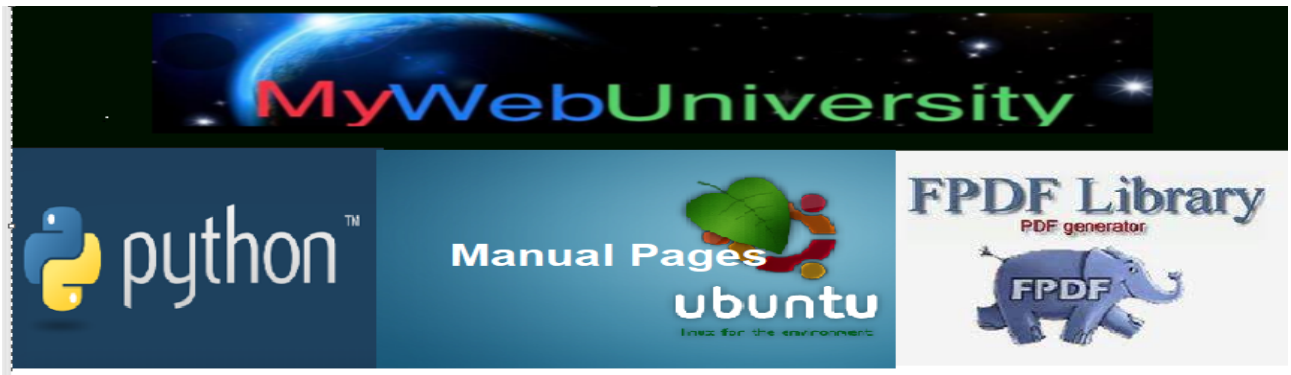

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man procps_pids.3

PROCPS_PIDS(3) Library Functions Manual PROCPS_PIDS(3)

NAME

procps_pids - API to access process information in the /proc filesystem

SYNOPSIS

```
#include <libproc2/pids.h>
int procps_pids_new   (struct pids_info **info, enum pids_item *items, int numitems);
int procps_pids_ref   (struct pids_info  *info);
int procps_pids_unref (struct pids_info **info);
struct pids_stack *procps_pids_get (
    struct pids_info *info,
    enum pids_fetch_type which);
struct pids_fetch *procps_pids_reap (
    struct pids_info *info,
    enum pids_fetch_type which);
struct pids_fetch *procps_pids_select (
    struct pids_info *info,
    unsigned *these,
    int numthese,
    enum pids_select_type which);
struct pids_stack **procps_pids_sort (
    struct pids_info *info,
    struct pids_stack *stacks[],
    int numstacked,
```

```
    enum pids_item sortitem,
    enum pids_sort_order order);
int procps_pids_reset (
    struct pids_info *info,
    enum pids_item *newitems,
    int newnumitems);
struct pids_stack *fatal_proc_unmounted (
    struct pids_info *info,
    int return_self);
```

Link with -lproc2.

DESCRIPTION

Overview

Central to this interface 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 this 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 pids.h 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 this interface.

1. fatal_proc_unmounted()
2. procps_pids_new()
3. procps_pids_get(), procps_pids_reap() or procps_pids_select()
4. procps_pids_unref()

The get function is an iterator for successive PIDs/TIDs, returning those `items' previously identified via new or reset.

Two functions support unpredictable variable outcomes. The reap function gathers data for

all processes while the select function deals with specific PIDs or UIDs. Both can return multiple `stacks' each containing multiple `result' structures. Optionally, a user may choose to sort such 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 <pids> API differs from others in that those items of interest must be provided at new or reset time, the latter being unique to this API. If either the items or numitems parameter is zero at new time, then reset becomes mandatory before issuing any other call.

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.

The get and reap functions use the which parameter to specify whether just tasks or both tasks and threads are to be fetched.

The select function requires an array of PIDs or UIDs as these along with numthese to identify which processes are to be fetched. This function then operates as a subset of reap.

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

Lastly, a fatal_proc_unmounted function may be called before any other function to ensure that the /proc/ directory is mounted. As such, the info parameter would be NULL and the return_self parameter zero. If, however, some items are desired for the issuing program (a return_self other than zero) then the new call must precede it to identify the items and obtain the required info pointer.

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. However, if one survives the fatal_proc_unmounted call, NULL is always returned when return_self is zero.

DEBUGGING

To aid in program development, there are two procps-ng provisions that can be exploited.

The first is a supplied file named `libproc.supp' which may be useful when developing a multi-threaded application. When used with the valgrind `--suppressions=' option, warnings associated with the procps library itself are avoided.

Such warnings arise because the library handles heap based allocations in a thread-safe manner. A single-threaded application will not receive those warnings.

The second provision 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 your project may employ.

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

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

ENVIRONMENT VARIABLE(S)

The value set for the following is unimportant, just its presence.

LIBPROC_HIDE_KERNEL

This will hide kernel threads which would otherwise be returned with a procps_pids_get, procps_pids_select or procps_pids_reap call.

SEE ALSO

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