DEOS-FY2013-EA-01J

JST-CREST

研究領域

「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」

DEOS プロジェクト

D-RE API 仕様書

Version 1.02

2013/09/01

DEOS 研究開発センター

科学技術振興機構
Japan Science and Technology Agency

目次

変更履歴

Version	変更内容	変更者	日付
1.01	誤表記を修正	Dependable Embedded OS R&D Center	2013.Jul.15
1.02	誤表記、記載漏れを修正	Dependable Embedded OS R&D Center	2013.Sep.1

1 コンテナ API

1.1 共通

1.1.1 使用の流れ

コンテナの使用の流れは、生成(create)→開始(start)→停止(stop)→破棄(destroy)となり、一度 create したコンテナは、destroy するまで何度でも使用可能です。

また、本 API を使用したプログラム中でコンテナの操作を行う際の処理の流れは、コンテナのハンドル取得(open)→コンテナの操作(start/stop など)→コンテナのハンドル解放(close)となり、一度 open したハンドルは、close するまで何度でも使用可能です。

1.1.2 構造体

1.1.2.1 dre_err_t

概要

エラーコード

実装

```
typedef enum {
  DRE_ERR_INVALID_ARGUMENT          = -101,   // 引数が不正
  DRE_ERR_PERMISSION_DENIED         = -102,   // 実行権がない
  DRE_ERR_DAWARE_MOD_NONEXISTENCE   = -103,   // daware.mod が存在しない
  DRE_ERR_NO_VT                     = -104,   // Intel-VT が使用できない
  DRE_ERR_CONTAINER_NONEXISTENCE    = -111,   // コンテナが存在しない
  DRE_ERR_CONTAINER_EXISTENCE       = -112,   // コンテナが既に存在する
  DRE_ERR_CONTAINER_RUNNING         = -113,   // コンテナが動作中
  DRE_ERR_CONTAINER_STOPPED         = -114,   // コンテナが停止中
  DRE_ERR_CONTAINER_SUSPENDED       = -115,   // コンテナが一時停止中
  DRE_ERR_CONTAINER_BUSY            = -116,   // コンテナがビジー
  DRE_ERR_CONTAINER_UNKNOWN_STATUS  = -117,   // コンテナの状態が不明
  DRE_ERR_BASE_NONEXISTENCE         = -121,   // ベースイメージが存在しない
  DRE_ERR_BASE_EXISTENCE            = -122,   // ベースイメージが既に存在する
  DRE_ERR_REMOVE_BASE_FAILED        = -123,   // ベースイメージの削除に失敗
  DRE_ERR_SNAPSHOT_NONEXISTENCE     = -131,   // スナップショットが存在しない
  DRE_ERR_SNAPSHOT_EXISTENCE        = -132,   // スナップショットが既に存在する
  DRE_ERR_SAVE_SNAPSHOT_FAILED      = -133,   // スナップショットの保存に失敗
  DRE_ERR_LOAD_SNAPSHOT_FAILED      = -134,   // スナップショットの復元に失敗
  DRE_ERR_REMOVE_SNAPSHOT_FAILED    = -135,   // スナップショットの削除に失敗
  DRE_ERR_COMMIT_SNAPSHOT_FAILED    = -136,   // スナップショットのコミットに失敗
  DRE_ERR_CGROUP_NONEXISTENCE       = -141,   // cgroup が存在しない
  DRE_ERR_CGROUP_EXISTENCE          = -142,   // cgroup が既に存在する
  DRE_ERR_PID_NONEXISTENCE          = -143,   // 当該 PID のプロセスが存在しない
  DRE_ERR_SET_LIMIT_FAILED          = -144,   // リソースの制限に失敗
  DRE_ERR_UNSET_LIMIT_FAILED        = -145,   // リソース制限の解除に失敗
```

```
    DRE_ERR_DESTROY_CGROUP_FAILED   = -146,   // cgroup の破棄に失敗
    DRE_ERR_DEVICE_BUSY             = -151,   // デバイスが応答しない
    DRE_ERR_NO_TAP_AVAILABLE        = -152,   // 使用可能な TAP デバイスが存在しない
    DRE_ERR_NO_SOCKET_AVAILABLE     = -153,   // 使用可能なソケットが存在しない
    DRE_ERR_NFS_MOUNT_FAILED        = -154,   // NFS サーバのマウントに失敗
    DRE_ERR_BIND_FAILED             = -155,   // バインドに失敗
    DRE_ERR_INVALID_HOSTNAME        = -156,   // ホスト名が不正
    DRE_ERR_NULL_POINTER            = -201,   // null pointer
    DRE_ERR_REQUEST_TOO_LONG        = -202,   // リクエストの文字列が長すぎる
} dre_err_t;
```

1.1.3その他型宣言

1.1.3.1 dre_list_t

概要

コンテナのリスト

実装

```
typedef dre_name_t *    dre_list_t;
```

1.1.3.2 dre_name_t

概要

コンテナ名

実装

```
typedef char *          dre_name_t;
```

1.1.3.3 dre_return_t

概要

関数の戻り値

実装

```
typedef int             dre_return_t;
```

1.2 アプリケーションコンテナ

1.2.1使用方法

ソースコードにヘッダファイル“dre_app.h”を include し、コンパイル時に D-RE ライブラリ(libdreapp.so または libdreapp.a)と D-Aware ライブラリ(libdaware.so または libdaware.a)をリンクして下さい。

1.2.2定数・マクロ

1.2.2.1 dre_app_resource_t

概要

リソースの種類

実装

```
typedef enum {
    DRE_APP_CONTAINER_RESOURCE_CPU = 1,  // CPU
    DRE_APP_CONTAINER_RESOURCE_CPU_CORE,// CPU コア
    DRE_APP_CONTAINER_RESOURCE_MEMORY,  // メモリ
    DRE_APP_CONTAINER_RESOURCE_MEMSW    // メモリとスワップ領域
} dre_app_resource_t;
```

1.2.2.2 dre_app_status_t

概要

コンテナの状態

実装

```
typedef enum {
    DRE_APP_CONTAINER_STATUS_STOP = 11, // 停止中
    DRE_APP_CONTAINER_STATUS_RUNNING,    // 動作中
    DRE_APP_CONTAINER_STATUS_SUSPENDED, // 一時停止中
    DRE_APP_CONTAINER_STATUS_BUSY        // ビジー
} dre_app_status_t;
```

1.2.2.3 dre_app_type_t

概要

コンテナの種類

実装

```
typedef enum {
    DRE_APP_CONTAINER_TYPE_LXC = 1       // LXC
```

```
} dre_app_type_t;
```

1.2.3構造体

1.2.3.1 dre_app_data

概要

コンテナの設定

実装

```
typedef struct {
        dre_app_type_t type;   // コンテナの種類
        union {
                union {
                        int port;     // コンテナ上の SSH が使用するポート番号
                        char *base; // コンテナのベースイメージ名(NULL でデフォルトを使用)
                } lxc;
        } udata;
} dre_app_handle;
```

ベースイメージ名には、dre-app commit で作成したベースイメージが指定可能です。ベースイメージを指定すると、そのイメージ内にあるファイルシステムを使用したアプリケーションコンテナが作成できます。

1.2.3.2 dre_app_handle

概要

コンテナのハンドル

実装

```
typedef struct {
        dre_app_type_t type;   // コンテナの種類
        char *name;            // コンテナ名
} dre_app_handle;
```

1.2.3.3 dre_app_pid_t

概要

コンテナ上で動作するプロセスの PID

実装

```
typedef struct {
        int pid;               // PID(ホスト OS 上で付与される PID)
        int vpid;              //  仮想 PID(コンテナ上で付与される PID)
```

```
} dre_app_pid_t;
```

1.2.3.4 dre_app_resource_value_t

概要

リソース使用状況

実装

```
typedef struct {
        dre_app_resource_t type;            // リソースの種類
        union {
                union {
                        int parcent;        // CPU 使用率(%)
                        char *cores;        // CPU コア
                } cpu;
                union {
                        int usage_in_bytes;         // メモリ使用量(Bytes)
                        int memsw_usage_in_bytes;// メモリ+スワップ領域使用量(Bytes)
                } mem;
        } udata;
} dre_app_resource_value_t;
```

1.2.4型宣言

1.2.4.1 dre_app_pid_list_t

概要

コンテナ上で動作するプロセスの PID のリスト

実装

```
typedef dre_app_pid_t **  dre_app_pid_list_t;
```

1.2.5関数

1.2.5.1 dre_app_bind_pid

概要

コンテナ外のプロセスをコントロールグループにバインドする
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t  dre_app_bind_pid  (
        const dre_app_handle *          h,
```

```
        const char *            cgroup,
        const int               pid
);
```

引数

```
const dre_app_handle *      h       // ハンドル
const char *                cgroup  // コントロールグループ名
const int                   pid     // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.2 dre_app_bind_vpid

概要

コンテナ上のプロセスをコントロールグループにバインドする
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_bind_vpid (
        const dre_app_handle *      h,
        const char *                cgroup,
        const int                   vpid
);
```

引数

```
const dre_app_handle *      h       // ハンドル
const char *                cgroup  // コントロールグループ名
const int                   vpid    // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.3 dre_app_checkpoint

概要

コンテナのチェックポイントを作成する(未実装)

関数インタフェース

```
dre_return_t dre_app_checkpoint (
        const dre_app_handle *      h,
        const char *                tag
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// チェックポイント名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.4 dre_app_close

概要

コンテナのハンドルを解放する

関数インタフェース

```
dre_return_t dre_app_close (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.5 dre_app_commit_snapshot

概要

スナップショットを使用して、コンテナのベースとなるイメージを作成する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_commit_snapshot (
        const dre_app_handle *          h,
        const char *                    tag,
        const char *                    base
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// スナップショット名
const char *	base	// 作成するイメージ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.6 dre_app_create_cgroup

概要

コンテナ上にコントロールグループを作成する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_create_cgroup (
        const dre_app_handle *          h,
        const char *                    cgroup
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.7 dre_app_create_container

概要

コンテナを生成する

関数インタフェース

```
dre_return_t dre_app_create_container (
        const char *                    name,
        const dre_app_data *            data
);
```

引数

const char *	name	// コンテナ名
const dre_app_data *	data	// コンテナの設定

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.8 dre_app_destroy_cgroup

概要

コンテナ上のコントロールグループを破棄する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_destroy_cgroup (
        const dre_app_handle *          h,
        const char *                    cgroup
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.9 dre_app_destroy_container

概要

コンテナを破棄する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_destroy_container (
        const char *                    name
);
```

引数

const char *	name	// コンテナ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.10 dre_app_get_base_list

概要

ベースイメージのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_app_release_base_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_app_get_base_list (
        dre_list_t *                    list
);
```

引数

```
dre_list_t *                    list   // コンテナのリストへのポインタ
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.11 dre_app_get_checkpoint_list

概要

コンテナのチェックポイントのリストを取得する(未実装)

関数インタフェース

```
dre_return_t dre_app_get_checkpoint_list (
        const dre_app_handle *          h,
        dre_list_t *                    list
);
```

引数

```
const dre_app_handle *          h     // ハンドル
dre_list_t *                    list  // チェックポイントのリストへのポインタ
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.12 dre_app_get_container_list

概要

コンテナのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_sys_release_container_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_app_get_container_list (
        dre_list_t *                    list
);
```

引数

```
dre_list_t *                    list   // コンテナのリストへのポインタ
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.13 dre_app_get_pid

概要

コンテナのプロセス ID を取得する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_get_pid (
        const dre_app_handle *      h,
        int *                       pid
);
```

引数

const dre_app_handle *	h	// ハンドル
int *	pid	// プロセス ID

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.14 dre_app_get_registered_pid_list

概要

D-Aware モジュールの管理下にある PID のリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_app_release_registered_pid_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_app_get_registered_pid_list (
        const dre_app_handle *      h,
        dre_pid_list_t *            list
);
```

引数

const dre_app_handle *	h	// ハンドル
dre_app_pid_list_t *	list	// PID のリストへのポインタ

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.15 dre_app_get_resource

概要

リソース使用量を取得する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_get_resource (
        const dre_app_handle *          h,
        const int                       pid,
        dre_app_resource_value_t *      resource
);
```

引数

const dre_app_handle *	h	// ハンドル
const int	pid	// プロセスの PID
dre_app_resource_value_t *	resource	// リソース

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.16 dre_app_get_snapshot_list

概要

コンテナのスナップショットのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_app_release_snapshot_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_app_get_snapshot_list (
        const dre_app_handle *          h,
        dre_list_t *                    list
);
```

引数

const dre_app_handle *	h	// ハンドル
dre_list_t *	list	// スナップショットのリストへのポインタ

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.17 dre_app_load_snapshot

概要

コンテナをスナップショットの状態に復元する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_load_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
);
```

引数

```
const dre_app_handle *          h     // ハンドル
const char *                    tag   // スナップショット名
```

戻り値

```
0                               成功
0 以外                          エラー(1.1.2.1 参照)
```

1.2.5.18 dre_app_open

概要

コンテナのハンドルを取得する

関数インタフェース

```
dre_app_handle * dre_app_open (
        const char *                    name
);
```

引数

```
const char *                    name     // コンテナ名
```

戻り値

```
NULL                            取得失敗
NULL 以外                       取得したハンドルへのポインタ
```

1.2.5.19 dre_app_register_pid

概要

プロセスを D-Aware モジュールの管理下に置く

関数インタフェース

```
dre_return_t dre_app_register_pid (
        const dre_app_handle *          h,
        const int                       pid
);
```

引数

```
const dre_app_handle *          h     // ハンドル
const int                       pid   // 対象のプロセス ID
```

戻り値

```
0                               成功
0 以外                          エラー(1.1.2.1 参照)
```

1.2.5.20 dre_app_register_vpid

概要

プロセスを D-Aware モジュールの管理下に置く

関数インタフェース

```
dre_return_t dre_app_register_vpid (
        const dre_app_handle *          h,
        const int                       vpid
);
```

引数

```
const dre_app_handle *          h     // ハンドル
const int                       vpid  // 対象のプロセス ID
```

戻り値

```
0                               成功
0 以外                          エラー(1.1.2.1 参照)
```

1.2.5.21 dre_app_release_base_list

概要

ベースイメージのリストを解放する

関数インタフェース

```
dre_return_t dre_app_release_base_list (
        dre_list_t                      list
);
```

引数

```
dre_list_t                      list  // コンテナのリスト
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.22 dre_app_release_checkpoint_list

概要

コンテナのチェックポイントのリストを解放する(未実装)

関数インタフェース

```
dre_return_t dre_app_release_checkpoint_list (
        dre_list_t                      list
);
```

引数

dre_list_t	list // チェックポイントのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.23 dre_app_release_container_list

概要

コンテナのリストを解放する

関数インタフェース

```
dre_return_t dre_app_release_container_list (
        dre_list_t                      list
);
```

引数

dre_list_t	list // コンテナのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.24 dre_app_release_registered_pid_list

概要

D-Aware モジュールの管理下にある PID のリストを解放する

関数インタフェース

```
dre_return_t dre_app_release_registered_pid_list (
        dre_app_pid_list_t              list
);
```

引数

dre_app_pid_list_t	list // PID のリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.25 dre_app_release_snapshot_list

概要

コンテナのスナップショットのリストを解放する

関数インタフェース

```
dre_return_t dre_app_release_snapshot_list (
        dre_list_t                      list
);
```

引数

dre_list_t	list // スナップショットのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.26 dre_app_remove_base

概要

ベースイメージを削除する

関数インタフェース

```
dre_return_t dre_app_remove_base (
        const char *                    base
);
```

引数

const char *	base // ベースイメージ名

戻り値

0	成功

0 以外	エラー(1.1.2.1 参照)

1.2.5.27 dre_app_remove_checkpoint

概要

コンテナのチェックポイントを削除する(未実装)

関数インタフェース

```
dre_return_t dre_app_remove_checkpoint (
        const dre_app_handle *          h,
        const char *                    tag
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// チェックポイント名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.28 dre_app_remove_snapshot

概要

コンテナのスナップショットを削除する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_remove_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// スナップショット名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.29 dre_app_restart

概要

チェックポイントからコンテナの動作を再開する(未実装)

関数インタフェース

```
dre_return_t dre_app_restart (
        const dre_app_handle *          h,
        const char *                    tag
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// チェックポイント名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.30 dre_app_resume

概要

一時停止中のコンテナの動作を再開する

関数インタフェース

```
dre_return_t dre_app_resume (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.31 dre_app_save_snapshot

概要

コンテナのスナップショットを作成する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_save_snapshot (
        const dre_app_handle *          h,
        const char *                    tag
```

```
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	tag	// スナップショット名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.32 dre_app_set_resource

概要

リソース使用量に制限をかける
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_set_resource (
        const dre_app_handle *          h,
        const char *                    cgroup,
        dre_app_resource_value_t *      resource
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名
dre_app_resource_value_t *	resource	// リソース

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.33 dre_app_start

概要

コンテナの動作を開始する

関数インタフェース

```
dre_return_t dre_app_start (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.34 dre_app_status

概要

コンテナの状態を取得する

関数インタフェース

```
dre_return_t dre_app_status (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

DRE_APP_CONTAINER_STATUS_STOP	停止中
DRE_APP_CONTAINER_STATUS_RUNNING	動作中
DRE_APP_CONTAINER_STATUS_SUSPENDED	一時停止中
DRE_APP_CONTAINER_STATUS_BUSY	ビジー
上記以外	エラー(1.1.2.1 参照)

1.2.5.35 dre_app_stop

概要

コンテナの動作を停止する

関数インタフェース

```
dre_return_t dre_app_stop (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.36 dre_app_suspend

概要

動作中のコンテナを一時停止する

関数インタフェース

```
dre_return_t dre_app_suspend (
        const dre_app_handle *          h
);
```

引数

const dre_app_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.37 dre_app_unbind_pid

概要

コンテナ外のプロセスをコントロールグループからアンバインドする
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_unbind_pid (
        const dre_app_handle *          h,
        const char *                    cgroup,
        const int                       pid
);
```

引数

const dre_app_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名
const int	pid	// 対象のプロセス ID

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.38 dre_app_unbind_vpid

概要

コンテナ上のプロセスをコントロールグループからアンバインドする
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_unbind_vpid (
        const dre_app_handle *          h,
        const char *                    cgroup,
        const int                       vpid
);
```

引数

```
const dre_app_handle *          h       // ハンドル
const char *                    cgroup  // コントロールグループ名
const int                       vpid    // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.39 dre_app_unregister_pid

概要

プロセスを D-Aware モジュールの管理下から除外する

関数インタフェース

```
dre_return_t dre_app_unregister_pid (
        const dre_app_handle *          h,
        const int                       pid
);
```

引数

```
const dre_app_handle *          h    // ハンドル
const int                       pid  // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.40 dre_app_unregister_vpid

概要

プロセスを D-Aware モジュールの管理下から除外する

関数インタフェース

```
dre_return_t dre_app_unregister_vpid (
        const dre_app_handle *          h,
        const int                       vpid
);
```

引数

```
const dre_app_handle *          h    // ハンドル
const int                       vpid // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.2.5.41 dre_app_unset_resource

概要

リソース使用量の制限を解除する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_app_unset_resource (
        const dre_app_handle *          h,
        const char *                    cgroup,
        dre_app_resource_value_t *      resource
);
```

引数

```
const dre_app_handle *          h          // ハンドル
const char *                    cgroup     // コントロールグループ名
dre_app_resource_value_t *      resource   // リソース
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3 システムコンテナ

1.3.1使用方法

ソースコードにヘッダファイル“dre_sys.h”を include し、コンパイル時に D-RE ライブラリ(libdresys.so または libdresys.a)と D-Aware ライブラリ(libdaware.so または libdaware.a)をリンクして下さい。

1.3.2定数・マクロ

1.3.2.1 dre_sys_resource_t

概要

リソースの種類

実装

```
typedef enum {
    DRE_SYS_CONTAINER_RESOURCE_CPU = 1,    // CPU
    DRE_SYS_CONTAINER_RESOURCE_CPU_CORE,   // CPU コア
    DRE_SYS_CONTAINER_RESOURCE_MEMORY,     // メモリ
    DRE_SYS_CONTAINER_RESOURCE_MEMSW       // メモリとスワップ領域
} dre_sys_resource_t;
```

1.3.2.2 dre_sys_status_t

概要

コンテナの状態

実装

```
typedef enum {
    DRE_SYS_CONTAINER_STATUS_STOP = 11,    // 停止中
    DRE_SYS_CONTAINER_STATUS_RUNNING,      // 動作中
    DRE_SYS_CONTAINER_STATUS_SUSPENDED,    // 一時停止中
    DRE_SYS_CONTAINER_STATUS_BUSY          // ビジー
} dre_sys_status_t;
```

1.3.2.3 dre_sys_type_t

概要

コンテナの種類

実装

```
typedef enum {
    DRE_SYS_CONTAINER_TYPE_KVM = 100 // KVM
```

```
} dre_sys_type_t;
```

1.3.3構造体

1.3.3.1 dre_sys_data

概要

コンテナの設定

実装

```
typedef struct {
        dre_sys_type_t type;            // コンテナの種類
        union {
                union {
                        int memory_size;    // コンテナに割り当てるメモリ容量
                        char *base;         // コンテナのベースイメージ(NULL でデフォルトを使用)
                        char *rootfs;       // KVM イメージの代わりにゲスト OS が使用するパーティション
                                            // (NULL で使用しない)
                } kvm;
        } udata;
} dre_sys_data;
```

ベースイメージ名には、dre-sys commit で作成したベースイメージ(KVM イメージ)が指定可能です。ベースイメージを指定すると、そのイメージをベースとした差分イメージを作成し、その差分イメージを使用したシステムコンテナが作成できます。rootfs を指定すると、当該パーティションにインストールされた OS をゲスト OS として使用可能です。

1.3.3.2 dre_sys_handle

概要

コンテナのハンドル

実装

```
typedef struct {
        dre_sys_type_t type;    // コンテナの種類
        char *name;             // コンテナ名
} dre_sys_handle;
```

1.3.3.3 dre_sys_resource_value_t

概要

リソース使用状況

実装

```
typedef struct {
        dre_sys_resource_t type;                // リソースの種類
        union {
                union {
                        int parcent;            // CPU 使用率(%)
                        char *cores;            // CPU コア
                } cpu;
                union {
                        int usage_in_bytes;     // メモリ使用量(Bytes)
                        int memsw_usage_in_bytes;  // メモリ+スワップ領域使用量(Bytes)
                } mem;
        } udata;
} dre_sys_resource_value_t;
```

1.3.4型宣言

なし

1.3.5関数

1.3.5.1 dre_sys_bind_pid

概要

コンテナ外のプロセスをコントロールグループにバインドする

関数インタフェース

```
dre_return_t  dre_sys_bind_pid  (
        const dre_sys_handle *          h,
        const char *                    cgroup,
        const int                       pid
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名
const int	pid	// 対象のプロセス ID

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.2 dre_sys_checkpoint

概要

コンテナのチェックポイントを作成する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_checkpoint (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

引数

```
const dre_sys_handle *          h     // ハンドル
const char *                    tag   // チェックポイント名
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.3 dre_sys_close

概要

コンテナのハンドルを解放する

関数インタフェース

```
dre_return_t dre_sys_close (
        const dre_sys_handle *          h
);
```

引数

```
const dre_sys_handle *          h     // ハンドル
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.4 dre_sys_commit_snapshot

概要

スナップショットを使用して、コンテナのベースとなるイメージを作成する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_commit_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag,
```

```
        const char *                base
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	tag	// スナップショット名
const char *	base	// 作成するイメージ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.5 dre_sys_create_cgroup

概要

コンテナ上にコントロールグループを作成する

関数インタフェース

```
dre_return_t dre_sys_create_cgroup (
        const dre_sys_handle *      h,
        const char *                cgroup
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.6 dre_sys_create_container

概要

コンテナを生成する

関数インタフェース

```
dre_return_t dre_sys_create_container (
        const char *                name,
        const dre_sys_data *        data
);
```

引数

const char *	name	// コンテナ名

```
const dre_sys_data *        data   // コンテナの設定
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.7 dre_sys_destroy_cgroup

概要

コンテナ上のコントロールグループを破棄する

関数インタフェース

```
dre_return_t dre_sys_destroy_cgroup (
        const dre_sys_handle *      h,
        const char *                cgroup
);
```

引数

```
const dre_sys_handle *      h        // ハンドル
const char *                cgroup   // コントロールグループ名
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.8 dre_sys_destroy_container

概要

コンテナを破棄する

関数インタフェース

```
dre_return_t dre_sys_destroy_container (
        const char *                name
);
```

引数

```
const char *                name   // コンテナ名
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.9 dre_sys_get_base_list

概要

ベースイメージのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_sys_release_base_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_sys_get_base_list (
        dre_list_t *                    list
);
```

引数

dre_list_t *	list	// コンテナのリストへのポインタ

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.10 dre_sys_get_checkpoint_list

概要

コンテナのチェックポイントのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_sys_release_checkpoint_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_sys_get_checkpoint_list (
        const dre_sys_handle *          h,
        dre_list_t *                    list
);
```

引数

const dre_sys_handle *	h	// ハンドル
dre_list_t *	list	// チェックポイントのリストへのポインタ

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.11 dre_sys_get_container_list

概要

コンテナのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_sys_release_container_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_sys_get_container_list (
        dre_list_t *                    list
);
```

引数

dre_list_t *	list	// コンテナのリストへのポインタ

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.12 dre_sys_get_pid

概要

コンテナのプロセス ID を取得する

関数インタフェース

```
dre_return_t dre_sys_get_pid (
        const dre_sys_handle *          h,
        int *                           pid
);
```

引数

const dre_sys_handle *	h	// ハンドル
int *	pid	// プロセス ID

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.13 dre_sys_get_resource

概要

リソース使用量を取得する

関数インタフェース

```
dre_return_t dre_sys_get_resource (
        const dre_sys_handle *          h,
        const int                       pid,
        dre_sys_resource_value_t *      resource
);
```

引数

```
const dre_sys_handle *          h         // ハンドル
const int                       pid       // プロセスの PID
dre_sys_resource_value_t *      resource  // リソース
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.14 dre_sys_get_snapshot_list

概要

コンテナのスナップショットのリストを取得する
(注: 関数内でメモリ領域の確保を行うため、使用後は必ず dre_sys_release_snapshot_list()関数で解放して下さい。)

関数インタフェース

```
dre_return_t dre_sys_get_snapshot_list (
        const dre_sys_handle *      h,
        dre_list_t *                list
);
```

引数

```
const dre_sys_handle *          h     // ハンドル
dre_list_t *                    list  // スナップショットのリストへのポインタ
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.15 dre_sys_immigration_start

概要

マイグレーション先のコンテナを生成する

関数インタフェース

```
dre_return_t dre_sys_immigration_start (
        const char *                name,
        const int                   port
);
```

引数

```
const char *                    name  // コンテナ名(マイグレーション元と同名)
const int                       port  // マイグレーションに使用するポート番号
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.16 dre_sys_load_snapshot

概要

コンテナをスナップショットの状態に復元する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_load_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

引数

```
const dre_sys_handle *          h     // ハンドル
const char *                    tag   // スナップショット名
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.17 dre_sys_migration_start

概要

マイグレーションを開始する

関数インタフェース

```
dre_return_t dre_sys_migration_start (
        const dre_sys_handle *          h,
        const char *                    target,
        const int                       port
);
```

引数

```
const dre_sys_handle *          h       // ハンドル
const char *                    target  // マイグレーション先の IP アドレス
const int                       port    // マイグレーションに使用するポート番号
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.18 dre_sys_open

概要

コンテナのハンドルを取得する

関数インタフェース

```
dre_sys_handle * dre_sys_open (
        const char *                    name
);
```

引数

const char *	name	// コンテナ名

戻り値

NULL	取得失敗
NULL 以外	取得したハンドルへのポインタ

1.3.5.19 dre_sys_release_base_list

概要

ベースイメージのリストを解放する

関数インタフェース

```
dre_return_t dre_sys_release_base_list (
        dre_list_t                      list
);
```

引数

dre_list_t	list	// コンテナのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.20 dre_sys_release_checkpoint_list

概要

コンテナのチェックポイントのリストを解放する

関数インタフェース

```
dre_return_t dre_sys_release_checkpoint_list (
        dre_list_t                      list
);
```

引数

dre_list	list // チェックポイントのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.21 dre_sys_release_container_list

概要

コンテナのリストを解放する

関数インタフェース

```
dre_return_t dre_sys_release_container_list (
        dre_list_t                  list
);
```

引数

dre_list_t	list // コンテナのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.22 dre_sys_release_snapshot_list

概要

コンテナのスナップショットのリストを解放する

関数インタフェース

```
dre_return_t dre_sys_release_snapshot_list (
        dre_list_t                  list
);
```

引数

dre_list	list // スナップショットのリスト

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.23 dre_sys_remove_base

概要

ベースイメージを削除する

関数インタフェース

```
dre_return_t dre_sys_remove_base (
        const char *                    base
);
```

引数

const char *	base // ベースイメージ名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.24 dre_sys_remove_checkpoint

概要

コンテナのチェックポイントを削除する

関数インタフェース

```
dre_return_t dre_sys_remove_checkpoint (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

引数

const dre_sys_handle *	h // ハンドル
const char *	tag // チェックポイント名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.25 dre_sys_remove_snapshot

概要

コンテナのスナップショットを削除する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_remove_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
```

```
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	tag	// スナップショット名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.26 dre_sys_restart

概要

チェックポイントからコンテナの動作を再開する
(コンテナ動作中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_restart (
        const dre_sys_handle *      h,
        const char *                tag
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	tag	// チェックポイント名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.27 dre_sys_resume

概要

一時停止中のコンテナの動作を再開する

関数インタフェース

```
dre_return_t dre_sys_resume (
        const dre_sys_handle *      h
);
```

引数

const dre_sys_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.28 dre_sys_save_snapshot

概要

コンテナのスナップショットを作成する
(コンテナ停止中のみ使用可能)

関数インタフェース

```
dre_return_t dre_sys_save_snapshot (
        const dre_sys_handle *          h,
        const char *                    tag
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	tag	// スナップショット名

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.29 dre_sys_set_resource

概要

リソース使用量に制限をかける

関数インタフェース

```
dre_return_t dre_sys_set_resource (
        const dre_sys_handle *          h,
        const char *                    cgroup,
        dre_sys_resource_value_t *      resource
);
```

引数

const dre_sys_handle *	h	// ハンドル
const char *	cgroup	// コントロールグループ名
dre_sys_resource_value_t *	resource	// リソース

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.30 dre_sys_start

概要

コンテナの動作を開始する

関数インタフェース

```
dre_return_t dre_sys_start (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.31 dre_sys_share

概要

コンテナを NFS サーバ上に置く

関数インタフェース

```
dre_return_t dre_sys_share (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.32 dre_sys_status

概要

コンテナの状態を取得する

関数インタフェース

```
dre_return_t dre_sys_status (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle * h // ハンドル

戻り値

DRE_SYS_CONTAINER_STATUS_STOP	停止中
DRE_SYS_CONTAINER_STATUS_RUNNING	動作中
DRE_SYS_CONTAINER_STATUS_SUSPENDED	一時停止中
DRE_SYS_CONTAINER_STATUS_BUSY	ビジー
上記以外	エラー(1.1.2.1 参照)

1.3.5.33 dre_sys_stop

概要

コンテナの動作を停止する

関数インタフェース

```
dre_return_t dre_sys_stop (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle * h // ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.34 dre_sys_suspend

概要

動作中のコンテナを一時停止する

関数インタフェース

```
dre_return_t dre_sys_suspend (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle * h // ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.35 dre_sys_unbind_pid

概要

プロセスをコントロールグループからアンバインドする

関数インタフェース

```
dre_return_t dre_sys_unbind_pid (
        const dre_sys_handle *          h,
        const char *                    cgroup,
        const int                       pid
);
```

引数

```
const dre_sys_handle *          h       // ハンドル
const char *                    cgroup  // コントロールグループ名
const int                       pid     // 対象のプロセス ID
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.36 dre_sys_unset_resource

概要

リソース使用量の制限を解除する

関数インタフェース

```
dre_return_t dre_sys_unset_resource (
        const dre_sys_handle *          h,
        const char *                    cgroup,
        dre_sys_resource_value_t *      resource
);
```

引数

```
const dre_sys_handle *          h       // ハンドル
const char *                    cgroup  // コントロールグループ名
dre_sys_resource_value_t *      resource // リソース
```

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

1.3.5.37 dre_sys_unshare

概要

コンテナを NFS サーバから削除する

関数インタフェース

```
dre_return_t dre_sys_unshare (
        const dre_sys_handle *          h
);
```

引数

const dre_sys_handle *	h	// ハンドル

戻り値

0	成功
0 以外	エラー(1.1.2.1 参照)

2 D-Aware Application API

2.1 使用の流れ

本 API を使用したプログラムを、D-Aware Application と呼びます。

D-Aware Application は、プログラム起動時に daware_termination_register_callback()をコールする必要があります。daware_termination_register_callback()は、プログラムが受信可能なシグナルと、プログラムが当該シグナルを受信した際にコールバックされる関数を設定します。

D-Aware Application は、D-RE のアプリケーションコンテナ上で動作することを前提に設計されており、PID と VPID の 2 つのプロセス ID を持ちます。PID はホスト OS で割り当てられたもので、ホスト OS からのみ参照できます。VPID はコンテナ上で割り当てられたもので、アプリケーションコンテナ上からのみ参照できます。

daware_termination_register_callback()がコールされると、プログラムの PID と VPID が紐付けされ、/proc/daware に登録されます。

D-Aware Application は、daware_log()をコールすることで、任意のタイミングでログ出力を行うことが出来ます。ログのフォーマットは、daware_log_set_format()で指定可能です。

D-Aware Application は、終了時に daware_termination_unregister_callback()をコールする必要があります。この関数がコールされることで、プログラムのプロセス ID が/proc/daware から削除されます。

2.2 使用方法

ソースコードにヘッダファイル“daware.h”を include し、コンパイル時に D-Aware ライブラリ(libdaware.so または libdaware.a)をリンクして下さい。

2.3 定数・マクロ

2.3.1 DLOG

概要

priority を指定してログを出力する

実装

```
DLOG (priority, fmt, ...)
```

引数

```
priority        // facility と level の OR したものを指定する
fmt, ...        // printf(fmt, ...)と同様の使い方
```

2.3.2 DLOG_EMERG/DLOG_ALERT/DLOG_CRIT/DLOG_ERR/DLOG_WARNING/DLOG_NOTICE/DLOG_INFO/DLOG_DEBUG

概要

各 priority でログを出力する

実装

```
DLOG_EMERG(fmt, ...)
DLOG_ALERT(fmt, ...)
DLOG_CRIT(fmt, ...)
DLOG_ERR(fmt, ...)
DLOG_WARNING(fmt, ...)
DLOG_NOTICE(fmt, ...)
DLOG_INFO(fmt, ...)
DLOG_DEBUG(fmt, ...)
```

引数

```
fmt, ...                // printf(fmt, ...)と同様の使い方
```

2.4 構造体

なし

2.5 型宣言

2.5.1 DTermination

概要

終了コールバックのハンドル

実装

```
typedef long            DTermination;
```

2.6 関数

2.6.1 daware_log

概要

ログを出力する。

関数インタフェース

```
daware_log (
        int             priority,
        const char *    file,
        int             line,
        const char *    func,
        const char *    fmt, ...
);
```

引数

```
int             priority    // facility と level(2.7 参照)の
                            // OR したものを指定する。
const char *    file        // ファイル名(__FILE__)。
int             line        // 行番号(__LINE__)。
const char *    func        // 関数名(__func__)。
const char *    fmt, ...    // printf(fmt, ...)と同様の使い方。
```

戻り値

なし

2.6.2 daware_log_close

概要

ログをクローズする。呼び出しは必須ではない。

関数インタフェース

```
void daware_log_close ();
```

引数

なし

戻り値

なし

2.6.3 daware_log_open

概要

ログをオープンする。呼び出しは必須ではない。

関数インタフェース

```
void daware_log_open (
        const char *    ident,
        int             option,
        int             facility
);
```

引数

```
const char *    ident       // ログを出力したプログラムを識別するための文字列。
                            // ログに出力される。
int             option      // openlog と同様。
                            // 値(2.7 参照)を OR したものが openlog()の option 引数になる。
                            // (openlog については、man syslog を参照。)
int             facility    // デフォルトの facility(2.7 参照)を設定する。
```

戻り値

なし

2.6.4 daware_log_set_format

概要

ログのフォーマットを指定する。

関数インタフェース

```
daware_log_set_format (
        const char *    logfmt
);
```

引数

```
const char *        logfmt          //  ログのフォーマット。
                                    //  以下のタグを使用することができる。

                    %faci%  :   syslog facility{cron|daemon|main|user|...}
                    %lvl%   :   syslog level{err|warning|notice|debug|...}
                    %time%  :   date and time
                    %host%  :   host name
                    %prog%  :   program name
                    %pid%   :   PID
                    %msg%   :   message
                    %file%  :   filename
                    %line%  :   line number
                    %func%  :   function name

                    使用例
                        daware_log_set_format("%time% %prog%[%pid%] %lvl%: %msg%");
```

戻り値

なし

2.6.5 daware_termination_register_callback

概要

終了コールバックを登録する

関数インタフェース

```
DTermination * daware_termination_register_callback (
        int             signo,
        int             (*func)(int, void *),
        void *          arg
);
```

引数

```
int             signo           // シグナル
                                // 指定したシグナルを受信すると
                                // コールバックを呼び出す
int             (*func)(int, void *) // 終了コールバック
void *          arg             // 終了コールバックへ渡される引数
```

戻り値

終了コールバックハンドル	成功
NULL	失敗

2.6.6 daware_termination_unregister_callback

概要

終了コールバックを登録解除する

関数インタフェース

```
void daware_termination_unregister_callback (
    DTermination *  dterm
);
```

引数

```
DTermination *      dterm           // 終了コールバックのハンドル
```

戻り値

なし

2.7 その他

2.7.1 facility

概要

メッセージを記録するプログラムのタイプ

指定可能な値

LOG_AUTH
　セキュリティ/認証 メッセージ (非推奨。代わりに LOG_AUTHPRIV を使用すること)
LOG_AUTHPRIV
　セキュリティ/認証 メッセージ (プライベート)
LOG_CRON
　クロックデーモン (cron と at)
LOG_DAEMON
　特定の facility 値を持たないシステムデーモン
LOG_FTP

ftp デーモン
LOG_KERN
カーネルメッセージ (ユーザプロセスから生成することはできない)
LOG_LOCAL0 から LOG_LOCAL7
ローカルな使用のためにリザーブされている
LOG_LPR
ラインプリンタ・サブシステム
LOG_MAIL
メール・サブシステム
LOG_NEWS
USENET ニュース・サブシステム
LOG_SYSLOG
syslogd(8) によって内部的に発行されるメッセージ
LOG_USER (デフォルト)
一般的なユーザレベルメッセージ
LOG_UUCP
UUCP サブシステム

2.7.2 level

概要

メッセージの優先度(レベル)

指定可能な値

LOG_EMERG
システムが使用不可
LOG_ALERT
直ちに行動を起こさなければならない
LOG_CRIT
危険な状態
LOG_ERR
エラーの状態
LOG_WARNING
ワーニングの状態
LOG_NOTICE
通常だが重要な状態
LOG_INFO
インフォメーションメッセージ
LOG_DEBUG
デバッグレベルのメッセージ

2.7.3 option

概要

openlog()に渡す option 引数

指定可能な値

LOG_CONS
　エラーがあれば、システムロガーに送る一方でシステムコンソールにも直接書く。
LOG_NDELAY
　ログ記録用プログラムとの接続を即座に開始する（通常は、最初のメッセージが記録される時に接続を開く)。
LOG_NOWAIT
　メッセージを記録する際に生成される子プロセスの終了を待たない。(GNU C ライブラリは子プロセスを生成しない。したがって、このオプションは Linux では無効である。)
LOG_ODELAY
　LOG_NDELAY の反対。syslog()が呼ばれるまで、接続の開始を行わない。(このオプションはデフォルトであり、特に指定する必要はない。)
LOG_PERROR
　stderr にも出力する。(POSIX.1-2001 では定義されていない)
LOG_PID
　個々のメッセージに PID を含める。