import os

import re

import sys

import math

import signal

import socket

import timeit

import platform

import threading

import uservar

import xbmc

import xbmcaddon

import xbmcgui

import xbmcplugin

import os

import sys

ADDONTITLE = uservar.ADDONTITLE

COLOR1 = uservar.COLOR1

COLOR2 = uservar.COLOR2

\_\_version\_\_ = '0.3.5'

user\_agent = None

source = None

shutdown\_event = None

scheme = 'http'

socket\_socket = socket.socket

try:

import xml.etree.cElementTree as ET

except ImportError:

try:

import xml.etree.ElementTree as ET

except ImportError:

from xml.dom import minidom as DOM

ET = None

try:

import xml.etree.cElementTree as ET

from xml.dom import minidom as DOM

except ImportError:

try:

import xml.etree.ElementTree as ET

except ImportError:

from xml.dom import minidom as DOM

ET = None

try:

from urllib2 import urlopen, Request, HTTPError, URLError

except ImportError:

from urllib.request import urlopen, Request, HTTPError, URLError

try:

from httplib import HTTPConnection, HTTPSConnection

except ImportError:

e\_http\_py2 = sys.exc\_info()

try:

from http.client import HTTPConnection, HTTPSConnection

except ImportError:

e\_http\_py3 = sys.exc\_info()

raise SystemExit('''Your python installation is missing required HTTP client classes:

Python 2: %s

Python 3: %s'''

% (e\_http\_py2[1], e\_http\_py3[1]))

try:

from Queue import Queue

except ImportError:

from queue import Queue

try:

from urlparse import urlparse

except ImportError:

from urllib.parse import urlparse

try:

from urlparse import parse\_qs

except ImportError:

try:

from urllib.parse import parse\_qs

except ImportError:

from cgi import parse\_qs

try:

from hashlib import md5

except ImportError:

from md5 import md5

try:

from argparse import ArgumentParser as ArgParser

except ImportError:

from optparse import OptionParser as ArgParser

try:

import builtins

except ImportError:

def print\_(\*args, \*\*kwargs):

fp = kwargs.pop('file', sys.stdout)

if fp is None:

return

def write(data):

if not isinstance(data, basestring):

data = str(data)

fp.write(data)

want\_unicode = False

sep = kwargs.pop('sep', None)

if sep is not None:

if isinstance(sep, unicode):

want\_unicode = True

elif not isinstance(sep, str):

raise TypeError('sep must be None or a string')

end = kwargs.pop('end', None)

if end is not None:

if isinstance(end, unicode):

want\_unicode = True

elif not isinstance(end, str):

raise TypeError('end must be None or a string')

if kwargs:

raise TypeError('invalid keyword arguments to print()')

if not want\_unicode:

for arg in args:

if isinstance(arg, unicode):

want\_unicode = True

break

if want\_unicode:

newline = unicode('\n')

space = unicode(' ')

else:

newline = '\n'

space = ' '

if sep is None:

sep = space

if end is None:

end = newline

for (i, arg) in enumerate(args):

if i:

write(sep)

write(arg)

write(end)

else:

print\_ = getattr(builtins, 'print')

del builtins

class SpeedtestCliServerListError(Exception):

"""

"""

def bound\_socket(\*args, \*\*kwargs):

global source

sock = socket\_socket(\*args, \*\*kwargs)

sock.bind((source, 0))

return sock

def distance(origin, destination):

(lat1, lon1) = origin

(lat2, lon2) = destination

radius = 6371 # km

dlat = math.radians(lat2 - lat1)

dlon = math.radians(lon2 - lon1)

a = math.sin(dlat / 2) \* math.sin(dlat / 2) \

+ math.cos(math.radians(lat1)) \* math.cos(math.radians(lat2)) \

\* math.sin(dlon / 2) \* math.sin(dlon / 2)

c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))

d = radius \* c

return d

def build\_user\_agent():

global user\_agent

if user\_agent:

return user\_agent

ua\_tuple = ('Mozilla/5.0', '(%s; U; %s; en-us)'

% (platform.system(), platform.architecture()[0]),

'Python/%s' % platform.python\_version(),

'(KHTML, like Gecko)', 'speedtest-cli/%s' % \_\_version\_\_)

user\_agent = ' '.join(ua\_tuple)

return user\_agent

def build\_request(url, data=None, headers={}):

if url[0] == ':':

schemed\_url = '%s%s' % (scheme, url)

else:

schemed\_url = url

headers['User-Agent'] = user\_agent

return Request(schemed\_url, data=data, headers=headers)

def catch\_request(request):

try:

uh = urlopen(request)

return uh

except (HTTPError, URLError, socket.error):

e = sys.exc\_info()[1]

return (None, e)

class FileGetter(threading.Thread):

def \_\_init\_\_(self, url, start):

self.url = url

self.result = None

self.starttime = start

threading.Thread.\_\_init\_\_(self)

def run(self):

self.result = [0]

try:

if timeit.default\_timer() - self.starttime <= 10:

request = build\_request(self.url)

f = urlopen(request)

while 1 and not shutdown\_event.isSet():

self.result.append(len(f.read(10240)))

if self.result[-1] == 0:

break

f.close()

except IOError:

pass

def downloadSpeed(files, quiet=False):

start = timeit.default\_timer()

def producer(q, files):

for file in files:

thread = FileGetter(file, start)

thread.start()

q.put(thread, True)

if not quiet and not shutdown\_event.isSet():

sys.stdout.write('.')

sys.stdout.flush()

finished = []

def consumer(q, total\_files):

while len(finished) < total\_files:

thread = q.get(True)

while thread.isAlive():

thread.join(timeout=0.1)

finished.append(sum(thread.result))

del thread

q = Queue(6)

prod\_thread = threading.Thread(target=producer, args=(q, files))

cons\_thread = threading.Thread(target=consumer, args=(q,

len(files)))

start = timeit.default\_timer()

prod\_thread.start()

cons\_thread.start()

while prod\_thread.isAlive():

prod\_thread.join(timeout=0.1)

while cons\_thread.isAlive():

cons\_thread.join(timeout=0.1)

return sum(finished) / (timeit.default\_timer() - start)

class FilePutter(threading.Thread):

def \_\_init\_\_(

self,

url,

start,

size,

):

self.url = url

chars = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'

data = chars \* int(round(int(size) / 36.0))

self.data = ('content1=%s' % data[0:int(size) - 9]).encode()

del data

self.result = None

self.starttime = start

threading.Thread.\_\_init\_\_(self)

def run(self):

try:

if timeit.default\_timer() - self.starttime <= 10 \

and not shutdown\_event.isSet():

request = build\_request(self.url, data=self.data)

f = urlopen(request)

f.read(11)

f.close()

self.result = len(self.data)

else:

self.result = 0

except IOError:

self.result = 0

def uploadSpeed(url, sizes, quiet=False):

start = timeit.default\_timer()

def producer(q, sizes):

for size in sizes:

thread = FilePutter(url, start, size)

thread.start()

q.put(thread, True)

if not quiet and not shutdown\_event.isSet():

sys.stdout.write('.')

sys.stdout.flush()

finished = []

def consumer(q, total\_sizes):

while len(finished) < total\_sizes:

thread = q.get(True)

while thread.isAlive():

thread.join(timeout=0.1)

finished.append(thread.result)

del thread

q = Queue(6)

prod\_thread = threading.Thread(target=producer, args=(q, sizes))

cons\_thread = threading.Thread(target=consumer, args=(q,

len(sizes)))

start = timeit.default\_timer()

prod\_thread.start()

cons\_thread.start()

while prod\_thread.isAlive():

prod\_thread.join(timeout=0.1)

while cons\_thread.isAlive():

cons\_thread.join(timeout=0.1)

return sum(finished) / (timeit.default\_timer() - start)

def getAttributesByTagName(dom, tagName):

elem = dom.getElementsByTagName(tagName)[0]

return dict(list(elem.attributes.items()))

def getConfig():

request = \

build\_request('http://www.speedtest.net/speedtest-config.php')

uh = catch\_request(request)

if uh is False:

print\_('Could not retrieve speedtest.net configuration: %s' % e)

sys.exit(1)

configxml = []

while 1:

configxml.append(uh.read(10240))

if len(configxml[-1]) == 0:

break

if int(uh.code) != 200:

return None

uh.close()

try:

try:

root = ET.fromstring(''.encode().join(configxml))

config = {

'client': root.find('client').attrib,

'times': root.find('times').attrib,

'download': root.find('download').attrib,

'upload': root.find('upload').attrib,

}

except Exception:

# Python3 branch

root = DOM.parseString(''.join(configxml))

config = {

'client': getAttributesByTagName(root, 'client'),

'times': getAttributesByTagName(root, 'times'),

'download': getAttributesByTagName(root, 'download'),

'upload': getAttributesByTagName(root, 'upload'),

}

except SyntaxError:

print\_('Failed to parse speedtest.net configuration')

sys.exit(1)

del root

del configxml

return config

def closestServers(client, all=False):

urls = ['http://www.speedtest.net/speedtest-servers-static.php',

'https://www.speedtest.net/speedtest-servers-static.php']

errors = []

servers = {}

for url in urls:

try:

request = build\_request(url)

uh = catch\_request(request)

if uh is False:

errors.append('%s' % e)

raise SpeedtestCliServerListError

serversxml = []

while 1:

serversxml.append(uh.read(10240))

if len(serversxml[-1]) == 0:

break

if int(uh.code) != 200:

uh.close()

raise SpeedtestCliServerListError

uh.close()

try:

try:

root = ET.fromstring(''.encode().join(serversxml))

elements = root.getiterator('server')

except Exception:

# Python3 branch

root = DOM.parseString(''.join(serversxml))

elements = root.getElementsByTagName('server')

except SyntaxError:

raise SpeedtestCliServerListError

for server in elements:

try:

attrib = server.attrib

except AttributeError:

attrib = dict(list(server.attributes.items()))

d = distance([float(client['lat']), float(client['lon'

])], [float(attrib.get('lat')),

float(attrib.get('lon'))])

attrib['d'] = d

if d not in servers:

servers[d] = [attrib]

else:

servers[d].append(attrib)

del root

del serversxml

del elements

except SpeedtestCliServerListError:

continue

if servers:

break

if not servers:

print\_('''Failed to retrieve list of speedtest.net servers:

%s'''

% '\n'.join(errors))

sys.exit(1)

closest = []

for d in sorted(servers.keys()):

for s in servers[d]:

closest.append(s)

if len(closest) == 5 and not all:

break

else:

continue

break

del servers

return closest

def getBestServer(servers):

results = {}

for server in servers:

cum = []

url = '%s/latency.txt' % os.path.dirname(server['url'])

urlparts = urlparse(url)

for i in range(0, 3):

try:

if urlparts[0] == 'https':

h = HTTPSConnection(urlparts[1])

else:

h = HTTPConnection(urlparts[1])

headers = {'User-Agent': user\_agent}

start = timeit.default\_timer()

h.request('GET', urlparts[2], headers=headers)

r = h.getresponse()

total = timeit.default\_timer() - start

except (HTTPError, URLError, socket.error):

cum.append(3600)

continue

text = r.read(9)

if int(r.status) == 200 and text == 'test=test'.encode():

cum.append(total)

else:

cum.append(3600)

h.close()

avg = round(sum(cum) / 6 \* 1000, 3)

results[avg] = server

fastest = sorted(results.keys())[0]

best = results[fastest]

best['latency'] = fastest

return best

def ctrl\_c(signum, frame):

global shutdown\_event

shutdown\_event.set()

raise SystemExit('\nCancelling...')

def version():

raise SystemExit(\_\_version\_\_)

def speedtest(

list=False,

mini=None,

server=None,

share=True,

simple=False,

src=None,

timeout=10,

units=('bit', 8),

version=False,

):

global shutdown\_event, source, scheme

shutdown\_event = threading.Event()

global line1, line2, line3

dp = xbmcgui.DialogProgress()

line1 = '[COLOR %s]Starting test..[/COLOR]' % COLOR2

dp.create('%s: [COLOR %s]Speed Test[/COLOR]' % (ADDONTITLE,

COLOR1), line1)

dp.update(0)

build\_user\_agent()

print\_('Retrieving speedtest.net configuration...')

line2 = \

'[COLOR %s]Retrieving speedtest.net configuration...[/COLOR]' \

% COLOR2

dp.update(2, line1, line2)

try:

config = getConfig()

except URLError:

print\_('Cannot retrieve speedtest configuration')

sys.exit(1)

print\_('Retrieving speedtest.net server list...')

line3 = '[COLOR %s]Retrieving speedtest.net server list...[/COLOR]' \

% COLOR2

dp.update(4, line1, line2, line3)

servers = closestServers(config['client'])

print\_('Testing from %(isp)s (%(ip)s)...' % config['client'])

line1 = '[COLOR ' + COLOR2 + ']Testing From:[/COLOR] [COLOR ' \

+ COLOR1 + ']%(isp)s (%(ip)s)[/COLOR]' % config['client']

dp.update(6, line1)

print\_('Selecting best server based on latency...')

line2 = \

'[COLOR %s]Selecting best server based on latency...[/COLOR]' \

% COLOR2

dp.update(8, '', line2)

best = getBestServer(servers)

print\_(('Hosted by %(sponsor)s (%(name)s) [%(d)0.2f km]: %(latency)s ms'

% best).encode('utf-8', 'ignore'))

line2 = ('[COLOR ' + COLOR2

+ ']Server location: %(name)s [%(d)0.2f km]: %(latency)s ms[/COLOR]'

% best).encode('utf-8', 'ignore')

dp.update(10, '', line2)

sizes = [

350,

500,

750,

1000,

1500,

2000,

2500,

3000,

3500,

4000,

]

urls = []

for size in sizes:

for i in range(0, 4):

urls.append('%s/random%sx%s.jpg'

% (os.path.dirname(best['url']), size, size))

print\_('Testing download speed', end='')

line3 = '[COLOR %s]Testing download speed...[/COLOR]' % COLOR2

dp.update(15, '', '', line3)

dlspeed = downloadSpeed(urls, simple)

print\_()

print\_('Download: %0.2f M%s/s' % (dlspeed / 1000 / 1000 \* units[1],

units[0]))

sizesizes = [int(.25 \* 1000 \* 1000), int(.5 \* 1000 \* 1000)]

sizes = []

for size in sizesizes:

for i in range(0, 25):

sizes.append(size)

print\_('[COLOR red]Testing upload speed[/COLOR]', end='')

line2 = \

'[COLOR %s]Testing download speed:[/COLOR] [COLOR %s]%0.2f M%s/s[/COLOR]' \

% (COLOR2, COLOR1, dlspeed / 1000 / 1000 \* units[1], units[0])

line3 = '[COLOR %s]Testing upload speed...[/COLOR]' % COLOR2

dp.update(65, '', line2, line3)

ulspeed = uploadSpeed(best['url'], sizes, simple)

print\_()

print\_('Upload: %0.2f M%s/s' % (ulspeed / 1000 / 1000 \* units[1],

units[0]))

i = 2

while ulspeed < 1:

dp.update(65, '', '', '[COLOR ' + COLOR2

+ ']Testing upload speed... [Attempt [/COLOR]'

+ str(i) + ']')

ulspeed = uploadSpeed(best['url'], sizes, simple)

print\_()

print\_('Upload: %0.2f M%s/s' % (ulspeed / 1000 / 1000

\* units[1], units[0]))

i = i + 1

if i == 6:

return uploadfail

line1 = line2

line2 = \

'[COLOR %s]Testing upload speed:[/COLOR] [COLOR %s]%0.2f M%s/s[/COLOR]' \

% (COLOR2, COLOR1, ulspeed / 1000 / 1000 \* units[1], units[0])

line3 = '[COLOR %s]Getting results...[/COLOR]' % COLOR2

dp.update(95, line1, line2, line3)

if share:

dlspeedk = int(round(dlspeed / 1000 \* 8, 0))

ping = int(round(best['latency'], 0))

ulspeedk = int(round(ulspeed / 1000 \* 8, 0))

apiData = [

'download=%s' % dlspeedk,

'ping=%s' % ping,

'upload=%s' % ulspeedk,

'promo=',

'startmode=%s' % 'pingselect',

'recommendedserverid=%s' % best['id'],

'accuracy=%s' % 1,

'serverid=%s' % best['id'],

'hash=%s' % md5(('%s-%s-%s-%s' % (ping, ulspeedk, dlspeedk,

'297aae72')).encode()).hexdigest(),

]

headers = \

{'Referer': 'http://c.speedtest.net/flash/speedtest.swf'}

request = build\_request('http://www.speedtest.net/api/api.php',

data='&'.join(apiData).encode(),

headers=headers)

f = catch\_request(request)

if f is False:

print\_('Could not submit results to speedtest.net: %s' % e)

sys.exit(1)

response = f.read()

code = f.code

f.close()

if int(code) != 200:

print\_('Could not submit results to speedtest.net')

sys.exit(1)

qsargs = parse\_qs(response.decode())

resultid = qsargs.get('resultid')

if not resultid or len(resultid) != 1:

print\_('Could not submit results to speedtest.net')

sys.exit(1)

print\_('Share results: %s://www.speedtest.net/result/%s.png'

% (scheme, resultid[0]))

dp.close

curserver = ('%(name)s [%(d)0.2f km]: %(latency)s ms'

% best).encode('utf-8', 'ignore')

return (

'%s://www.speedtest.net/result/%s.png' % (scheme,

resultid[0]),

dlspeed / 1000 / 1000 \* units[1],

units[0],

ulspeed / 1000 / 1000 \* units[1],

units[0],

ping,

curserver,

)

def main():

try:

speedtest()

except KeyboardInterrupt:

print\_('\nCancelling...')

dp.close()

sys.exit()

if \_\_name\_\_ == '\_\_main\_\_':

main()