**PYTHON ASSIGNMENT**

**MUSKAN BHATIA**

**16CSU219**

**Project Title: TEXT FILE LOCK**

***Introduction:-***

Two threads or processes want to modify the same file at the same time. Here’s a Python solution that covers such problems in Mac OSX and Linux.

A lock is a way to tell other threads or processes that a resource (like a file) is in use. Unfortunately, file locks on Unix are advisory only. This means that, by default, processes won’t care if you’ve locked a file. Rule number one of file locking is that you need two processes that want to work together. Thankfully, if you’re the developer, you can force them both to play nice.

FileLock vs SoftFileLock

The \**FileLock*\* is platform dependent while the \**SoftFileLock*\* is not. Use the \**FileLock*\* if all

instances of your application are running on the same host and a \**SoftFileLock*\* otherwise.

The \**SoftFileLock*\* only watches the existence of the lock file. This makes it ultra portable, but

also more prone to dead locks if the application crashes. You can simply delete the lock file in

such cases.

You lock and unlock files on Unix using the system call flock. You can see the documentation by running man flock.

***Procedure:-***

*A \*FileLock\* is used to indicate another process of your application that a resource or working*

*directory is currently used. To do so, create a \*FileLock\* first:*

code-block:: python

from filelock import Timeout, FileLock

file\_path = "high\_ground.txt"

lock\_path = "high\_ground.txt.lock"

lock = FileLock(lock\_path, timeout=1)

*The lock object supports multiple ways for acquiring the lock, including the ones used to acquire*

*standard Python thread locks:*

code-block:: python

with lock:

open(file\_path, "a").write("Hello there!")

lock.acquire()

try:

open(file\_path, "a").write("General Kenobi!")

finally:

lock.release()

*The \*acquire()\* method accepts also a \*timeout\* parameter. If the lock cannot be acquired*

*within \*timeout\* seconds, a \*Timeout\* exception is raised.:*

code-block:: python

try:

with lock.acquire(timeout=10):

open(file\_path, "a").write("I have a bad feeling about this.")

except Timeout:

print("Another instance of this application currently holds the lock.")

*The lock objects are recursive locks, which means that once acquired, they will not block on*

*successive lock requests:*

code-block:: python

def cite1():

with lock:

open(file\_path, "a").write("I hate it when he does that.")

def cite2():

with lock:

open(file\_path, "a").write("You don't want to sell me death sticks.")

# *The lock is acquired here.*

with lock:

cite1()

cite2()

# *And released here.*

**Expected Outcome:-**

The code will help us to lock and unlock the text files as per our requirement.