

# libdaemon: a lightweight daemon framework for OpenBSD

Kyle Isom  
coder@kyleisom.net

February 12, 2011

## Introduction

`libdaemon` is a lightweight daemon framework for OpenBSD. It provides facilities for logging and a signal handler to enable graceful shutdown, as well as file locking to ensure that only a single copy of a given daemon is running at a time. Currently, `libdaemon` is not threaded.

`libdaemon` may be installed using the typical `make && make install` and may be removed with `make uninstall`. It installs to `/usr/local` as per the OpenBSD defaults.

## Using libdaemon

`libdaemon` has four usable functions:

`daemonise` is the daemon initialisation code.

`daemon_set_logfile` is used to set a file to logfile (or alternatively, setup syslog logging).

`daemon_log` writes a simple `char *` buffer to the logfile.

`daemon_vlog` is the varargs variant of `daemon_log`.

A typical daemon would call `daemonise` with the daemon's name as an argument. If the `daemonise()` call returned `EXIT_SUCCESS`, the daemon should call `daemon_set_logfile()` with either the full path to the logfile or `NULL`. At this point, the daemon can move into it's code loop and use the functions `daemon_log` and `daemon_vlog` to write to the logs. The daemon may be gracefully shutdown by sending the daemon the signal `SIGUSR1`.

## Function Reference

### daemonise: daemonise a program

**Prototype:** int daemonise(char \*)

**Arguments:** char \*d\_name: contains the name of the daemon

**Returns:** EXIT\_SUCCESS: program successfully daemonised

EXIT\_FAILURE: program could not be daemonised

**Notes:** there are several reasons why the program might not be able to daemonise. One of the most common is that a copy of the daemon is already running, or the daemon doesn't have the appropriate permissions. By default, the daemon runs as root:daemon and requires the ability to create /var/run/<daemon name>/ and write files in that directory.

### daemon\_set\_logfile: set the daemon's logfile

**Prototype:** int daemon\_set\_logfile(char \*)

**Arguments:** char \*filename: the full path to the logfile (keeping in mind the daemon chdir()'s to / on daemonisation). If set to NULL, the daemon will log to syslog. libdaemon will **not** create any directories; the calling code must ensure the directory component exists.

**Returns:** EXIT\_SUCCESS: the logfile was successfully set. If not logging to syslog, this means the daemon now has an open file descriptor for the logfile that remains open until the daemon is shutdown.

EXIT\_FAILURE: the logfile could not be opened. Another point of failure is the case where the daemon does not have the appropriate permissions to open the logfile (or cannot open the logfile for another reason, such as the directory does not exist).

## **daemon\_log: log a string in the daemon's logfile**

**Prototype:** int daemon\_log\_(int, char \*)

### **daemon\_log**

**Arguments:** 1. **int:** the log level. This is only important when the daemon is using syslog. If the daemon is using its own logfile, this value is ignored. This should be one of the standard syslog(3) enumerations such as LOG\_INFO or LOG\_WARN.

2. **char \*:** the message to be sent to the logfile.

## **daemon\_vlog: send a format string to the logfile**

**Prototype:** int daemon\_vlog\_(int, char \*)

**Arguments:** 1. **int:** the log level, as in `daemon_log`

2. **char \*:** the format string - all the standard best practices and security techniques should be observed when handling format strings!

3. **...:** a list of arguments appropriate to the supplied format string should be supplied as well.

**Notes:** the macro `LIBDAEMON_LOG_MAX` contains the maximum length that a message string may be. By default, this is the 128 bytes. If a filename was passed in, it needs to be within the length of the macro `LIBDAEMON_FILENAME_MAX` which, by default, is 128 bytes. If the filename is longer than this, rather than truncate it, the function will fail.

## Example Daemon

This is a simple daemon that checks the load average every 60 seconds, and if the load is higher than hardcoded value, logs the condition. It is a pretty simple daemon that's not of much use, but it's a good display of how to use the daemon functions.

```
/*
 * testd.c
 * Kyle Isom <coder@kyleisom.net>
 *
 * test daemon code to show usage of libdaemon
 *
 * released under an ISC license.
 */

#include <stdio.h>
#include <stdlib.h>
#include <syslog.h>
#include <unistd.h>
#include "daemon.h"

#define LOAD_WARN_THRESH 1.5

#ifdef USE_SYSLOG
#define LOGFILE NULL
#else
#define LOGFILE "/var/log/testd.log"
#endif

int
main(int argc, char **argv)
{
    double lavg[1];          /* load average */

    /* daemonise the program */
    if (EXIT_SUCCESS != daemonise("testd") {
        syslog(LOG_INFO, "error daemonising!\n");
        return EXIT_FAILURE;
    }

    /* set up logging */
    if (EXIT_SUCCESS == daemon_set_logfile(LOGFILE))
        daemon_log(-1, "running!");
    else
        return result;
}
```

```
/* main run loop */
while (1) {
    /* get one minute load average */
    if (-1 == getloadavg(lavg, 1))
        daemon_log(LOG_INFO, "error retrieving load average!");
    else if (lavg[0] > LOAD_WARN_THRESH)
        daemon_vlog(LOG_WARNING, "load average exceeded %f: "
                    "is %f!\n", LOAD_WARN_THRESH, lavg[0]);
    else
        daemon_log(LOG_INFO, "wakes up...\n");
    sleep(60);
}

/* should not end up here - all shutdowns should be via signal */
return EXIT_SUCCESS;
}
```