AIX バージョン 7.2

Coherent Accelerator Processor Interface (CAPI) プログラミング

AIX バージョン 7.2

Coherent Accelerator Processor Interface (CAPI) プログラミング

お願い

本書および本書で紹介する製品をご使用になる前に、 31 ページの『特記事項』に記載されている情報をお読みください。

本書は AIX バージョン 7.2 および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： AIX Version 7.2
Coherent Accelerator Processor
Interface (CAPI) programming

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

本書について

Coherent Accelerator Processor Interface (CAPI) を使用して、Field Programmable Gate Array (FPGA) ベースのアクセラレーターがアプリケーション (ユーザー・スペース) メモリーに直接アクセスできるようにすることができます。

強調表示

本書では、次の強調表示規則を使用しています。

太字	コマンド、サブルーチン、キーワード、ファイル、構造体、ディレクトリー、およびシステムによって名前が事前に定義されているその他の項目を表します。 さらに太字の強調表示は、ユーザーが選択するボタン、ラベル、およびアイコンなどのグラフィカル・オブジェクトも示します。
イタリック	ユーザーが入力する実際の名前または値のパラメーターを示します。
モノスペース	具体的なデータ値の例、表示される可能性があるテキストの例、プログラマーとして作成する可能性があるものに似たプログラム・コードの一部の例、システムからのメッセージ、またはユーザーが入力しなければならないテキストを示します。

AIX でのケース・センシティブ

AIX® オペレーティング・システムでは、すべてケース・センシティブとなっています。これは、英大文字と小文字を区別するという意味です。 例えば、**ls** コマンドを使用するとファイルをリストできます。 `LS` と入力すると、システムはそのコマンドが「`is not found`」と応答します。 同様に、**FILEA**、**FiLea**、および **filea** は、同じディレクトリーにある場合でも、3 つの異なるファイル名です。 予期しない処理が実行されないように、常に正しい大/小文字を使用するようにしてください。

ISO 9000

当製品の開発および製造には、ISO 9000 登録品質システムが使用されました。

CAPI プログラミング

Coherent Accelerator Processor Interface (CAPI) を使用して、Field Programmable Gate Array (FPGA) ベースのアクセラレーターがアプリケーション (ユーザー・スペース) メモリーに直接アクセスできるようにすることができます。

従来型の FPGA ベースのアクセラレーターは、アクセラレーターとアプリケーションとの間でデータを移動するために、PCI スタック内で直接メモリー・アクセス (DMA) 転送を行います。CAPI が提供する汎用フレームワークは CAPI ベースのアクセラレーターを備えており、DMA を必要とせずにアプリケーション・メモリーとの間で双方向にデータを転送することができます。

CAPI フラッシュ・アダプター

Coherent Accelerator Processor Interface (CAPI) は、外部デバイス、POWER8® コア、およびシステムのオープン・メモリー・アーキテクチャーの間に高帯域幅で低遅延のパスを提供します。CAPI アダプターは、PCI Express (PCIe) x16 スロットに取り付けられ、基礎となるトランスポート・メカニズムとして PCIe Gen3 アダプターを使用します。

CAPI に対応したデバイスは、POWER8 コアで稼働するプログラムを実行できるアプリケーション・プログラムでも、カスタム・アクセラレーション実装環境を提供できるアプリケーション・プログラムでも、それらのプログラムに取って代わることができます。CAPI フラッシュ・アダプターは入出力サブシステムの複雑さを解消し、アクセラレーターをアプリケーションの一部として操作できるようにします。その結果、アプリケーションはオペレーティング・システムのカーネルを使用しなくてもフラッシュ・アクセラレーターと直接対話することができるため、コード・パスが縮小されます。

CAPI フラッシュ・ブロック・ライブラリー

Coherent Accelerator Processor Interface (CAPI) フラッシュ・アダプターのブロック・ライブラリーは、読み取りおよび書き込み入出力要求に対して、カーネルをバイパスし、ブロック・レベルまたはセクター・レベルで CAPI フラッシュ・ディスクへのユーザー・スペース・インターフェースを提供します。CAPI フラッシュ・アダプターのブロック・ライブラリーは、アプリケーションが CAPI フラッシュ・アダプターの低レベルの詳細にアクセスせずに済むよう、アプリケーション用のインターフェースを作成します。

AIX オペレーティング・システムでは、CAPI フラッシュ・アダプターのブロック・ライブラリーは `libcflsh_block.a` です。Linux プラットフォームでは、このライブラリーは `libcflsh_block.so` です。

cblk_init API

目的

Coherent Accelerator Processor Interface (CAPI) フラッシュ・アダプターのブロック・ライブラリーを初期化します。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX

int rc = cblk_int(void *arg, int flags)
```

説明

cblk_init API は、CAPI フラッシュ・アダプターのブロック・ライブラリーを初期化します。CAPI フラッシュ・アダプターのブロック・ライブラリーにある他の API を使用するには、前もって **cblk_init** API を呼び出しておく必要があります。

パラメーター

arg
このパラメーターは、現在使用されていません。これは NULL に設定されます。

フラグ
初期化のためのフラグを指定します。デフォルト値は 0 です。

戻り値

0 API は正常に完了しました。

ゼロ以外の値
エラーが発生しました。

cblk_term API

目的

Coherent Accelerator Processor Interface (CAPI) フラッシュ・ブロック・ライブラリーがもはや使用されなくなったときに、そのライブラリーのリソースをクリーンアップします。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_term(void *arg, int flags)
```

説明

cblk_term API は、CAPI フラッシュ・アダプターが使用されない場合に、アダプターのブロック・ライブラリーを終端処理します。

パラメーター

arg
このパラメーターは、現在使用されていません (NULL に設定されます)。

フラグ
初期化のためのフラグを指定します。デフォルト値は 0 です。

戻り値

0 API は正常に完了しました。

ゼロ以外の値
エラーが発生しました。

cblk_open API

目的

入出力 (読み取りおよび書き込み) 操作を実行できる Coherent Accelerator Processor Interface (CAPI) フラッシュ・デバイス上の、チャンク と呼ばれる連続したブロックの集合を開きます。チャンクは、セク

ター 0 からセクター *n*-1 までへのアクセスを提供する LUN (論理装置番号) と見なすことができます。ここで、*n* はセクター単位のチャンクのサイズです。仮想 LUN が指定された場合、チャンクは物理 LUN 上のセクターのサブセットです。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX

chunk_id_t chunk_id = cblk_open(const char *path, int max_num_requests, int mode,
uint64_t ext_arg, int flags)
```

説明

cblk_open API は、CAPI フラッシュ LUN 上にチャンクを作成します。このチャンクは、入出力 (cblk_read または cblk_write) 要求に使用されます。返された chunk_id 値は特定のアダプターから呼び出しプロセスへの特定のパスに割り当てられます。CBLK_OPN_VIRT_LUN フラグがセットされている場合、チャンクによって使用される基礎となる物理セクターは、ブロック・レイヤーのユーザーに直接的には見えません。

cblk_open API 呼び出しが正常に完了すると、作成されたチャンク・インスタンスを表すチャンク ID が呼び出しプロセスに返され、以後の API 呼び出しに使用されます。

パラメーター

path
このパラメーターは、CAPIディスクの特殊ファイル名を識別します。例えば、/dev/hdisk1 (AIX) および /dev/sg0 (Linux) です。

max_num_requests
このパラメーターは、特定の時間に特定のチャンクについてアダプターのキューに入れることができる最大コマンド数を示します。この値が 0 の場合、ブロック・レイヤーはデフォルト・サイズを選択します。指定された値が大きすぎる場合、cblk_open 要求は ENOMEM エラー値で失敗します。

mode
このパラメーターは、アクセス・モード (O_RDONLY、O_WRONLY、または O_RDWR) を指定します。

ext_arg
このパラメーターは、現在使用されていません。

フラグ
このパラメーターは、以下のビット・フラグの集合です。

CBLK_OPN_VIRT_LUN
このフラグは、仮想 LUN が物理 LUN 上にプロビジョンされていることを示します。このフラグが指定されなかった場合は、完全な物理 LUN への直接アクセスが提供されます。このフラグは、一時ストレージの場合にのみ有効です。cblk_close API が呼び出されると、このチャンクのすべてのデータ・セクターは解放され、他の操作に使用されます。

CBLK_OPN_NO_INTRP_THREADS
このフラグは、cflash ブロック・ライブラリーが、CAPI アダプターからの入出力要求の非同期完了に関する情報を処理および抽出するために、バックグラウンド・スレッドを開始しないことを示します。このライブラリーを使用するプロセスは、cblk_aresult または cblk_listio のいずれかのライブラリーを使用して、入出力操作の完了についてポーリングを行う必要があります。

CBLK_OPN_SCRUB_DATA
このフラグは、CBLK_OPN_VIRT_LUN フラグが指定された場合にのみ有効です。このフラグは、仮想

LUN 上のデータを消去してからでないと、その LUN を他の操作に再利用できないことを示します。現在、このフラグは AIX オペレーティング・システムではサポートされません。

CBLK_OPN_MPIO_FO
このフラグは、AIX オペレーティング・システムの場合にのみ有効です。このフラグは、cflash ブロック・ライブラリーがマルチパス入出力 (MPIO) フェイルオーバーを使用することを示します。パス固有のエラーが検出されない限り、すべての入出力要求に 1 つのパスが使用されます。そのようなパス・エラーが発生した場合は、代替パス (使用可能な場合) が使用されます。CAPI フラッシュ・ディスク用のパスを識別するには、**lspath -l hdiskN** コマンドを実行します。このフラグは、CBLK_OPN_VIRT_LUN、CBLK_OPN_RESERVE、CBLK_OPN_FORCED_RESERVE のいずれかのフラグが指定されている場合は有効ではありません。

CBLK_OPN_RESERVE
このフラグは、AIX オペレーティング・システムの場合にのみ有効です。このフラグは、cflash ブロック・ライブラリーが、ディスク予約を設定するディスクに関連付けられている予約ポリシー属性を使用することを示します。このフラグを CBLK_OPN_MPIO_FO フラグと一緒に使用することはできません。

CBLK_OPN_FORCED_RESERVE
このフラグは、AIX オペレーティング・システムの場合にのみ有効です。このフラグの動作は、デバイスが初めて開かれたとき、解決されていないディスク予約を取り消すことを除けば、CBLK_OPEN_RESERVE フラグと同じです。このフラグを CBLK_OPN_MPIO_FO フラグと一緒に使用することはできません。

戻り値

NULL_CHUNK_ID
エラーが発生しました。

cblk_close API

目的

入出力 (読み取りおよび書き込み) 操作を実行できる Coherent Accelerator Processor Interface (CAPI) フラッシュ・メモリー・デバイス上の、チャンクと呼ばれる連続したブロックの集合を閉じます。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_close(chunk_id_t chunk_id, int flags))
```

説明

cblk_close API はチャンクに関連付けられているブロックを、他の操作に再利用するために解放します。この chunk_id 値を返した対応する cblk_open API で CBLK_OPN_SCRUB_DATA フラグがセットされていた場合、ブロックを他の操作に再利用できるようにするには、前もってデータ・ブロックのユーザー・データを消去して、削除しておく必要があります。

パラメーター

chunk_id
再利用のために閉じて解放するチャンクのハンドル。

フラグ
ビット・フラグの集合。

戻り値

0 API は正常に完了しました。

ゼロ以外の値
エラーが発生しました。

cblk_get_lun_size API

目的

特定のチャンクが関連付けられている物理 LUN (論理装置番号) のサイズ (ブロック数) を返します。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_get_lun_size(chunk_id_t chunk_id, size_t *size, int flags))
```

説明

cblk_get_lun_size API は、このチャンクに関連付けられている物理 LUN のブロック数を返します。cblk_get_lun_size サービスを使用するには、有効な chunk_id 値を受け取るために cblk_open API を完了している必要があります。

パラメーター

chunk_id
物理 LUN サイズを返す必要があるチャンクのハンドル。

size
特定のチャンクに関連付けられている物理 LUN の 4K ブロックの合計数を指定します。

フラグ
ビット・フラグの集合。

戻り値

0 API は正常に完了しました。

>0 エラーが発生しました。

cblk_get_size API

目的

仮想 LUN (論理装置番号) である特定のチャンク ID に割り当てられているサイズ (ブロック数) を返します。つまり、このチャンク ID を返した cblk_open 呼び出しに CBLK_OPN_VIRT_LUN フラグが指定されます。このサービスは、チャンクが cblk_open API を使用して開かれたときに、CBLK_OPN_VIRT_LUN フラグがセットされていなかった LUN には無効です。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_get_size(chunk_id_t chunk_id, size_t *size, int flags))
```

説明

cblk_get_size サービスは、特定のチャンクに割り振られたブロックの数を返します。cblk_get_size サービスを使用するには、有効な chunk_id 値を受け取るために cblk_open API を完了している必要があります。

パラメーター

chunk_id
LUN サイズを変更する必要があるチャンクのハンドル。

size
特定のチャンクに関連付けられている LUN の 4K ブロックの数を指定します。

フラグ
ビット・フラグの集合。

戻り値

0 API は正常に完了しました。

>0 エラーが発生しました。

cblk_set_size API

目的

仮想 LUN (論理装置番号) である特定のチャンク ID にサイズ (ブロック数) を割り当てます。つまり、このチャンク ID を返した cblk_open 呼び出しに CBLK_OPN_VIRT_LUN フラグが指定されます。このチャンク ID に既にブロックが割り当てられている場合は、それより大きいサイズまたは小さいサイズを指定することにより、サイズを増減することができます。このサービスは、チャンクが cblk_open API を使用して開かれたときに、CBLK_OPN_VIRT_LUN フラグがセットされていなかった LUN には無効です。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_set_size(chunk_id_t chunk_id, size_t size, int flags))
```

説明

仮想 LUN を使用する場合、cblk_set_size サービスは特定のチャンクにいくつかのブロックを割り振ります。このチャンクを cblk_read や cblk_write で呼び出す前に、cblk_set_size API を呼び出す必要があります。cblk_set_size サービスを使用するには、また、有効な chunk_id 値を受け取るには、cblk_open 呼び出しが完了している必要があります。

当初このチャンクにブロックが割り当てられており、cblk_set_size API が同じチャンクに新規のブロックを割り振った後にそれらのブロックが再利用されていなかった場合、しかも、**flags** パラメーター内で CBLK_SCRUB_DATA_FLG フラグがセットされている場合には、当初のブロックは、クリアされた後に他の cblk_set_size 操作で再利用可能になります。

cblk_set_size API の正常終了後、チャンクは 0 から 1 までの範囲で、読み取りまたは書き込みが可能な論理ブロック・アドレス (LBA) サイズを持つことができます。

パラメーター

chunk_id
LUN サイズを設定する必要があるチャンクのハンドル。

size
特定のチャンクに関連付けられている LUN の 4K ブロックの数を指定します。

フラグ
ビット・フラグの集合。

戻り値

0 API は正常に完了しました。

>0 エラーが発生しました。

cblk_get_stats API

目的

特定のチャンク ID の統計情報を返します。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
typedef struct chunk_stats_s {
uint64_t max_transfer_size;    /* Maximum transfer size in */
                               /* blocks of this chunk. */
uint64_t num_reads;            /* Total number of reads issued */
                               /* via cblk_read interface */
uint64_t num_writes;           /* Total number of writes issued */
                               /* via cblk_write interface */
uint64_t num_areads;           /* Total number of async reads */
                               /* issued via cblk_aread interface */
uint64_t num_awrites;          /* Total number of async writes */
                               /* issued via cblk_awrite interface*/
uint32_t num_act_reads;        /* Current number of reads active */
                               /* via cblk_read interface */
uint32_t num_act_writes;       /* Current number of writes active */
                               /* via cblk_write interface */
uint32_t num_act_areads;       /* Current number of async reads */
                               /* active via cblk_aread interface */
uint32_t num_act_awrites;      /* Current number of async writes */
                               /* active via cblk_awrite interface*/
uint32_t max_num_act_writes;   /* High water mark on the maximum */
                               /* number of writes active at once */
uint32_t max_num_act_reads;    /* High water mark on the maximum */
                               /* number of reads active at once */
uint32_t max_num_act_awrites;  /* High water mark on the maximum */
                               /* number of asyync writes active */
                               /* at once. */
uint32_t max_num_act_areads;   /* High water mark on the maximum */
                               /* number of asyync reads active */
                               /* at once. */
uint64_t num_blocks_read;      /* Total number of blocks read */
uint64_t num_blocks_written;   /* Total number of blocks written */
uint64_t num_errors;           /* Total number of all error */
                               /* responses seen */
uint64_t num_aresult_no_cmplt; /* Number of times cblk_aresult */
                               /* returned with no command */
                               /* completion */
uint64_t num_retries;          /* Total number of all commmand */
                               /* retries. */
uint64_t num_timeouts;         /* Total number of all commmand */
                               /* time-outs. */
uint64_t num_fail_timeouts;    /* Total number of all commmand */
                               /* time-outs that led to a command */
                               /* failure. */
uint64_t num_no_cmds_free;     /* Total number of times we didm't */
                               /* have free command available */
uint64_t num_no_cmd_room ;     /* Total number of times we didm't */
                               /* have room to issue a command to */
                               /* the AFU. */
uint64_t num_no_cmds_free_fail; /* Total number of times we didn't */
                               /* have free command available and */
                               /* failed a request because of this*/
uint64_t num_fc_errors;        /* Total number of all FC */
```

```
                              /* error responses seen */
uint64_t num_port0_linkdowns;  /* Total number of all link downs */
                              /* seen on port 0. */
uint64_t num_port1_linkdowns;  /* Total number of all link downs */
                              /* seen on port 1. */
uint64_t num_port0_no_logins;  /* Total number of all no logins */
                              /* seen on port 0. */
uint64_t num_port1_no_logins;  /* Total number of all no logins */
                              /* seen on port 1. */
uint64_t num_port0_fc_errors;  /* Total number of all general FC */
                              /* errors seen on port 0. */
uint64_t num_port1_fc_errors;  /* Total number of all general FC */
                              /* errors seen on port 1. */
uint64_t num_cc_errors;        /* Total number of all check */
                              /* condition responses seen */
uint64_t num_afu_errors;       /* Total number of all AFU error */
                              /* responses seen */
uint64_t num_capi_false_reads; /* Total number of all times */
                              /* poll indicated a read was ready */
                              /* but there was nothing to read. */
uint64_t num_capi_adap_resets; /* Total number of all adapter */
                              /* reset errors. */
uint64_t num_capi_afu_errors;  /* Total number of all */
                              /* CAPI error responses seen */
uint64_t num_capi_afu_intrpts; /* Total number of all */
                              /* CAPI AFU interrupts for command */
                              /* responses seen. */
uint64_t num_capi_unexp_afu_intrpts; /* Total number of all of */
                              /* unexpected AFU interrupts */
uint64_t num_active_threads;   /* Current number of threads */
                              /* running. */
uint64_t max_num_act_threads;  /* Maximum number of threads */
                              /* running simultaneously. */
uint64_t num_cache_hits;       /* Total number of cache hits */
                              /* seen on all reads */
} chunk_stats_t;
int rc = cblk_get_stats(chunk_id_t chunk_id, chunk_stats_t *stats, int flags))
```

説明

cblk_get_stats サービスは、特定のチャンク ID の統計情報を返します。

パラメーター

chunk_id
統計情報を判別する必要があるチャンクのハンドル。

stats
chunk_stats_t 構造体のアドレスを指定します。

フラグ
ビット・フラグの集合。

戻り値

0 API は正常に完了しました。

>0 エラーが発生しました。

cblk_read API

目的

指定された論理ブロック・アドレス (LBA) にあるチャンクから、指定されたバッファーに 4K ブロックを読み取ります。仮想 LUN (論理装置番号) を使用する場合、この LBA は LUN の LBA と同じものではありません。チャンクは常に LUN の LBA、0 から始まるわけではないからです。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_read(chunk_id_t chunk_id, void *buf, off_t lba, size_t nblocks, int flags));
```

説明

cblk_read サービスはチャンクからデータを読み取り、そのデータを、提供されたバッファーに入れます。この呼び出しは、読み取り操作が正常に完了するかエラーで完了するまで阻止されます。つまり、この呼び出しは読み取り操作が完了するまで戻りません。仮想 LUN の場合は、特定のチャンクに対する cblk_read、cblk_write、cblk_aread、cblk_awrite の各呼び出しの前に、cblk_set_size API を呼び出す必要があります。

パラメーター

chunk_id
読み取られるチャンクのハンドル。

buf
チャンクからのデータを読み込むバッファーを指定します。このパラメーター値は、16 バイト境界に位置合わせされている必要があります。

lba
チャンク内部の LBA (4K オフセット) を指定します。

nblocks
転送のサイズを 4K セクター単位で指定します。物理 LUN の場合、上限は 16 MB です。仮想 LUN の場合、上限は 4K です。

フラグ
ビット・フラグの集合。

戻り値

-1 エラーを示します。詳細について、エラー番号が設定されます。

0 データが読み取られなかったことを示します。

***n* > 0**
読み取り操作が正常に完了したことを示します。ここで、*n* は読み取られたブロックの数です。

cblk_write API

目的

指定されたバッファーからのデータを使用して、指定された論理ブロック・アドレス (LBA) にあるチャンクに 4K ブロックを書き込みます。仮想 LUN (論理装置番号) を使用する場合、この LBA は LUN の LBA と同じものではありません。チャンクは LBA 0 から始まらないからです。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
int rc = cblk_write(chunk_id_t chunk_id, void *buf, off_t lba, size_t nblocks, int flags));
```

説明

cblk_write API は、指定のバッファーから指定の LBA にあるチャンクへデータを書き込みます。cblk_write 呼び出しは、書き込み操作が正常に完了するかエラーで完了するまで阻止されます。つまり、この呼び出しは書き込み操作が完了するまで戻りを行いません。仮想 LUN の場合は、特定のチャンクに対して cblk_write API を呼び出す前に cblk_set_size API を呼び出す必要があります。

パラメーター

chunk_id
書き込まれるチャンクのハンドル。

buf
チャンクへ書き込むデータのバッファーを指定します。このパラメーター値は、16 バイト境界に位置合わせされている必要があります。

lba
チャンク内部の LBA (4K オフセット) を指定します。

nblocks
転送のサイズを 4K セクター単位で指定します。物理 LUN の場合、上限は 16 MB です。仮想 LUN の場合、上限は 4K です。

フラグ
ビット・フラグの集合。

戻り値

-1 エラーを示します。詳細について、エラー番号が設定されます。

0 データが書き込まれなかったことを示します。

***n* > 0**
書き込み操作が正常に完了したことを示します。ここで、*n* は書き込まれたブロックの数です。

cblk_aread API

目的

指定された論理ブロック・アドレス (LBA) にあるチャンクから、指定されたバッファーに 4K ブロックを読み取ります。仮想 LUN (論理装置番号) を使用する場合、この LBA は LUN の LBA と同じものではありません。チャンクは LBA 0 から始まらないからです。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX


typedef enum {
    CBLK_ARW_STATUS_PENDING = 0,     /* Command has not completed */
    CBLK_ARW_STATUS_SUCCESS = 1      /* Command completed successfully */
    CBLK_ARW_STATUS_INVALID = 2      /* Caller's request is invalid */
    CBLK_ARW_STATUS_FAIL = 3         /* Command completed with an error */
} cblk_status_type_t;

typedef struct cblk_arw_status_s {
```

```
    cblk_status_type_t status;       /* Status of the command */
                                     /* See errno field for additional */
                                     /* details about the failure */
    size_t blocks_transferred;       /* Number of block transferred by */
                                     /* this reqeuest. */
    int errno;                       /* Errno when status indicates */
                                     /* CBLK_ARW_STAT_FAIL */
} cblk_arw_status_t;

int rc = cblk_aread(chunk_id_t chunk_id, void *buf, off_t lba, size_t nblocks, int
*tag, cblk_arw_status_t *status, int flags));
```

説明

cblk_aread サービスはチャンクからデータを読み取り、そのデータを、提供されたバッファーに入れます。この呼び出しは、読み取り操作が完了するまで阻止されません。つまり、この呼び出しは要求が発行された直後に戻りを行うため、読み取り操作が完了していない場合があります。後続の cblk_aresult 呼び出しを起動して、完了についてのポーリングを行う必要があります。仮想 LUN の場合は、cblk_aread API を呼び出す前に cblk_set_size API を呼び出す必要があります。

パラメーター

chunk_id
読み取られるチャンクのハンドル。

buf
チャンクからのデータを読み込むバッファーを指定します。このパラメーター値は、16 バイト境界に位置合わせされている必要があります。

lba
チャンク内部の LBA (4K オフセット) を指定します。

nblocks
転送のサイズを 4K セクター単位で指定します。物理 LUN の場合、上限は 16 MB です。仮想 LUN の場合、上限は 4K です。

tag
発行された各コマンドを一意的に識別できるよう、返される ID を指定します。

status
呼び出しプロセスによって提供されたアドレスを指定します。このアドレスは、capiblockライブラリーによって cblk_aread API の完了時に更新されます。アプリケーションは、cblk_aresult サービスを使用する代わりに、**status** 引数についてのポーリング・プロセスを使用できます。

CAPI アダプターは、このフィールドを直接更新することはできません。status パラメーターを更新するには、ソフトウェア・スレッドが必要です。このフィールドは、この chunk_id 値を返した cblk_open API に対して CBLK_OPN_NO_INTRP_THREADS フラグが指定されていた場合は使用されません。

フラグ

以下のビット・フラグからなる集合です。

CBLK_ARW_WAIT_CMD_FLAGS
要求を発行するために free コマンドが使用可能になるまで、cblk_aread サービスを阻止します。その他の場合、このサービスは -1 の値を EWOULDBLOCK のエラー値と一緒に返すことがあります。(現在使用可能な free コマンドがない場合)。

CBLK_ARW_USER_TAG_FLAGS
呼び出しプロセスが、この要求にユーザー定義タグを指定しようとしていることを示します。その場合、呼び出し元は、このタグを cblk_aresult API と一緒に使用して、CBLK_ARESULT_USER_TAG フラグを設定する必要があります。

CBLK_ARW_USER_STATUS_FLAG
呼び出しプロセスが、コマンドの完了時に更新される **status** パラメーターを設定したことを示します。

戻り値

-1 エラーを示します。詳細について、エラー番号が設定されます。

0 この API が正常に完了したことを示します。

***n* > 0**
読み取り操作 (キャッシュからの場合もあります) が完了したことを示します。ここで、*n* は読み取られたブロックの数です。

cblk_awrite API

目的

指定されたバッファーからのデータを使用して、指定された論理ブロック・アドレス (LBA) にあるチャンクに 4K ブロックを書き込みます。仮想 LUN (論理装置番号) を使用する場合、この LBA は LUN の LBA と同じものではありません。チャンクは LBA 0 から始まらないからです。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX

typedef enum {
       CBLK_ARW_STAT_NOT_ISSUED = 0  /* Command has not been issued */
       CBLK_ARW_STAT_PENDING = 1     /* Command has not completed */
       CBLK_ARW_STAT_SUCCESS = 2     /* Command completed successfully */
       CBLK_ARW_STAT_FAIL = 3        /* Command completed with error */
} cblk_status_type_t;

typedef struct cblk_arw_status_s {
       cblk_status_type_t status;    /* Status of command */
                                     /* See errno field for additional */
                                     /* details about the failure */
       size_t blocks_transferred;    /* Number of block transferred by */
                                     /* this reqeuest. */
       int errno;                    /* Errno when status indicates */
                                     /* CBLK_ARW_STAT_FAIL */
} cblk_arw_status_t;

int rc = cblk_awrite(chunk_id_t chunk_id, void *buf, off_t lba, size_t nblocks, int
*tag, cblk_arw_status_t *status, int flags));
```

説明

cblk_awrite API は、指定のバッファーから指定の LBA にあるチャンクへデータを書き込みます。この呼び出しは、書き込み操作が完了するまで阻止されません。つまり、この呼び出しは要求が発行された直後に戻りを行うため、書き込み操作が完了していない場合があります。後続の cblk_aresult 呼び出しを起動して、完了についてのポーリングを行う必要があります。仮想 LUN の場合は、cblk_awrite API を呼び出す前に cblk_set_size API を呼び出す必要があります。

パラメーター

chunk_id
書き込まれるチャンクのハンドル。

buf
チャンクへ書き込むデータのバッファーを指定します。このパラメーター値は、16 バイト境界に位置合わせされている必要があります。

lba
チャンク内部の LBA (4K オフセット) を指定します。

nblocks
転送のサイズを 4K セクター単位で指定します。物理 LUN の場合、上限は 16 MB です。仮想 LUN の場合、上限は 4K です。

tag
発行された各コマンドを一意的に識別できるよう、返される ID を指定します。

status
呼び出しプロセスによって提供されたアドレスを指定します。capiblock ライブラリーは cblk_aread API の完了時にこのアドレスを更新します。アプリケーションで、cblk_aresult サービスを使用する代わりに、cblk_aread API を使用できます。

CAPI アダプターは、このフィールドを直接更新することはできません。ソフトウェア・スレッドで状況領域を更新する必要があります。このフィールドは、この chunk_id 値を返した cblk_open API に対して CBLK_OPN_NO_INTRP_THREADS フラグが指定されていた場合は使用されません。

フラグ

以下のビット・フラグからなる集合です。

CBLK_ARW_WAIT_CMD_FLAGS
cblk_aread サービスが、要求を発行するために free コマンドを待つのを阻止します。その他の場合、このサービスは -1 の値を EWOULDBLOCK のエラー値と一緒に返すことがあります。(現在使用可能な free コマンドがない場合)。

CBLK_ARW_USER_TAG_FLAGS
呼び出しプロセスが、この要求にユーザー定義タグを指定しようとしていることを示します。その場合、呼び出しプロセスは、このタグを cblk_aresult API と一緒に使用して、CBLK_ARESULT_USER_TAG フラグを設定する必要があります。

CBLK_ARW_USER_STATUS_FLAG
呼び出しプロセスが、コマンドの完了時に更新される **status** パラメーターを設定したことを示します。

戻り値

-1 エラーを示します。詳細について、エラー番号が設定されます。

0 この API が正常に発行されたことを示します。

***n* > 0**
読み取り操作が完了したことを示します。ここで、*n* は書き込まれたブロックの数です。

cblk_aresult API

目的

非同期要求の状況および完了情報を返します。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
rc = cblk_aresult(chunk_id_t chunk_id, int *tag, uint64_t *status, int flags);
```

説明

cblk_aresult API は、cblk_aread API や cblk_awrite API を使用して発行された保留中の要求の状況を返します。それらの保留中の要求が完了した場合、この API は完了情報を返します。

パラメーター

chunk_id
書き込まれるチャンクのハンドル。

tag
呼び出しプロセスに、要求の完了を待機中であるというタグを付けるためのポインター。
CBLK_ARESULT_NEXT_TAG フラグがセットされている場合、このフィールドは次の非同期要求完了のタグを返します。

status
状況を指すポインター。状況は、要求の完了時に返されます。

フラグ
以下のフラグを cblk_aresult API に指定します。

CBLK_ARESULT_BLOCKING
コマンド (アクティブなコマンドが存在する場合) が完了するまで cblk_aresult API を阻止するには、このフラグを指定します。CBLK_ARESULT_NEXT_TAG フラグが指定された場合、この呼び出しは非同期入出力要求が完了した後に戻りを行います。

CBLK_ARESULT_USER_TAG
このフラグは、ユーザー指定のタグを使用して発行された非同期要求の状況を検査するために使用します。

戻り値

-1 エラーを示します。詳細について、エラー番号が設定されます。

0 この API が正常に発行されたことを示します。

***n* > 0**
要求が完了したことを示します。ここで、*n* は読み取られたかまたは書き込まれたブロック数です。

cblk_clone_after_fork API

目的

子プロセスが同じ仮想 LUN (論理装置番号) に親プロセスとしてアクセスすることを指定します。このサービスは、Linux プラットフォームの場合にのみ有効です。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX
rc = cblk_clone_after_fork(chunk_id_t chunk_id, int mode, int flags);
```

説明

cblk_clone_after_fork サービスは、子プロセスが親プロセスからデータにアクセスすることを指定します。子プロセスは、この操作を **fork()** システム呼び出しの直後に、そのストレージにアクセスするための親のチャンク ID を使用して実行する必要があります。子プロセスは、この操作を実行しなかった場合、親のチャンク ID にアクセスできなくなります。このサービスは、物理 LUN に対しては無効です。

注**:** このサービスは、Linux プラットフォームの場合にのみ有効です。

パラメーター

chunk_id
親プロセスによって使用中であるチャンクのハンドル。この呼び出しが正常に戻りを完了した場合、このチャンク ID を子プロセスも使用できます。

mode
子プロセスのアクセス・モード (O_RDONLY、O_WRONLY、または O_RDWR) を指定します。

注**:** 子プロセスは、親プロセスより大きなアクセス権限を持つことはできません。下位のプロセスは、より小さい権限を持つことができます。

フラグ
このパラメーターは、呼び出しプロセスによって指定されるビット・フラグです。

戻り値

0 要求が正常に完了したことを示します。

-1 エラーを示します。詳細について、エラー番号が設定されます。

cblk_listio API

目的

単一の呼び出しで Coherent Accelerator Processor Interface (CAPI) フラッシュ・ディスクに対して複数の入出力要求を発行し、CAPI フラッシュ・ディスクからの複数の入出力要求の完了を待ちます。

構文

```
#include <capiblock.h> for Linux or <sys/capiblock.h> for AIX

typedef struct cblk_io {
      uchar version;                /* Version of the structure */
#define CBLK_IO_VERSION_0 "I"       /* Initial version 0 */
      int flags;                    /* Flags for request */
#define CBLK_IO_USER_TAG 0x0001     /* Caller is specifying a user defined */
                                    /* tag. */
#define CBLK_IO_USER_STATUS 0x0002  /* Caller is specifying a status location */
                                    /* to be updated */
#define CBLK_IO_PRIORITY_REQ 0x0004 /* This is (high) priority request that */
                                    /* must be expediated vs non-priority */
                                    /* requests */
      uchar request_type;           /* Type of request */
#define CBLK_IO_TYPE_READ 0x01      /* Read data request */
#define CBLK_IO_TYPE_WRITE 0x02     /* Write data request */
      void *buf;                    /* Data buffer for the request */
      offset_t lba;                 /* Starting Logical block address for */
                                    /* the request. */
      size_t nblocks;               /* Size of request based on number of */
                                    /* blocks. */
      int tag;                      /* Tag for the request. */
```

```
      cblk_arw_status_t stat;        /* Status of the request */
} cblk_io_t

int rc = cblk_listio(chunk_id_t chunk_id,cblk_io_t *issue_io_list[],int
issue_io_items,cblk_io_t *pending_io_list[],int pending_io_items, cblk_io_t
*wait_io_list[], int wait_items, cblk_io_t *completion_io_list[],int
*completion_items, uint64_t timeout, int flags));
```

説明

cblk_listio サービスは、単一の呼び出しで複数の入出力要求を発行するためのインターフェースを提供し、単一の呼び出しを使用して、複数の入出力要求が完了したかどうかのポーリングを行います。個々の要求は cblk_io_t タイプによって指定され、データ・バッファー、開始論理ブロック・アドレス (LBA)、および 4K ブロック単位の転送サイズを含んでいます。

このサービスでは、cblk_io_t タイプに関連付けられている入出力要求を更新 (つまり、状況、タグ、およびフラグを入出力要求の処理に基づいて更新) することができます。

このサービスを使用して、cblk_aread API や cblk_awrite API を介して発行された入出力要求が完了したかどうかを確認することはできません。

パラメーター

chunk_id
入出力要求に関連付けられているチャンクのハンドル。

issue_io_list
このパラメーターは、CAPI フラッシュ・ディスクに対して発行する入出力要求の配列を指定します。タイプ cblk_io_t の個々の配列エレメントは、データ・バッファー、開始 LBA、および 4K ブロック単位の転送サイズを含んでいる個々の入出力要求を指定します。完了状況とタグを示すために、これらの配列要素をこの API によって更新できます。個々の cblk_io_t 配列エレメントの状況フィールドは、このAPI によって初期化されます。**issue_io_list** パラメーターがヌルの場合は、**pending_io_list** パラメーターを設定することにより、この API を使用して、前の cblk_listio 呼び出しによって発行された他の要求の完了を待つことができます。

issue_io_items
issue_io_list 配列の配列エレメントの数を指定します。

pending_io_list
前の cblk_listio 要求を介して発行された、入出力要求の配列を指定します。**pending_io_list** パラメーターを使用すると、すべての要求の完了を待つ (つまり、**completion_io_list** パラメーターを設定する) ことなく、入出力要求が完了したかどうかをポーリングすることができます。

pending_io_items
pending_io_list 配列内の配列エレメントの数を指定します。

wait_io_list
cblk_listio サービスが入出力要求の完了まで阻止されている、入出力要求の配列を指定します。これらの入出力要求は、**issue_io_list** パラメーターまたは **pending_io_list** パラメーターのいずれかでも指定されている必要があります。 **issue_io_list** 配列内の入出力要求が、呼び出しプロセスによる無効な設定のために、またはリソースがないために発行されなかった場合、io_list 内のその入出力要求のエレメントは、この失敗を示すために更新され (状況は CBLK_ARW_STAT_NOT_ISSUED として設定されます)、cblk_listio API はその入出力要求が完了するのを待ちません。このため、

wait_io_list 配列内の完了したすべての入出力要求は、CBLK_ARW_STAT_SUCCESS または CBLK_ARW_STAT_FAIL という状況になります。完了していない入出力要求の状況は、更新されません。

wait_items
wait_io_list 配列内の配列エレメントの数を指定します。

completion_io_list
このパラメーターは、呼び出しプロセスによって、初期化 (ゼロ化) された入出力要求配列に設定され、**completion_items** パラメーターは配列内の配列エレメントの数に設定されます。cblk_listio API が戻ったとき、配列には **issue_io_list** パラメーターおよび **pending_io_list** パラメーターで指定され、CAPI デバイスによって完了されたが **wait_io_list** パラメーターで指定されなかった入出力要求が入っています。io_list 配列内の入出力要求が、呼び出しプロセスによる無効な設定のために、またはリソースがないために発行されなかった場合、その入出力要求のエレメントは **completion_io_list** パラメーターにコピーされず、そのエレメントの io_list 配列内の状況は、この失敗を示すために更新されます (状況は CBLK_ARW_STAT_NOT_ISSUED として設定されます)。このため、このリスト内で返されるすべての入出力要求は、CBLK_ARW_STAT_SUCCESS または CBLK_ARW_STAT_FAIL という状況になります。

completion_items
このパラメーターは、呼び出し側プロセスによって、この API が **completion_io_list** パラメーター内に配置した配列エレメントの数のアドレスに設定されます。この API が戻ったとき、このパラメーターの値は、**completion_io_list** パラメーター内に配置された入出力要求の数に更新されます。

timeout
wait_io_list パラメーター内のすべての入出力要求を待つタイムアウト値を、マイクロ秒単位で指定します。このパラメーターは、**wait_io_list** パラメーターがヌルでない場合にのみ有効です。
wait_io_list パラメーター内のいずれかの入出力要求がこのタイムアウト値以内に完了しなかった場合、この API は -1 の値を返し、エラー番号の値を ETIMEDOUT に設定します (このエラーが発生した場合でも、一部のコマンドは **wait_io_list** パラメーター内で完了済みになっている可能性があります)。このため、呼び出しプロセスは、**wait_io_list** パラメーター内の各要求を検査して、どの要求が完了したかを判別する必要があります。呼び出しプロセスは、この API を次に呼び出す前に、完了した項目を **pending_io_list** パラメーターから削除する必要があります。0 のタイムアウト値は、**wait_io_list** パラメーター内の要求が完了するまで、この API が阻止されることを示します。

フラグ
以下のビット・フラグを指定します。

CBLK_LISTIO_WAIT_ISSUE_CMD
タイムアウト値を超過して CBLK_LISTIO_WAIT_CMD_FLAG フラグがセットされた場合でも、free コマンドですべての要求を発行できるようになるまで、cblk_listio API を阻止します。その他の場合、free コマンドが現在使用可能でないと、このサービスは -1 の値と EWOULDBLOCK のエラー値を返すことがあります (この状況の場合、一部のコマンドは発行されたリストのキューに正常に入っている可能性があります。呼び出しプロセスは、**issue_io_list** パラメーター内の個々の入出力要求を検査して、どの要求が失敗したかを判別する必要があります)。

戻り値

-1 詳細について、エラーおよびエラー番号が設定されました。

0 この API はエラーなしに正常に完了しました。

CAPI フラッシュ・キー/値ライブラリー

キー/値ライブラリーは、配列を保管、検索、および管理するための、Coherent Accelerator Processor Interface (CAPI) フラッシュ・デバイスへのインターフェースを提供します。キー/値ライブラリーは、キー/値のセマンティクスを CAPI フラッシュ・ブロック・ライブラリーにマップします。

AIX オペレーティング・システムでは、キー/値ライブラリーは `libarkdb.a` です。Linux プラットフォームでは、このライブラリーは `libarkdb.so` です。

ark_create API

目的

キー/値ストア・インスタンスを作成します。

構文

```
int ark_create(path, ark, flags)
char * file;
ARK ** handle;
uint64_t flags;
```

説明

ark_create API は、ホスト・システム上にキー/値ストア・インスタンスを作成します。

path パラメーターを使用して、フラッシュ・ストレージ上に作成された物理 LUN (論理装置番号) を表す特殊ファイル (例えば、Linux プラットフォーム用の `/dev/sdx` ファイルや AIX オペレーティング・システム用の `/dev/hdiskx` ファイル) を指定できます。**path** パラメーターが特殊ファイルでない場合、API は、そのファイルをキー/値ストアに使用する必要があると想定します。そのファイルが存在しない場合は、ファイルが作成されます。**path** パラメーターが NULL の場合、メモリーがキー/値ストアに使用されます。

flags パラメーターは、キー/値ストアのプロパティーを示します。物理 LUN の特殊ファイルを指定する場合は、物理 LUN 内の既存のキー/値ストアを使用するか、それとも仮想 LUN 内にキー/値ストアを作成するかを指定できます。デフォルトでは、物理 LUN 全体がキー/値ストアに使用されます。仮想 LUN が必要な場合は、 **flags** パラメーター内で `ARK_KV_VIRTUAL_LUN` ビット・フラグを設定する必要があります。

物理 LUN 全体を使用するように構成されたキー/値ストアは、永続させることができます。キー/値ストアの永続性を使用して (つまり、現行の状態をデータとして保存して) キー/値ストアをシャットダウンした後、同じ物理 LUN を開き、前のキー/値ストア・インスタンスを、閉じたときと同じ状態でロードすることができます。キー/値ストア・インスタンスを、シャットダウン (ark_delete) されたときも永続するように構成するには、**flags** パラメーター内で `ARK_KV_PERSIST_STORE` ビットをセットします。デフォルトでは、キー/値ストアは永続しないように構成されます。物理 LUN 上に保管されている永続キー/値ストア・インスタンスをロードするには、**flags** パラメーター内で `ARK_KV_PERSIST_LOAD` ビットをセットします。デフォルトでは、永続インスタンスは、存在する場合でもロードされず、新規の永続データ (ある場合) によって上書きされます。

物理 LUN 上に保管されたキー/値ストアのみを永続させることができます。

正常終了すると、**handle** パラメーターは新規に作成されたキー/値ストア・インスタンスを表し、以後の API 呼び出しにはそのインスタンスが使用されます。

パラメーター

path
キー/値ストア用の CAPI アダプター、ファイル、またはメモリーを指定します。

ark
キー/値ストアを表すハンドルを指定します。

フラグ
キー/値ストアのプロパティーを決定する、以下のビット・フラグからなる集合。

ARK_KV_VIRTUAL_LUN
特殊ファイルによって表される物理 LUN から作成された仮想 LUN をキー/値ストアに使用するよう指定します。

ARK_KV_PERSIST_STORE
キー/値ストア・インスタンスを、シャットダウン時にも永続するように構成します。ark_delete API を使用して、キー/値ストア・インスタンスのシャットダウンや削除を行うことができます。

ARK_KV_PERSIST_LOAD
物理 LUN 上に永続性データが存在する場合は、保管された構成をロードします。

戻り値

0 正常終了を示します。**handle** パラメーターは、新規に作成されたキー/値ストア・インスタンスを指します。

EINVAL
いずれかのパラメーターの値が無効です。

ENOSPC
メモリーまたはフラッシュ・ストレージが不足しています。

ENOTREADY
システムは、キー/値ストアの構成の準備が整っていません。

ark_delete API

目的

キー/値ストア・インスタンスを削除します。

構文

```
int ark_delete(ark)
ARK *ark;
```

説明

ark_delete API は、ホスト・システム上の **ark** パラメーターによって指定されたキー/値ストア・インスタンスを削除します。正常終了の場合、関連するすべてのメモリーおよびストレージ・リソースが解放されます。また、ARK インスタンスが永続するように構成されている場合は構成が永続するため、後でそのインスタンスをロードすることができます。

パラメーター

ark
キー/値ストア・インスタンスを表すハンドル。

戻り値

正常終了すると、ark_delete API はキー/ 値ストアに関連付けられているすべてのリソースをクリーンアップして削除し、0 を返します。正常に完了しなかった場合、ark_delete API は以下のいずれかの非ゼロ・エラー・コードを返します。

0 API は正常に完了しました。キー/値ストア・インスタンスに関連付けられているすべてのリソースは削除されます。

EINVAL
キー/値ストア・ハンドルが無効です。

ゼロ以外の値
エラーが発生し、API は正常に完了しませんでした。

ark_set、ark_set_async_cb API

目的

キー/値ペアを書き込みます。

構文

```
int ark_set(ark, klen, key, vlen, val, res)
int ark_set_async_cb(ark, klen, key, vlen, val, callback, dt)

ARK * ark;
uint64_t klen;
void * key;
uint64_t vlen;
void * val;
void *(*callback)(int errcode, uint64_t dt, uint64_t res);
uint64_t dt;
```

説明

ark_set API は、キーと値を、**ark** パラメーターによって表されるキー/値ストア・インスタンスのストアに保管します。ark_set_async_cb API は非同期モードで動作し、即時に呼び出しプロセスに戻り、操作の実行がスケジュールされます。操作が実行された後、呼び出しプロセスに操作の完了を通知するために、callback 関数が呼び出されます。

キー/値ストア・インスタンスでは、キーが存在する場合、保管されている値は val 値に置き換えられます。

正常終了の場合、キー/値ペアはストアに書き込まれ、キー/値ストアに書き込まれたバイト数が **res** パラメーターを介して呼び出しプロセスに返されます。

パラメーター

ark
キー/値ストア・インスタンスの接続を表すハンドルを示します。

key
キー/値ペアのキーを指定します。

klen
キーの長さをバイト単位で示します。

val
キー/値ペアの値を指定します。

vlen
値の長さをバイト単位で示します。

res
入出力操作の正常終了時にキー/値ストアに書き込まれるバイト数を示します。

callback
入出力操作の完了時に呼び出す関数を指定します。

dt 非同期 API 呼び出しにタグ付けする 64 ビット値を示します。

戻り値

ark_set API および ark_set_async_cb API は、正常終了の場合、キー/値ストア・インスタンスに関連付けられているストアにキー/値を書き込み、書き込まれたバイト数を返します。ark_set API の戻り値は、操作の状況を示します。ark_set_async_cb API の戻り値は、非同期操作が受け入れられたか、それとも拒否されたかを示します。callback 関数が実行されると、状況が **errcode** パラメーターに保管されます。ark_set API および ark_set_async_cb API は、正常に完了しなかった場合、以下のいずれかの非ゼロのエラー・コードを返します。

EINVAL
パラメーターが無効です。

ENOSPC
キー/値ストアに十分なスペースが残っていません。

ark_get、ark_get_async_cb API

目的

特定のキーの値を検索します。

構文

```
int ark_get(ark, klen, key, vbuflen, vbuf, voff, res)
int ark_get_async_cb(ark, klen, key, vbuflen, vbuf, voff, callback, dt)

ARK * ark;
uint64_t klen;
void * key;
uint64_t vbuflen;
void * vbuf;
uint64_t voff;
void *(*callback)(int errcode, uint64_t dt, uint64_t res);
uint64_t dt;
```

説明

ark_get API および ark_get_async_cb API は、**ark** パラメーターに関連付けられているキー/値ストアに特定の **key** パラメーターを照会します。キーが見つかった場合、キーの値が、最大 **vbuflen** バイトの **vbuf** パラメーターに返され、キーの値の **voff** オフセット・パラメーターの位置から始まるキー/値ストアに書き込まれます。ark_get_async_cb API は非同期モードで動作し、即時に呼び出しプロセスに戻り、検索操作の実行がスケジュールされます。操作が完了した後、呼び出しプロセスに操作の完了を通知するために、callback 関数が呼び出されます。

API が正常に完了した場合、キーの値の長さが callback 関数の **res** パラメーターに保管されます。

パラメーター

ark
キー/値ストア・インスタンスの接続を表すハンドルを示します。

key
キー/値ペアのキーを指定します。

klen
キーの長さをバイト単位で示します。

vbuf
キー/値ペアのキーの値を保管するバッファーを指定します。

vbuflen
vbuf バッファーの長さを指定します。

voff
読み取り操作を開始する、キー内のオフセット値を指定します。

res
ark_get API が正常に完了した場合、キーのサイズをバイト単位で保管します。

callback
入出力操作が完了したときに呼び出される callback 関数を指定します。

dt 非同期 API 呼び出しにタグ付けする 64 ビット値を指定します。

戻り値

ark_get API および ark_get_async_cb API は、正常終了の場合、0 を返します。ark_get API の戻り値は、操作の状況を示します。ark_get_async_cb API の戻り値は、非同期操作が受け入れられたか、それとも拒否されたかを示します。非同期 API の状況は、callback 関数の **errcode** パラメーターに保管されます。ark_get API および ark_get_async_cb API は、正常に完了しなかった場合、以下のいずれかの非ゼロのエラー・コードを返します。

EINVAL
パラメーターが無効です。

ENOENT
キーが見つかりませんでした。

ENOSPC
メモリー・バッファーにキーの値を保管する十分なスペースがありません。

ark_del、ark_del_async_cb API

目的

特定のキーに関連付けられている値を削除します。

構文

```
int ark_del(ark, klen, key, res)
int ark_del_async_cb(ark, klen, key, callback, dt)

ARK * ark
uint64_t klen;
void * key;
void *(*callback)(int errcode, uint64_t dt, uint64_t res);
uint64_t dt;
```

説明

ark_del API および ark_del_async_cb API は、**handle** パラメーターに関連付けられているキー/値ストアに特定の **key** パラメーターを照会します。そのキーが見つかった場合、ark_del API は、その値をキー/値ストアから削除します。ark_del_async_cb API は非同期モードで動作し、即時に呼び出しプロセスに戻り、削除操作の実行がスケジュールされます。操作が完了した後、呼び出しプロセスに操作の完了を通知するために、callback 関数が呼び出されます。

API が正常に完了した場合、キーの値の長さが callback 関数の **res** パラメーターで呼び出しプロセスに返されます。

パラメーター

ark
キー/値ストア・インスタンスの接続を表すハンドルを示します。

key
キー/値ペアのキーを指定します。

klen
キーの長さをバイト単位で示します。

res
この API が正常に完了した場合、キーのサイズをバイト単位で保管します。

callback
入出力操作が完了したときに呼び出される callback 関数を指定します。

dt 非同期 API 呼び出しにタグ付けする 64 ビット値を指定します。

戻り値

ark_del API および ark_del_async_cb API は、正常終了の場合、0 の値を返します。ark_del API の戻り値は、操作の状況を示します。ark_del_async_cb API の戻り値は、非同期操作が受け入れられたか、それとも拒否されたかを示します。非同期 API の状況は、callback 関数の **errcode** パラメーターに保管されます。ark_del API および ark_del_async_cb API は、正常に完了しなかった場合、以下のいずれかの非ゼロのエラー・コードを返します。

EINVAL
パラメーターが無効です。

ENOENT
キーが見つかりませんでした。

ark_exists、ark_exists_async_cb API

目的

特定のキーが存在するかどうかを調べるために、キー/値ストアに照会します。

構文

```
int ark_exist(ark, klen, key, res)
int ark_exist_async_cb(ark, klen, key, callback, dt)

ARK * ark
```

```
uint64_t klen;
void * key;
void *(*callback)(int errcode, uint64_t dt, uint64_t res);
uint64_t dt;
```

説明

ark_exists API および ark_exists_async_cb API は、**ark** パラメーターに関連付けられているキー/値ストアに特定の **key** パラメーターを照会します。そのキーが見つかった場合、ark_exist API は、値のサイズをバイト単位で **res** パラメーターに返します。キーとその値は変更されません。ark_exists_async_cb API は非同期モードで動作し、即時に呼び出しプロセスに戻り、照会操作の実行がスケジュールされます。操作が実行された後、呼び出しプロセスに操作の完了を通知するために、callback 関数が呼び出されます。

パラメーター

ark
キー/値ストア・インスタンスの接続を表すハンドルを示します。

key
キー/値ペアのキーを指定します。

klen
キーの長さをバイト単位で示します。

res
API が正常に完了した場合、キーのサイズをバイト単位で保管します。

callback
入出力操作の完了時に呼び出される callback 関数を指定します。

dt 非同期 API 呼び出しにタグ付けする 64 ビット値を指定します。

戻り値

ark_exists API および ark_exists_async_cb API は、正常終了の場合、0 の値を返します。ark_exists API の戻り値は、操作の状況を示します。ark_exists_async_cb API の戻り値は、非同期操作が受け入れられたか、それとも拒否されたかを示します。非同期 API の状況は、callback 関数の **errcode** パラメーターに保管されます。ark_exists API および ark_exists_async_cb API は、正常に完了しなかった場合、以下のいずれかの非ゼロのエラー・コードを返します。

EINVAL
パラメーターが無効です。

ENOENT
キーが見つかりませんでした。

ark_first API

目的

キー/値ストアで見つかった最初のキーを返し、キー/値ストアを反復検索するためのハンドルを返します。

構文

```
ARI*ark_first(ark, kbuflen, klen, kbuf)
ARK * ark
uint64_t buflen;
int64_t *klen;
void * kbuf;
```

説明

ark_first API は、キー/値ストアで見つかった最初のキーを **kbuf** バッファーに返し、そのキーのサイズを **klen** パラメーターに返します。ただし、キー・サイズ (**klen**) は **kbuf** サイズ (**kbuflen**) 未満です。

この API が正常に完了した場合、呼び出しプロセスにイテレーター・ハンドルが返され、ark_next API を呼び出してキー/値ストア内の次のキーを検索するには、このハンドルを使用する必要があります。

パラメーター

ark
キー/値ストア・インスタンスの接続を表すハンドルを示します。

kbuflen
kbuf パラメーターの長さを示します。

klen
kbuf パラメーターに返されるキーのサイズを指定します。

kbuf
キーを保持するバッファーを指定します。

戻り値

正常終了の場合、ark_first API は、ark_next API を使用した後続の呼び出しでキー/値ストアを反復検索するために使用する必要があるハンドルを返します。正常に完了しなかった場合、ark_first API は NULL を返し、エラー番号を以下のいずれかの値に設定します。

EINVAL
パラメーターが無効です。

ENOSPC
kbuf パラメーターに、キーを保管するための十分なスペースがありません。

ark_next API

目的

キー/値ストアで次に見つかったキーを返します。

構文

```
int ark_next(iter, kbuflen, klen, kbuf)
ARK * iter
uint64_t buflen;
int64_t *klen;
void *kbuf;
```

説明

ark_next API は、イテレーター・ハンドル **iter** に基づいてキー/値ストア内で次に見つかったキーを **kbuf** バッファーに返し、そのキーのサイズを **klen** パラメーターに返します。ただし、キー・サイズ (**klen**) は **kbuf** サイズ (**kbuflen**) 未満です。

正常に完了した場合、呼び出しプロセスにハンドルが返され、ark_next API を呼び出してキー/値ストア内の次のキーを検索するには、このハンドルを使用する必要があります。キー/値ストアの終わりに到達した場合は、ENOENT エラー・コードが返されます。

注**:** ストアの動的な性格上、書き込まれたキーの一部が返されない場合もあります。

パラメーター

iter
キー/値ストア内で検索を開始する場所のイテレーター・ハンドルを指定します。

kbuf
キーを保持するバッファーを指定します。

kbuflen
kbuf パラメーターの長さを示します。

klen
kbuf パラメーターに返されるキーのサイズを指定します。

戻り値

正常終了の場合、ark_next API は、ark_next API を使用した後続の呼び出しでキー/値ストアを反復検索するために使用する必要があるハンドルを返します。正常に完了しなかった場合、ark_next API は以下のいずれかの値を返します。

EINVAL
パラメーターが無効です。

ENOENT
ストアの終わりに到達しました。

ark_allocated API

目的

ストアに割り振られているバイト数を返します。

構文

```
int ark_allocated(ark, size)
ARK * ark;
uint64_t *size;
```

説明

ark_allocated API は、**size** パラメーターを介して、キー/値ストアに割り振られているバイト数を返します。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

size
キー/値ストアに割り振られているブロックのサイズをバイト単位で保持します。

戻り値

0 正常終了を示します。

EINVAL
無効なパラメーターによる失敗を示します。

ark_inuse API

目的

キー/値ストア内で使用中のバイト数を返します。

構文

```
int ark_inuse(ark, size)
ARK * ark;
uint64_t *size;
```

説明

ark_inuse API は、**size** パラメーターを介して、キー/値ストア内で使用中のバイト数を返します。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

size
使用中のストアのサイズをバイト単位で保持します。

戻り値

0 正常終了を示します。

EINVAL
無効なパラメーターによる失敗を示します。

ark_actual API

目的

キー/値ストア内で使用中のバイト数を返します。

構文

```
int ark_actual(ark, size)
ARK * ark;
uint64 * size;
```

説明

ark_actual API は、**size** パラメーターを介して、キー/値ストア内で使用中のバイト数を返します。この API が ark_inuse API と異なる点は、この API が個々のキーとそれらの値の実際のサイズを使用することです。それらの値を保管するための汎用ブロック割り振り数を使用するのではありません。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

size
使用中のブロックのサイズをバイト単位で保持します。

戻り値

0 正常終了を示します。**handle** パラメーターは、新規に作成されたキー/値ストア・インスタンスを指します。

EINVAL
無効なパラメーターによる失敗を示します。

ark_fork、ark_fork_done API

目的

アーカイブの目的でキー/値ストアを fork します。このサービスは、Linux プラットフォームの場合にのみ有効です。

構文

```
int ark_fork(ark)
int ark_fork_done(ark)
ARK * handle;
```

説明

ark_fork API および ark_fork_done API は、親のキー/値ストア・プロセスによって呼び出され、キー/値ストアを fork (複数のプロセスに分割) するための準備を行い、子プロセスを fork し、子プロセス終了後に呼び出し状態のクリーンアップを行います。ark_fork API は子プロセスを fork し、完了後に子プロセスのプロセス ID を親プロセスに返し、子プロセスに 0 を返します。親プロセスが子プロセスの終了を検出した後、ark_fork 呼び出しからのすべての状態をクリーンアップするために、ark_fork_done API が呼び出されます。

注**:** 未処理の非同期コマンドが存在する場合、ark_fork API は失敗します。ark_fork サービスは、Linux プラットフォームの場合にのみ有効です。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

戻り値

0 正常終了を示します。

EINVAL
無効なパラメーターによる失敗を示します。

EBUSY
未処理の非同期操作による失敗を示します。

ENOMEM
ストアを複製するスペースの不足による失敗を示します。

ark_random API

目的

キー/値ストアからランダム・キーを返します。

構文

```
int ark_random(ark, kbuflen, klen, kbuf)
ARK * ark;
uint64_t kbuflen
int64_t *klen;
void * kbuf;
```

説明

ark_random API は、**ark** ハンドルに基づいてキー/値ストアからランダム・キーを **kbuf** バッファーに返し、そのキーのサイズを **klen** パラメーターに返します。ただし、キー・サイズ (**klen**) は **kbuf** サイズ (**kbuflen**) 未満です。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

kbuflen
キー/値ストアのサイズをバイト単位で保持します。

klen
kbuf パラメーターに返されるキーのサイズを指定します。

kbuf
キーを保持するバッファーを指定します。

戻り値

0 正常終了を示します。

EINVAL
無効なパラメーターによる失敗を示します。

ark_count API

目的

キー/値ストア内で見つかったキーのカウント数を返します。

構文

```
int ark_count(ark, count)
ARK * ark;
int * count;
```

説明

ark_count API は、**ark** ハンドルに基づいて、キー/値ストア内にあるキーの合計数を返し、結果を **count** パラメーターに保管します。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

count
キー/値ストア内で見つかったキーの数を返します。

戻り値

0 正常終了を示します。

EINVAL
無効なパラメーターによる失敗を示します。

ark_stats API

目的

キー/値入出力操作およびブロック入出力操作の数を返します。

構文

```
#include <arkdb.h>

int ark_stats(ARK *ark, uint64_t *ops, uint64_t *ios);
```

説明

ark_stats API は、**ops** パラメーターを介してキー/値入出力操作の合計数を返し、**ios** パラメーターを介してブロック入出力操作の合計数を返します。

パラメーター

ark
キー/値ストアを表すハンドルを指定します。

ops
キー/値入出力操作の合計数を示します。

ios
ブロック入出力操作の合計数を示します。

戻り値

0 正常終了を示します。

EINVAL
エラーが検出されたことを示します。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年).

このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

© Copyright IBM Corp. _年を入れる_.

プライバシー・ポリシーに関する考慮事項

サービス・ソリューションとしてのソフトウェアも含めた IBM® ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらのCookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項を確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie などの各種テクノロジーの使用について詳しくは、『IBM オンラインでのプライバシー・ステートメントのハイライト』(http://www.ibm.com/privacy/jp/ja/)、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』というタイトルのセクション、および『IBM Software Products and Software-as-a-Service Privacy Statement』(http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。 現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml の「Copyright and trademark information」をご覧ください。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

A

C

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21