import os

import sys

import shutil

import logging

import tempfile

from optparse import OptionParser

from flvlib import \_\_versionstr\_\_

from flvlib.constants import TAG\_TYPE\_AUDIO, TAG\_TYPE\_VIDEO, TAG\_TYPE\_SCRIPT

from flvlib.constants import AAC\_PACKET\_TYPE\_SEQUENCE\_HEADER

from flvlib.constants import H264\_PACKET\_TYPE\_SEQUENCE\_HEADER

from flvlib.primitives import make\_ui8, make\_ui24, make\_si32\_extended

from flvlib.astypes import MalformedFLV

from flvlib.tags import FLV, EndOfFile, AudioTag, VideoTag, ScriptTag

from flvlib.helpers import force\_remove

log = logging.getLogger('flvlib.retimestamp-flv')

class\_to\_tag = {

AudioTag: TAG\_TYPE\_AUDIO,

VideoTag: TAG\_TYPE\_VIDEO,

ScriptTag: TAG\_TYPE\_SCRIPT

}

def is\_nonheader\_media(tag):

if isinstance(tag, ScriptTag):

return False

if isinstance(tag, AudioTag):

return tag.aac\_packet\_type != AAC\_PACKET\_TYPE\_SEQUENCE\_HEADER

if isinstance(tag, VideoTag):

return tag.h264\_packet\_type != H264\_PACKET\_TYPE\_SEQUENCE\_HEADER

def output\_offset\_tag(fi, fo, tag, offset):

new\_timestamp = tag.timestamp - offset

# do not offset non-media and media header

if not is\_nonheader\_media(tag):

new\_timestamp = tag.timestamp

# write the FLV tag value

fo.write(make\_ui8(class\_to\_tag[tag.\_\_class\_\_]))

# the tag size remains unchanged

fo.write(make\_ui24(tag.size))

# wirte the new timestamp

fo.write(make\_si32\_extended(new\_timestamp))

# seek inside the input file

# seek position: tag offset + tag (1) + size (3) + timestamp (4)

fi.seek(tag.offset + 8, os.SEEK\_SET)

# copy the tag content to the output file

# content size: tag size + stream ID (3) + previous tag size (4)

fo.write(fi.read(tag.size + 7))

def retimestamp\_tags\_inplace(f, fu):

flv = FLV(f)

offset = None

for tag in flv.iter\_tags():

if offset is None and is\_nonheader\_media(tag):

offset = tag.timestamp

log.debug("Determined the offset to be %d", offset)

# optimise for offset == 0, which in case of inplace updating is a noop

if offset is not None and offset != 0:

fu.seek(tag.offset + 4, os.SEEK\_SET)

fu.write(make\_si32\_extended(tag.timestamp - offset))

def retimestamp\_file\_inplace(inpath):

try:

f = open(inpath, 'rb')

fu = open(inpath, 'rb+')

except IOError, (errno, strerror):

log.error("Failed to open `%s': %s", inpath, strerror)

return False

try:

retimestamp\_tags\_inplace(f, fu)

except IOError, (errno, strerror):

log.error("Failed to create the retimestamped file: %s", strerror)

return False

except MalformedFLV, e:

message = e[0] % e[1:]

log.error("The file `%s' is not a valid FLV file: %s", inpath, message)

return False

except EndOfFile:

log.error("Unexpected end of file on file `%s'", inpath)

return False

f.close()

fu.close()

return True

def retimestamp\_file\_atomically(inpath, outpath):

try:

f = open(inpath, 'rb')

except IOError, (errno, strerror):

log.error("Failed to open `%s': %s", inpath, strerror)

return False

if outpath:

try:

fo = open(outpath, 'w+b')

except IOError, (errno, strerror):

log.error("Failed to open `%s': %s", outpath, strerror)

return False

else:

try:

fd, temppath = tempfile.mkstemp()

# preserve the permission bits

shutil.copymode(inpath, temppath)

fo = os.fdopen(fd, 'wb')

except EnvironmentError, (errno, strerror):

log.error("Failed to create temporary file: %s", strerror)

return False

try:

shutil.copyfileobj(f, fo)

except EnvironmentError, (errno, strerror):

log.error("Failed to create temporary copy: %s", strerror)

force\_remove(temppath)

return False

f.seek(0)

fo.seek(0)

try:

retimestamp\_tags\_inplace(f, fo)

except IOError, (errno, strerror):

log.error("Failed to create the retimestamped file: %s", strerror)

if not outpath:

force\_remove(temppath)

return False

except MalformedFLV, e:

message = e[0] % e[1:]

log.error("The file `%s' is not a valid FLV file: %s", inpath, message)

if not outpath:

force\_remove(temppath)

return False

except EndOfFile:

log.error("Unexpected end of file on file `%s'", inpath)

if not outpath:

force\_remove(temppath)

return False

f.close()

fo.close()

if not outpath:

# If we were not writing directly to the output file

# we need to overwrite the original

try:

shutil.move(temppath, inpath)

except EnvironmentError, (errno, strerror):

log.error("Failed to overwrite the original file "

"with the indexed version: %s", strerror)

return False

return True

def retimestamp\_file(inpath, outpath=None, inplace=False):

out\_text = (outpath and ("into file `%s'" % outpath)) or "and overwriting"

log.debug("Retimestamping file `%s' %s", inpath, out\_text)

if inplace:

log.debug("Operating in inplace mode")

return retimestamp\_file\_inplace(inpath)

else:

log.debug("Not operating in inplace mode, using temporary files")

return retimestamp\_file\_atomically(inpath, outpath)

def process\_options():

usage = "%prog [-i] [-U] file [outfile|file2 file3 ...]"

description = (

"""Rewrites timestamps in FLV files making by the first media tag timestamped

with 0. The rest of the tags is retimestamped relatively. With the -i

(inplace) option modifies the files without creating temporary copies. With

the -U (update) option operates on all parameters, updating the files in

place. Without the -U option accepts one input and one output file path.

""")

version = "%%prog flvlib %s" % \_\_versionstr\_\_

parser = OptionParser(usage=usage, description=description,

version=version)

parser.add\_option("-i", "--inplace", action="store\_true",

help=("inplace mode, does not create temporary files, but "

"risks corruption in case of errors"))

parser.add\_option("-U", "--update", action="store\_true",

help=("update mode, overwrites the given files "

"instead of writing to outfile"))

parser.add\_option("-v", "--verbose", action="count",

default=0, dest="verbosity",

help="be more verbose, each -v increases verbosity")

options, args = parser.parse\_args(sys.argv)

if len(args) < 2:

parser.error("You have to provide at least one file path")

if not options.update and options.inplace:

parser.error("You need to use the update mode if you are updating "

"files in place")

if not options.update and len(args) != 3:

parser.error("You need to provide one infile and one outfile "

"when not using the update mode")

if options.verbosity > 3:

options.verbosity = 3

log.setLevel({0: logging.ERROR, 1: logging.WARNING,

2: logging.INFO, 3: logging.DEBUG}[options.verbosity])

return options, args

def retimestamp\_files():

options, args = process\_options()

clean\_run = True

if not options.update:

clean\_run = retimestamp\_file(args[1], args[2])

else:

for filename in args[1:]:

if not retimestamp\_file(filename, inplace=options.inplace):

clean\_run = False

return clean\_run

def main():

try:

outcome = retimestamp\_files()

except KeyboardInterrupt:

# give the right exit status, 128 + signal number

# signal.SIGINT = 2

sys.exit(128 + 2)

except EnvironmentError, (errno, strerror):

try:

print >>sys.stderr, strerror

except StandardError:

pass

sys.exit(2)

if outcome:

sys.exit(0)

else:

sys.exit(1)