

NAME

pnotify – event notification framework

SYNOPSIS

```
#include <pnotify.h>

int pnotify_init();
int pnotify_add_watch(struct pnotify_watch *watch);
int pnotify_rm_watch(int wd);
int pnotify_get_event(struct pnotify_event *event);
int pnotify_dispatch();
int pnotify_trap_signal(int signum, void (*cb)(), void *arg);
int pnotify_watch_vnode(char *path, int mask, void (*cb)(), void *arg);
int pnotify_watch_fd(int fd, int mask, void (*cb)(), void *arg);
int pnotify_set_timer(int interval, int mask, void (*cb)(), void *arg);
void pnotify_free();
```

DESCRIPTION

The **pnotify** library provides a portable event notification framework.

pnotify_init() initializes the event queue, and must be called before any of the other **pnotify** functions. **pnotify_free()** releases all of the memory and resources used by **pnotify** and destroys all watches and pending events.

EVENTS

The mask parameter is composed of one or more bitflags from the following list of events:

PN_ATTRIB	The attributes of a file have been modified.
PN_CLOSE	A file descriptor was closed by the remote end.
PN_CREATE	A file was created in a watched directory.
PN_DELETE	The file was deleted.
PN_ERROR	An error occurred in the kernel event queue.
PN_MODIFY	The contents of a file have been modified.
PN_READ	Data can be read from a file descriptor without blocking.
PN_SIGNAL	A signal was delivered to the process.
PN_TIMEOUT	A user-defined time interval has elapsed.
PN_WRITE	Data can be written to a file descriptor without blocking.

When creating a new watch, the following flag may also be used:

PN_ONESHOT Delete the watch after an event occurs.

WATCHES

A watch is a set of events that an application wishes to be notified about. There is a function for each different type of event that can be monitored. All of the functions for creating watches take a ‘mask’ parameter, which is a bitmask of events to watch for. They also take a callback function that will be invoked when the event occurs. The following functions are used to create watches:

pnotify_trap_signal() causes an event to be generated when a signal is received by the process. The signal is otherwise blocked, and will not cause a signal handler to be run.

pnotify_watch_vnode() causes an event to be generated when a file or directory is changed.

pnotify_watch_fd() causes an event to be generated when an open file descriptor is ready for reading, ready for writing, or closed by the remote end.

pnotify_set_timer() causes an event to be generated at a regular interval.

pnotify_add_event() is a low-level function that adds a watch to the list of active watches. The only reason to use this function is to add an event to a different context than the current thread.

When a watch is created, a unique watch descriptor is returned. To delete the watch, call **pnotify_rm_watch()** and pass the watch descriptor as an argument.

ACTIONS

Programs need to respond to events. When a watch is created, a callback function can be provided. This callback function will be executed each time a matching event occurs. It is also possible to process each event manually and construct your own event loop.

pnotify_get_event() waits for a single event to occur, and fills the `pnotify_event` structure with information about the event. If an error occurs in the underlying kernel event queue, an event is returned with the `PN_ERROR` flag set.

The `pnotify_event` structure contains the following fields:

```
struct pnotify_event {
    int      wd;
    int      mask;
    char     name[NAME_MAX + 1];
};
```

pnotify_dispatch() waits for events and invokes the appropriate callback when an event occurs. This function never returns, and is intended to serve as the applications main event loop.

RETURN VALUES

pnotify_add_watch() returns a positive integer that represents a unique watch descriptor, or -1 if an error occurs.

All other functions return zero if successful, or -1 if an error occurs.

EXAMPLES

The following example creates a watch on the `/tmp` directory and prints the names of files that are created or deleted within the directory.

```
struct pnotify_event evt;

pnotify_init();
pnotify_watch_vnode("/tmp", PN_CREATE | PN_DELETE, NULL, NULL);
while (pnotify_get_event(&evt) == 0) {
    if (evt.mask & PN_CREATE)
        printf("file created: %s", evt.name);
    if (evt.mask & PN_DELETE)
        printf("file deleted: %s", evt.name);
    if (evt.mask & PN_ERROR) {
        printf("an error occurred");
        break;
    }
}
```

The next example shows how to use the **pnotify_dispatch()** function. If the `SIGHUP` signal is sent to the process, it prints out a message. After five seconds, the program will terminate.

```
void got_signal(int signum)
{
    printf("got signal %d\n", signum);
}

void got_timeout()
{
    printf("timed out\n");
    exit(0);
}

int main(int argc, char **argv)
{
    pnotify_init();
    pnotify_trap_signal(SIGHUP, got_signal, NULL);
    pnotify_set_timer(5, PN_ONESHOT, got_timeout, NULL);
    pnotify_dispatch();
    /* NOTREACHED */
}
```

THREADSAFETY

pnotify is a multi-threaded library and is fully threadsafe. Each thread must call **pnotify_init()** before calling any other library functions. It is possible for one thread to add a watch to another thread by setting the 'ctx' context variable before calling **pnotify_add_event()**

SEE ALSO

kqueue(4) inotify(7)

HISTORY

pnotify was first released on July 25th, 2007 with support for inotify and kqueue.

AUTHORS

Mark Heily <devel@heily.com>