

---

# **lockmgr**

***Release 1.8.2***

**Gene C**

**Jan 06, 2026**



# CONTENTS

<b>1</b>	<b>lockmgr</b>	<b>1</b>
1.1	Overview . . . . .	1
1.2	New / Interesting . . . . .	1
<b>2</b>	<b>Using LockMgr Class</b>	<b>3</b>
<b>3</b>	<b>Research on file locking</b>	<b>5</b>
3.1	Linux: C Implementations . . . . .	5
3.2	Python . . . . .	5
3.3	Examples . . . . .	6
3.4	C-code . . . . .	6
3.5	Tests: c_lock_test . . . . .	6
3.6	Python : lock_fcntl . . . . .	7
3.7	Python : lock_flock . . . . .	7
<b>4</b>	<b>Appendix</b>	<b>9</b>
4.1	Installation . . . . .	9
4.2	Dependencies . . . . .	9
4.3	Philosophy . . . . .	10
4.4	License . . . . .	10
<b>5</b>	<b>License</b>	<b>11</b>
<b>6</b>	<b>API Reference</b>	<b>13</b>
6.1	lockmgr . . . . .	13
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



**LOCKMGR**

## 1.1 Overview

lockmgr : Python Class implementing file locking

All git tags are signed with [arch@sapience.com](mailto:arch@sapience.com) key which is available via WKD or download from <https://www.sapience.com/tech>. Add the key to your package builder gpg keyring. The key is included in the Arch package and the source= line with *?signed* at the end can be used to verify the git tag. You can also manually verify the signature

## 1.2 New / Interesting

### 1.8.0

- Switch packaging from hatch to uv
- Testing to confirm all working on python 3.14.2
- License GPL-2.0-or-later

### Older

- Tidy ups: PEP-8, PEP-257, PEP-484 PEP-561
- improve reference API doc.
- Add py.typed so type checkers like mypy can be used with the module.



## USING LOCKMGR CLASS

Using the class is straightforward. Choose a file to use as the lockfile - avoid NFS mounts. I find using */tmp* works well since its a *tmpfs* file system.

An sample application is available in the examples directory. To see it in action, simply run *test\_lock.py* in 2 different terminals. One process will acquire the lock while the other will wait until its released.

Typical usage:

```
import os
from lockmgr import LockMgr

lockdir = '/tmp/xxx'
os.makedirs(lockdir, exist_ok=True)
lockfile = os.path.join(lockdir, 'foo.lock')
lockmgr = LockMgr(lockfile)

...

# Acquire lock
if lockmgr.acquire_lock(wait=True, timeout=30):
    # do stuff
    ...

    # release lock
    lockmgr.release_lock()
else:
    # failed to get lock
    ...
```





## RESEARCH ON FILE LOCKING

I share my notes and research on file locking. All code samples are available in the examples/research directory.

### 3.1 Linux: C Implementations

File Locking has linux kernel support via standard C library using `fcntl()`. The mechanism uses 'struct flock' as the communication mechanism with (posix) `fcntl()`.

The standard locking mechanism uses `F_SETLK`. This lock lives at the process level and is the original locking in linux. Around 2015 or so 'open file description' locks came to be via `F_OFD_SETLK` (See<sup>1</sup> and<sup>2</sup>). These are at the open file level. So while `F_SETLK` is not passed to child processes, OFD locks are. And OFD locks remain attached to the open file handle. This can be enormously useful and also surprising.

Posix locks are attached to (inode, pid) pair which means they work at the process level. If thread opens same file, even tho has different file handle, when that thread closes the fd, the lock is released for all threads in same process.

OFD locking was introduced to deal with this. Locking is attached to the "open file" and not the PID.

linux also provides "lockf" which is a wrapper around `fcntl()` locks - should only be used in simple cases and will interact with "normal" `fcntl()` locks - caveat emptor.

Summary:

- Posix Lock: `fcntl()` - `F_SETLK`
- OFD Lock: `fcntl()` - `F_OFD_SETLK`
- `lockf` - bah

NB. The flock struct contains `l_pid` - this MUST be 0 for OFD locks and PID for posix locks.

Its also worth noting that locking can be file system dependent. In particular NFS should probably be avoided. Since my dominant use case is single machine, multiple process, I use `/tmp` which is a TMPFS file system and works well.

### 3.2 Python

Python provides for same locking mechanisms - I recommend only 1 way for file locks in python. Python library provides for:

- `fcntl.fcntl` => do not use

As with C there is support for `F_SETLK` and `F_OFD_SETLK`. While these work fine, they require using 'struct' module to 'pack' and 'unpack' the C flock struct. To make this work the caller must provide the sizes (coded with letters as per the python struct module) of each element being packed.

---

<sup>1</sup> File private locks <https://lwn.net/Articles/586904/>

<sup>2</sup> Open File Description <https://lwn.net/Articles/640404/>

The python 3.12 docs have examples<sup>3</sup>, and while they may well work for a Sun workstation or similar, if you have one, the struct element sizes dont seem correct for X86\_64.

I provide a little C-program to print out the correct byte sizes which you can then map to the python struct letter codes<sup>4</sup>

This approach is brittle - its one thing when you are coding with your own C structures, its another entirely when using system ones - these sizes should be compiled into python - while these routines work I strongly recommend not using them for this reason.

- `fcntl.lockf` => do not use

Wrapper around `fcntl()` - in spite of name this is NOT C `lockf()` function.

- `fcntl.flock` => *use this one*

Wrapper around `fcntl` with OFD support. i.e. this lock is associated with open file descriptor. This is what I use and recommend.

## 3.3 Examples

## 3.4 C-code

Sample code for F\_SETLK and F\_OFD\_SETLK To compile:

```
make
```

Builds 2 programs - *flock\_sizes* and *c\_lock\_test*.

*flock\_sizes* is used To print size of struct flock elements which provide the correct sizes to use in python `fcntl.fcntl` approach.

```
./flock_sizes
```

The test program demonstrates locking with and without OFD. To run the test program see the *Tests: c\_lock\_test* section below.

## 3.5 Tests: c\_lock\_test

To run locking tests, use 2 terminals. Run `c_lock_test` in both. The first will acquire lock while second will fail until first exits or is interrupted.

### 3.5.1 Test 1 : Using F\_SETLK

```
./c_lock_test
```

### 3.5.2 Test 2 : Using F\_OFD\_SETLK

Repeat test but with argument to turn on OFD

```
./c_lock_test ofd
```

Test (1) and (2) both work.

---

<sup>3</sup> Python `fcntl` docs: <https://docs.python.org/3/library/fcntl.html>

<sup>4</sup> Python `struct` module: <https://docs.python.org/3/library/struct.html>

## 3.6 Python : lock\_fcntl

F\_SETLK and F\_OFD\_SETLK tests in python. Run test in 2 terminals as above:

### 3.6.1 Test 3 : Using F\_SETLK

```
./lock_fcntl.py
```

### 3.6.2 Test 4 : Using F\_OFD\_SETLK

```
./lock_fcntl.py ofd
```

Test (3) and (4) both work.

## 3.7 Python : lock\_flock

This is what I am using. As above, run test in 2 terminals.

### 3.7.1 Test 5 :

```
./lock_fcntl.py
```

Test (5) works.



## 4.1 Installation

### Available on

- [Github](#)
- [Archlinux AUR](#)

On Arch you can build using the provided PKGBUILD in the packaging directory or from the AUR. To build manually, clone the repo and :

```
rm -f dist/*  
/usr/bin/python -m build --wheel --no-isolation  
root_dest="/"   
./scripts/do-install $root_dest
```

When running as non-root then set root\_dest a user writable directory

## 4.2 Dependencies

- Run Time :
  - python (3.11 or later)
- Building Package:
  - git
  - hatch (aka python-hatch)
  - wheel (aka python-wheel)
  - build (aka python-build)
  - installer (aka python-installer)
  - rsync
- Optional for building docs:
  - sphinx
  - texlive-latexextra (archlinux packaguing of texlive tools)

## 4.3 Philosophy

We follow the *live at head commit* philosophy as recommended by Google's Abseil team<sup>5</sup>. This means we recommend using the latest commit on git master branch.

## 4.4 License

Created by Gene C. and licensed under the terms of the GPL-2.0-or-later license.

- SPDX-License-Identifier: GPL-2.0-or-later
- Copyright (c) 2023 Gene C

---

<sup>5</sup> <https://abseil.io/about/philosophy#upgrade-support>

## LICENSE

lockmgr: Python Lock Manager Class

Copyright © 2022-present Gene C <[arch@sapience.com](mailto:arch@sapience.com)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.





## API REFERENCE

This page contains auto-generated API reference documentation<sup>1</sup>.

### 6.1 lockmgr

locking

#### 6.1.1 Submodules

##### lockmgr.class\_lock

File locks

##### Module Contents

**class** **LockMgr**(*lockfile*)

Robust file locking manager.

**acquire\_lock**(*wait: bool = False, timeout: int = 30*) → bool

Try to acquire a lock.

**Args:**

wait (bool):

If True and timeout > 0: wait until lock is acquired (up to 10 attempts). This can be racy from the time inotify returns till we acquire lock will fail and we will try again. If False, then do not retry if unable to acquire lock on first attempt.

timeout (int): Number of seconds > 0 to wait between attempts to acquire the lock Will retry up to 10 times.

Returns: bool: True if lock acquired

**release\_lock**()

Release an Acquired Lock

Drop the acquired lock. No-op if there is no acquired lock.

##### lockmgr.version

Project lockmgr

---

<sup>1</sup> Created with sphinx-autoapi



## PYTHON MODULE INDEX

|

`lockmgr`, 13

`lockmgr.class_lock`, 13

`lockmgr.version`, 13



## INDEX

### A

`acquire_lock()` (*LockMgr* method), 13

### L

`lockmgr`

    module, 13

`LockMgr` (class in `lockmgr.class_lock`), 13

`lockmgr.class_lock`

    module, 13

`lockmgr.version`

    module, 13

### M

module

`lockmgr`, 13

`lockmgr.class_lock`, 13

`lockmgr.version`, 13

### R

`release_lock()` (*LockMgr* method), 13