectを取得する関数。
PWR_ObjGetType	○	指定したObjectのPWR_ObjTypeを取得する関数。
PWR_ObjGetName	○	指定したObjectのユニーク名を問い合わせる関数。
PWR_ObjGetParent	○	指定したObjectの親Objectを取得する関数。
PWR_ObjGetChildren	○	指定したObjectの子Object群を取得する関数。
PWR_CntxtGetObjByName	○	指定したユニーク名に対応するObjectを取得する関数。
PWR_GrpCreate	○	複数のObjectを格納するためのGroupを生成する関数。
PWR_GrpDestroy	○	指定したGroupを破棄する関数。
PWR_GrpAddObj	○	指定したGroupに指定したObjectを追加する関数。
PWR_GrpRemoveObj	○	指定したGroupから指定したObjectを削除する関数。
PWR_GrpGetNumObjs	○	指定したGroupの中に含まれるObjectの数を取得する関数。
PWR_GrpGetObjByIndx	○	指定したGroupから、指定した要素番号のObjectを取得する関数。
PWR_GrpDuplicate	○	指定したGroupを複製する関数。
PWR_GrpUnion	○	指定した2つのGroupの和集合から構成されるGroupを生成する関数。
PWR_GrpIntersection	○	指定した2つのGroupの積集合から構成されるGroupを生成する関数。
PWR_GrpDifference	○	指定した2つのGroupの差集合から構成されるGroupを生成する関数。
PWR_GrpSymDifference	○	指定した2つのGroupの対称差集合から構成されるGroupを生成する関数。
PWR_CntxtGetGrpByName	×	指定したユニーク名に対応するシステム定義のGroupを取得する関数。本システムではGroupが定義されていないため、本関数は利用できません。
PWR_ObjAttrGetValue	○	指定したObjectに対して、指定したAttributeの値を取得する関数。
PWR_ObjAttrSetValue	○[FX]	指定したObjectに対して、指定したAttributeの値を設定する関数。本関数は、FXサーバでだけ利用可能です。
PWR_StatusCreate	○	複数のObjectやAttributeに対して一括で取得・設定する際に、それぞれのエラー情報を保持するためのstatusを生成する関数。
PWR_StatusDestroy	○	指定したstatusを破棄する関数。
PWR_StatusPopError	○	指定したstatusに保持されているエラー情報を取得する関数。
PWR_StatusClear	○	指定したstatusに保持されているエラー情報をクリアする関数。
PWR_ObjAttrGetValues	○	指定したObjectに対して、指定した複数のAttributeの値を一括で取得する関数。
PWR_ObjAttrSetValues	○[FX]	指定したObjectに対して、指定した複数のAttributeの値を一括で取得する関数。本関数は、FXサーバでだけ利用可能です。
PWR_ObjAttrIsValid	○	指定したObjectに対して、指定したAttributeが利用可能かどうかを問い合わせる関数。
PWR_GrpAttrGetValue	○	指定したGroupに属する全Objectに対して、指定したAttributeの値を一括で取得する関数。
PWR_GrpAttrSetValue	○[FX]	指定したGroupに属する全Objectに対して、指定したAttributeの値を一括で設定する関数。本関数は、FXサーバでだけ利用可能です。

関数名	利用可否	説明
PWR_GrpAttrGetValues	○	指定したGroupに属する全Objectに対して、指定した複数のAttributeの値を一括で取得する関数。
PWR_GrpAttrSetValues	○[FX]	指定したGroupに属する全Objectに対して、指定した複数のAttributeの値を一括で設定する関数。本関数は、FXサーバでだけ利用可能です。
PWR_ObjAttrGetMeta	○	指定したObjectおよびAttributeに関する詳細説明情報を取得する関数。
PWR_ObjAttrSetMeta	×	指定したObjectおよびAttributeに関する詳細設定情報を設定する関数。本システムでは、エンドユーザによる詳細設定情報の設定を禁止しています。そのため、エンドユーザは本関数を利用できません。
PWR_MetaValueAtIndex	○	指定したObjectおよびAttributeに設定可能な値を問い合わせるための関数。
PWR_ObjGetStat	×	指定したObjectおよびAttributeの統計情報を、システムに蓄積されたデータから取得する関数。本システムのPower APIではサポートしていません。
PWR_GrpGetStats	×	指定したGroupに属する全Object、およびAttributeに対して、各Objectの統計情報をシステムに蓄積されたデータから取得する関数。本システムのPower APIではサポートしていません。
PWR_ObjCreateStat	○	指定したObjectおよびAttributeの統計情報をリアルタイムで採取するためのobjstatを生成する関数。
PWR_GrpCreateStat	○	指定したGroupに属する全Object、およびAttributeに対して、各Objectの統計情報をリアルタイムで採取するためのgrpstatを生成する関数。
PWR_StatStart	○	指定したobjstatまたはgrpstatの統計情報の収集を開始する関数。
PWR_StatStop	○	指定したobjstatまたはgrpstatの統計情報の収集を停止する関数。
PWR_StatClear	○	指定したobjstatまたはgrpstatの統計情報の収集データをクリアする関数。
PWR_StatGetValue	○	指定したobjstatから統計値を取得する関数。
PWR_StatGetValues	○	指定したgrpstatに含まれる各Objectの統計値を取得する関数。
PWR_StatGetReduce	○	指定したgrpstatに含まれる各Objectの統計値に対して、reduction演算をする関数。
PWR_GrpGetReduce	×	指定したGroupに属する全Object、およびAttributeに対して、各Objectの統計情報をシステムに蓄積されたデータから取得し、reduction演算をする関数。本システムのPower APIではサポートしていません。
PWR_StatDestroy	○	指定したstatを破棄する関数。
PWR_GetMajorVersion	○	Power APIのメジャーバージョンを取得する関数。
PWR_GetMinorVersion	○	Power APIのマイナーバージョンを取得する関数。
PWR_GetReportByID	×	指定したAttributeの統計情報を取得する関数。Sandia Power APIの仕様上、エンドユーザは本関数を利用できません。
PWR_StateTransitDelay	×	指定したObjectで、指定した2つの電源管理に関するstateの間の遷移にかかる時間を取得する関数。本システムのPower APIではサポートしていません。
PWR_AppHintCreate	×	あるプログラム区間に対するアプリケーションのチューニングヒント情報を作成する関数。本システムのPower APIではサポートしていません。
PWR_AppHintDestroy	×	チューニングヒント情報を破棄する関数。本システムのPower APIではサポートしていません。
PWR_AppHintStart	×	チューニングヒント情報の対象となるプログラム処理が開始したことをOSに通知する関数。本システムのPower APIではサポートしていません。
PWR_AppHintStop	×	チューニングヒント情報の対象となるプログラム処理が終了したことをOSに通知する関数。本システムのPower APIではサポートしていません。
PWR_AppHintProgress	×	指定したプログラム区間における計算の進捗率を示す関数。 本システムのPower APIではサポートしていません。

関数名	利用可否	説明
PWR_SetSleepStateLimit	×	OSに対して遷移を許可する最も深いスリープ状態を設定する関数。本システムのPower APIではサポートしていません。
PWR_WakeUpLatency	×	指定したスリープ状態からの復帰遷移時間を取得する関数。本システムのPower APIではサポートしていません。
PWR_RecommendSleepState	×	指定した復帰遷移時間の範囲内で、最も深いスリープ状態を取得する関数。本システムのPower APIではサポートしていません。
PWR_SetPerfState	×	指定したObjectに対し、Objectに定義されているパフォーマンスレベルを設定する関数。本システムのPower APIではサポートしていません。
PWR_GetPerfState	×	指定したObjectの現在のパフォーマンスレベルを取得する関数。本システムのPower APIではサポートしていません。
PWR_GetSleepState	×	指定したObjectの現在のスリープ状態を取得する関数。本システムのPower APIではサポートしていません。

○:利用可能、×:利用不可

利用不可の関数を実行した場合は、エラーが返されます。

参照

各関数の詳細は、Sandia National LaboratoriesのWebページを参照してください。

http://powerapi.sandia.gov/

付録B　サンプルプログラム

ここでは、Power APIを利用したサンプルプログラムについて説明しています。

B.1　パッケージに含まれるサンプルプログラム

Power APIを利用したサンプルプログラムは、C言語のものと、Fortran言語のものがあります。
それぞれ、ログインノードの以下のディレクトリに配置されます。

- C言語のサンプルプログラム

```
/usr/src/FJSVtcs/pwrm/powerapi/c/
```

- Fortran言語のサンプルプログラム

```
/usr/src/FJSVtcs/pwrm/powerapi/fortran/
```

それぞれのディレクトリには、以下に示すサンプルプログラムが配置されています。

表B.1 ディレクトリに含まれるサンプルプログラム

サンプルプログラム	ファイル名 (C言語)	ファイル名 (Fortran言語)
電力量計測	pwrget.c	pwrget.f03
電力制御	pwrset.c	pwrset.f03
統計情報取得	pwrstat.c	pwrstat.f03
複数のObjectの電力量計測	pwrget_multi.c	pwrget_multi.f03
複数のObjectの電力制御	pwrset_multi.c	pwrset_multi.f03
FortranにおけるPower APIの関数・変数・型定義 (ヘッダーファイルに該当)	-	pwrf.f03
		pwrtypesf.f03

エンドユーザは、ログインノード上でこれらのサンプルプログラムを自分のディレクトリにコピーした後で、必要に応じて改変やコンパイルをしてください。

C言語のサンプルプログラムでは、コンパイルの対象とするサンプルプログラムのファイル名を指定して、コンパイルしてください。コンパイルに必要なヘッダーファイルやライブラリについては、”2.2 コンパイル方法”を参照してください。Fortran言語のサンプルプログラムでは、コンパイル対象とするサンプルプログラムのファイル名とpwrf.f03を指定して、コンパイルしてください。ファイルの指定では、必ずpwrf.f03を先頭にしてください。

以下に、クロスコンパイラー(コマンド名をfrtpxとします)を使用して、電力量計測のサンプルプログラム(pwrget.f03)をコンパイルする際のコマンドライン例を示します。コンパイル時に必要なPower APIのライブラリの指定については、”2.2 コンパイル方法”を参照してください。

```
$ frtpx pwrf.f03 pwrget.f03 -L /opt/FJSVtcs/pwrm/aarch64/lib64 -lpwr
```

以降では、C言語のサンプルプログラムのうち、以下のものについて掲載しています。

- 電力量計測
- 電力制御
- 統計情報取得

B.2　電力量計測のプログラム例

以下のプログラムは、FXサーバのObjectのユニーク名を指定してObjectを取得し、電力量を取得するプログラムです。

```
#include <stdio.h>
#include <unistd.h>
#include "pwr.h"
```

```
int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    int rc;
    double energy1 = 0.0;
    double energy2 = 0.0;
    double ave_power = 0.0;
    PWR_Time ts1 = 0;
    PWR_Time ts2 = 0;

    // Power APIのコンテキスト取得
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed¥n");
        return 1;
    }

    // 電力量計測の対象となるObjectの取得
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed¥n");
        return 1;
    }

    // Objectの推定電力量を取得
    rc = PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy1, &ts1);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrGetValue Failed (rc = %d)¥n", rc);
        return 1;
    }

    sleep(3);

    rc = PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy2, &ts2);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrGetValue Failed (rc = %d)¥n", rc);
        return 1;
    }

    // 2つの計測点の電力量から、平均電力を算出
    ave_power = (energy2 - energy1) / ((ts2 - ts1) / 1000000000.0);
    printf("ave_power = %lf¥n", ave_power);

    // コンテキスト破棄
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

B.3 電力制御のプログラム例

以下のプログラムは、FXサーバのObjectのユニーク名を指定してObjectを取得し、周波数を設定するプログラムです。

```
#include <stdio.h>
#include "pwr.h"

int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    int rc;
```

```
    double freq = 0.0;

    // Power APIのコンテキスト取得
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed¥n");
        return 1;
    }

    // 周波数設定の対象となるObjectの取得
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node.cpu", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed¥n");
        return 1;
    }

    // 設定する周波数を指定
    freq = 2000000000.0;
    // Objectに周波数を設定
    rc = PWR_ObjAttrSetValue(obj, PWR_ATTR_FREQ, &freq);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrSetValue Failed (rc = %d)¥n", rc);
        return 1;
    }

    // コンテキスト破棄
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

B.4 統計情報取得のプログラム例

以下のプログラムは、FXサーバ上で、ある区間の電力の最小値を取得するプログラムです。

```
#include <stdio.h>
#include <unistd.h>
#include "pwr.h"

int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    PWR_Stat stat = NULL;
    int rc;
    double min_power = 0.0;
    PWR_TimePeriod period = { PWR_TIME_UNINIT, PWR_TIME_UNINIT, PWR_TIME_UNINIT };

    // Power APIのコンテキスト取得
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed¥n");
        return 1;
    }

    // 統計情報取得の対象となるObjectの取得
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed¥n");
        return 1;
    }
```

```
    // 電力の最小値を取得する統計Objectの取得
    rc = PWR_ObjCreateStat(obj, PWR_ATTR_POWER, PWR_ATTR_STAT_MIN, &stat);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjCreateStat Failed (rc = %d)¥n", rc);
        return 1;
    }

    // 統計取得開始
    rc = PWR_StatStart(stat);

    sleep(3);

    // 統計取得終了
    rc = PWR_StatStop(stat);

    // 統計量取得
    rc = PWR_StatGetValue(stat, &min_power, &period);
    if (rc != PWR_RET_SUCCESS) {
        printf("StatGetValue Failed (rc = %d)¥n", rc);
        return 1;
    }

    printf("minimum power : %lf¥n", min_power);
    printf("start : %lu¥n", period.start);
    printf("stop : %lu¥n", period.stop);
    printf("instant : %lu¥n", period.instant);

    // 統計Object破棄
    PWR_StatDestroy(stat);

    // コンテキスト破棄
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```