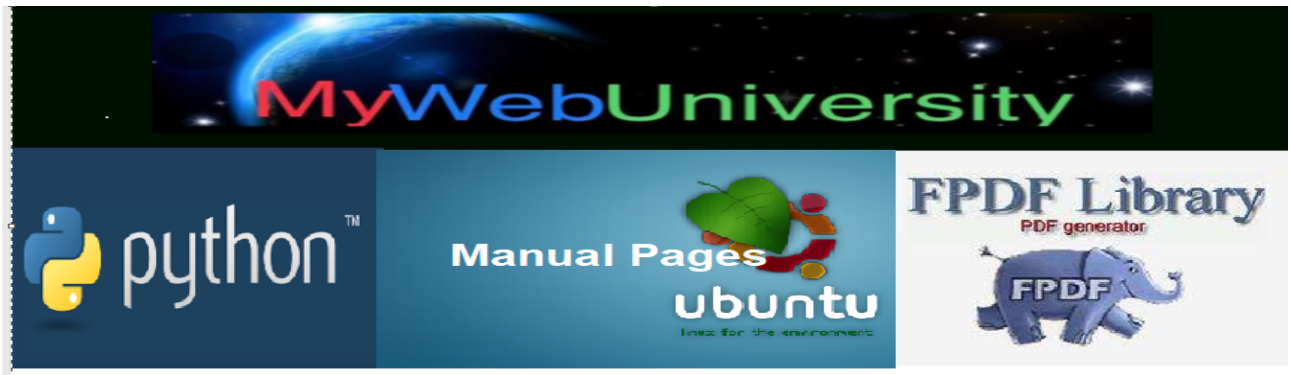

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'procps.3'

$ man procps.3

PROCPS(3) Library Functions Manual PROCPS(3)

NAME

procps - API to access system level information in the /proc filesystem

SYNOPSIS

Five distinct interfaces are represented in this synopsis and named after the files they access in the /proc pseudo filesystem: diskstats, meminfo, slabinfo, stat and vmstat.

```
#include <libproc2/named_interface.h>
int procps_new   (struct info **info);
int procps_ref   (struct info  *info);
int procps_unref (struct info **info);
struct result *procps_get (
    struct info *info,
[   const char *name,      ]   diskstats api only
    enum item item);
struct stack *procps_select (
    struct info *info,
[   const char *name,      ]   diskstats api only
    enum item *items,
    int numitems);
struct reaped *procps_reap (
    struct info *info,
[   enum reap_type what,   ]   stat api only
    enum item *items,
```

```
    int numitems);
struct stack **procps_sort (
    struct info *info,
    struct stack *stacks[],
    int numstacked,
    enum item sortitem,
    enum sort_order order);
```

The above functions and structures are generic but the specific named_interface would also be part of any identifiers. For example, `procps_new' would actually be `procps_meminfo_new' and `info' would really be `diskstats_info', etc.

The same named_interface is used in each header file name with an appended `.h' suffix.

Link with -lproc2.

DESCRIPTION

Overview

Central to these interfaces is a simple `result' structure reflecting an `item' plus its value (in a union with standard C language types as members). All `result' structures are automatically allocated and provided by the library.

By specifying an array of `items', these structures can be organized as a `stack', potentially yielding many results with a single function call. Thus, a `stack' can be viewed as a variable length record whose content and order is determined solely by the user.

As part of each interface there are two unique enumerators. The `noop' and `extra' items exist to hold user values. They are never set by the library, but the `extra' result will be zeroed with each library interaction.

The named_interface header file will be an essential document during user program development. There you will find available items, their return type (the `result' struct member name) and the source for such values. Additional enumerators and structures are also documented there.

Usage

The following would be a typical sequence of calls to these interfaces.

1. procps_new()
2. procps_get(), procps_select() or procps_reap()
3. procps_unref()

The get function is used to retrieve a `result' structure for a single `item'.

Alternatively, a GET macro is available when only the return value is of interest.

The select function can retrieve multiple `result' structures in a single `stack'.

For unpredictable variable outcomes, the diskstats, slabinfo and stat interfaces export a reap function. It is used to retrieve multiple `stacks' each containing multiple `result' structures. Optionally, a user may choose to sort those results.

To exploit any `stack', and access individual `result' structures, a relative_enum is required as shown in the VAL macro defined in the header file. Such values could be hard coded as: 0 through numitems-1. However, this need is typically satisfied by creating your own enumerators corresponding to the order of the `items' array.

Caveats

The new, ref, unref, get and select functions are available in all five interfaces.

For the new and unref functions, the address of an info struct pointer must be supplied. With new it must have been initialized to NULL. With unref it will be reset to NULL if the reference count reaches zero.

In the case of the diskstats interface, a name parameter on the get and select functions identifies a disk or partition name

For the stat interface, a what parameter on the reap function identifies whether data for just CPUs or both CPUs and NUMA nodes is to be gathered.

When using the sort function, the parameters stacks and numstacked would normally be those returned in the `reaped' structure.

RETURN VALUE

Functions Returning an `int'

An error will be indicated by a negative number that is always the inverse of some well known errno.h value.

Success is indicated by a zero return value. However, the ref and unref functions return the current info structure reference count.

Functions Returning an `address'

An error will be indicated by a NULL return pointer with the reason found in the formal errno value.

Success is indicated by a pointer to the named structure.

DEBUGGING

To aid in program development, there is a provision that can help ensure `result' member references agree with library expectations. It assumes that a supplied macro in the header

file is used to access the `result' value.

This feature can be activated through either of the following methods and any discrepancies will be written to stderr.

1) Add CFLAGS='-DXTRA_PROCPS_DEBUG' to any other ./configure options employed.

2) Add #include <procps/xtra-procps-debug.h> to any program after the named interface includes.

This verification feature incurs substantial overhead. Therefore, it is important that it not be activated for a production/release build.

SEE ALSO

procps_misc(3), procps_pids(3), proc(5).