#!/usr/bin/env python

#

# grabserial - program to read a serial port and send the data to stdout

#

# Copyright 2006,2016 Sony Corporation

#

# This program is provided under the Gnu General Public License (GPL)

# version 2 ONLY. This program is distributed WITHOUT ANY WARRANTY.

# See the LICENSE file, which should have accompanied this program,

# for the text of the license.

#

# 2016-05-10 by Tim Bird <tim.bird@am.sony.com>

# 2006-09-07 by Tim Bird

#

# To do:

# \* buffer output chars??

#

# CHANGELOG:

# 2016.08.30 - Version 1.9.3 - allow forcing the baudrate with -B

# 2016.07.01 - Version 1.9.2 - change how version is stored

# 2016.05.10 - Version 1.9.1 - allow skipping the tty check with -S

# 2016.05.10 - Version 1.9.0 - support use as a python module

# Note that the main module routine will be grabserial.grab(args,[outputfd])

# where args is a list of command-line-style args

# as they would be passed using the standalone program. e.g.

# grabserial.grab(None, ["-d", "/dev/ttyUSB0", "-v"])

# output from the serial port (with timing data) is sent to outputfd

# 2015.04.23 - Version 1.8.1 - remove instructions for applying LICENSE text

# to new files, and add no-warranty language to grabserial.

# 2015.03.10 - Version 1.8.0 - add -o option for saving output to a file

# add -T option for absolute times. Both contributed by ramaxlo

# 2015.03.10 - Version 1.7.1 - add line feed to instantpat result line

# 2014.09.28 - Version 1.7.0 - add option for force reset for USB serial

# contributed by John Mehaffey <mehaf@gedanken.com>

# 2014.01.07 - Version 1.6.0 - add option for exiting based on a

# mid-line pattern (quitpat). Simeon Miteff <simeon.miteff@gmail.com>

# 2013.12.19 - Version 1.5.2 - verify Windows ports w/ serial.tools.list\_ports

# (thanks to Yegor Yefromov for the idea and code)

# 2013.12.16 - Version 1.5.1 - Change my e-mail address

# 2011.12.19 - Version 1.5.0 - add options for mid-line time capture

# (instantpat) and base time from launch of program instead of

# first char seen (launchtime) - contributed by Kent Borg

# 2011-09-24 - better time output and time delta

# Constantine Shulyupin <const@makelinux.com>

# 2008-06-02 - Version 1.1.0 add support for sending a command to

# the serial port before grabbing output

VERSION=(1,9,3)

import os, sys

import getopt

import serial

import time

import re

verbose = 0

def vprint(message):

if verbose:

print(message)

def usage(rcode):

cmd = "grabserial"

print("""%s : Serial line reader

Usage: %s [options] <config\_file>

options:

-h, --help Print this message

-d, --device=<devpath> Set the device to read (default '/dev/ttyS0')

-b, --baudrate=<val> Set the baudrate (default 115200)

-B <val> Force the baudrate to the indicated value

(grabserial won't check that the baudrate is legal)

-w, --width=<val> Set the data bit width (default 8)

-p, --parity=<val> Set the parity (default N)

-s, --stopbits=<val> Set the stopbits (default 1)

-x, --xonxoff Enable software flow control (default off)

-r, --rtscts Enable RTS/CTS flow control (default off)

-f, --force-reset Force pyserial to reset device parameters

-e, --endtime=<secs> End the program after the specified seconds have

elapsed.

-c, --command=<cmd> Send a command to the port before reading

-t, --time Print time for each line received. The time is

when the first character of each line is

received by %s

-T, --systime Print system time for each line received. The time

is the absolute local time when the first character

of each line is received by %s

-m, --match=<pat> Specify a regular expression pattern to match to

set a base time. Time values for lines after the

line matching the pattern will be relative to

this base time.

-i, --instantpat=<pat> Specify a regular expression pattern to have its time

reported at end of run. Works mid-line.

-q, --quitpat=<pat> Specify a regular expression pattern to end the

program. Works mid-line.

-l, --launchtime Set base time from launch of program.

-o, --output=<name> Output data to the named file.

-v, --verbose Show verbose runtime messages

-V, --version Show version number and exit

-S, --skip Skip sanity checking of the serial device.

May be needed for some devices.

-P, --acrn\_pwd=<pwd> Acrn pwassword for test

-u, --acrn\_user=<user> Acrn username for test

Ex: %s -e 30 -t -m "^Linux version.\*"

This will grab serial input for 30 seconds, displaying the time for

each line, and re-setting the base time when the line starting with

"Linux version" is seen.

""" % (cmd, cmd, cmd, cmd, cmd))

sys.exit(rcode)

def device\_exists(device):

try:

from serial.tools import list\_ports

for port in list\_ports.comports():

if port[0] == device:

return True

return False

except:

return os.path.exists(device)

# grab - main routine to grab a serial port and time the output of each line

# takes a list of arguments, as they would have been passed in sys.argv

# that is, a list of strings.

# also can take an optional file descriptor for where to send the data

# by default, data read from the serial port is sent to sys.stdout, but

# you can specify your own (already open) file descriptor, or None. This

# would only make sense if you specified another outputfile with

# "-o","myoutputfile"

def grab(arglist, outputfd=sys.stdout):

global verbose

# parse the command line options

try:

opts, args = getopt.getopt(arglist,

"hli:d:b:B:w:p:s:xrfc:tTm:e:o:vVq:SP:u:", [

"help",

"launchtime",

"instantpat=",

"device=",

"baudrate=",

"width=",

"parity=",

"stopbits=",

"xonxoff",

"rtscts",

"force-reset",

"command=",

"time",

"systime",

"match=",

"endtime=",

"output=",

"verbose",

"version",

"quitpat=",

"skip",

"acrn\_pwd=",

"acrn\_user="])

except:

# print help info and exit

print("Error parsing command line options")

usage(2)

sd = serial.Serial()

sd.port="/dev/ttyS0"

sd.baudrate=115200

sd.bytesize=serial.EIGHTBITS

sd.parity=serial.PARITY\_NONE

sd.stopbits=serial.STOPBITS\_ONE

sd.xonxoff=False

sd.rtscts=False

sd.dsrdtr=False

# specify a read timeout of 1 second

sd.timeout=1

force = False

show\_time = 0

show\_systime = 0

basepat = ""

instantpat = ''

quitpat = ''

basetime = 0

instanttime = None

endtime = 0

outputfile = None

command = ""

skip\_device\_check = 0

acrn\_user = ''

acrn\_pwd = ''

for opt, arg in opts:

if opt in ["-h", "--help"]:

usage(0)

if opt in ["-d", "--device"]:

device = arg

if not skip\_device\_check and not device\_exists(device):

print("Error: serial device '%s' does not exist, aborting." % device)

print(" If you think this port really exists, then try using the -S option")

print(" to skip the serial device check. (put it before the -d argument)")

sd.close()

usage(2)

sd.port = device

if opt in ["-b", "--baudrate"]:

baud = int(arg)

if baud not in sd.BAUDRATES:

print("Error: invalid baud rate '%d' specified" % baud)

print("Valid baud rates are: %s" % str(sd.BAUDRATES))

print("You can force the baud rate using the -B option")

sd.close()

sys.exit(3)

sd.baudrate = baud

if opt == "-B":

sd.baudrate = int(arg)

if opt in ["-p", "--parity"]:

par = arg.upper()

if par not in sd.PARITIES:

print("Error: invalid parity '%s' specified" % par)

print("Valid parities are: %s" % str(sd.PARITIES))

sd.close()

sys.exit(3)

sd.parity = par

if opt in ["-w", "--width"]:

width = int(arg)

if width not in sd.BYTESIZES:

print("Error: invalid data bit width '%d' specified" % width)

print("Valid data bit widths are: %s" % str(sd.BYTESIZES))

sd.close()

sys.exit(3)

sd.bytesize = width

if opt in ["-s", "--stopbits"]:

stop = int(arg)

if stop not in sd.STOPBITS:

print("Error: invalid stopbits '%d' specified" % stop)

print("Valid stopbits are: %s" % str(sd.STOPBITS))

sd.close()

sys.exit(3)

sd.stopbits = stop

if opt in ["-c", "--command"]:

command = arg

if opt in ["-x", "--xonxoff"]:

sd.xonxoff = True

if opt in ["-r", "--rtscts"]:

sd.rtscts = True

if opt in ["-f", "--force-set"]:

force = True

if opt in ["-t", "--time"]:

show\_time=1

show\_systime=0

if opt in ["-T", "--systime"]:

show\_time=0

show\_systime=1

if opt in ["-m", "--match"]:

basepat=arg

if opt in ["-i", "--instantpat"]:

instantpat=arg

if opt in ["-q", "--quitpat"]:

quitpat=arg

if opt in ["-l", "--launchtime"]:

print('setting basetime to time of program launch')

basetime = time.time()

if opt in ["-e", "--endtime"]:

endstr=arg

try:

endtime = time.time()+float(endstr)

except:

print("Error: invalid endtime %s specified" % arg)

sd.close()

sys.exit(3)

if opt in ["-o", "--output"]:

outputfile = arg

if opt in ["-v", "--verbose"]:

verbose=1

if opt in ["-V", "--version"]:

print("grabserial version %d.%d.%d" % VERSION)

sd.close()

sys.exit(0)

if opt in ["-S"]:

skip\_device\_check=1

if opt in ["-P", "--acrn\_pwd="]:

acrn\_pwd=arg

if opt in ["-u", "--acrn\_user="]:

acrn\_user = arg

# if verbose, show what our settings are

vprint("Opening serial port %s" % sd.port)

vprint("%d:%d%s%s:xonxoff=%d:rtscts=%d" % (sd.baudrate, sd.bytesize,

sd.parity, sd.stopbits, sd.xonxoff, sd.rtscts))

if endtime:

vprint("Program will end in %s seconds" % endstr)

if show\_time:

vprint("Printing timing information for each line")

if show\_systime:

vprint("Printing absolute timing information for each line")

if basepat:

vprint("Matching pattern '%s' to set base time" % basepat)

if instantpat:

vprint("Instant pattern '%s' to set base time" % instantpat)

if quitpat:

vprint("Instant pattern '%s' to exit program" % quitpat)

if skip\_device\_check:

vprint("Skipping check of serial device")

if outputfile:

try:

out = open(outputfile, "wb")

except IOError:

print("Can't open output file '%s'" % outputfile)

sys.exit(1)

vprint("Saving data to '%s'" % outputfile)

prev1 = 0

linetime = 0

newline = 1

curline = ""

vprint("Use Control-C to stop...")

if force:

# pyserial does not reconfigure the device if the settings

# don't change from the previous ones. This causes issues

# with (at least) some USB serial converters

toggle = sd.xonxoff

sd.xonxoff = not toggle

sd.open()

sd.close()

sd.xonxoff = toggle

sd.open()

sd.flushInput()

sd.flushOutput()

if command:

sd.write(command + "\n")

sd.flush()

# read from the serial port until something stops the program

isAcrnWrite = False

isCrashWrite = False

isLaunch = False

isPwd = False

line = ""

while(1):

try:

# read for up to 1 second

x = sd.read()

line += x

# see if we're supposed to stop yet

if endtime and time.time()>endtime:

break

# if we didn't read anything, loop

if len(x)==0:

continue

# ignore carriage returns

if x=="\r":

line = ""

continue

if line.find("ACRN:") != -1 and not isAcrnWrite:

isAcrnWrite = True

sd.write('sos\_console 0\r'.encode())

if line.find("login:") != -1 and not isCrashWrite:

isCrashWrite = True

sd.write('{0}\r'.encode().format(acrn\_user))

if line.find("Password:") != -1 and not isPwd:

isPwd = True

sd.write('{0}\r'.encode().format(acrn\_pwd))

sd.write('sh ./launch\_UOS.sh -V 2\r'.encode())

if line.find("sh: ./launch\_UOS.sh: No such file or directory") != -1 and not isLaunch:

isLaunch = True

sd.write('sh /usr/share/acrn/samples/apl-mrb/launch\_uos.sh -V 2\r'.encode())

# set basetime to when first char is received

if not basetime:

basetime = time.time()

if show\_time and newline:

linetime = time.time()

elapsed = linetime-basetime

delta = elapsed-prev1

msg ="[%4.6f %2.6f] " % (elapsed, delta)

if outputfd:

outputfd.write(msg)

if outputfile:

out.write(msg)

prev1 = elapsed

newline = 0

if show\_systime and newline:

linetime = time.time()

linetimestr = time.strftime(

'%y-%m-%d %H:%M:%S',

time.localtime(linetime))

elapsed = linetime-basetime

delta = elapsed-prev1

msg = "[%s %2.6f] " % (linetimestr, delta)

sys.stdout.write(msg)

if outputfile:

out.write(msg)

prev1 = elapsed

newline = 0

# FIXTHIS - should I buffer the output here??

sys.stdout.write(x)

if outputfile:

out.write(x)

curline += x

# watch for patterns

if instantpat and not instanttime and \

re.search(instantpat, curline):

# instantpat in curline:

instanttime = time.time()

# Exit the loop if quitpat matches

if quitpat and re.search(quitpat, curline):

break

if x=="\n":

newline = 1

if basepat and re.match(basepat, curline):

basetime = linetime

elapsed = 0

prev1 = 0

curline = ""

sys.stdout.flush()

if outputfile:

out.flush()

except:

break

sd.close()

if instanttime:

instanttime\_str = '%4.6f' % (instanttime-basetime)

msg = '\nThe instantpat: "%s" was matched at %s\n' % \

(instantpat, instanttime\_str)

sys.stdout.write(msg)

sys.stdout.flush()

if outputfile:

out.write(msg)

out.flush()

if outputfile:

out.close()

if \_\_name\_\_=="\_\_main\_\_":

grab(sys.argv[1:])