Redirect command line results to a tkinter GUI

Solution:

You could create a script wrapper that runs your command line program as a sub process, then add the output to something like a text widget.

```
from tkinter import *
import subprocess as sub
p = sub.Popen('./script',stdout=sub.PIPE,stderr=sub.PIPE)
output, errors = p.communicate()

root = Tk()
text = Text(root)
text.pack()
text.insert(END, output)
root.mainloop()
```

where script is your program. You can obviously print the errors in a different colour, or something like that.

To display subprocess' output in a GUI *while it is still running*, a portable stdlib-only solution that works on both Python 2 and 3 has to use a background thread:


```
#!/usr/bin/python
"""
- read output from a subprocess in a background thread
- show the output in the GUI
```

```
"""
import sys
from itertools import islice
from subprocess import Popen, PIPE
from textwrap import dedent
from threading import Thread

try:
    import Tkinter as tk
    from Queue import Queue, Empty
except ImportError:
    import tkinter as tk # Python 3
    from queue import Queue, Empty # Python 3

def iter_except(function, exception):
    """Works like builtin 2-argument `iter()`, but stops on `exception`."""
    try:
        while True:
            yield function()
    except exception:
        return

class DisplaySubprocessOutputDemo:
    def __init__(self, root):
        self.root = root

        # start dummy subprocess to generate some output
        self.process = Popen([sys.executable, "-u", "-c", dedent("""
            import itertools, time

            for i in itertools.count():
                print("%d.%d" % divmod(i, 10))
                time.sleep(0.1)
            """)], stdout=PIPE)

        # launch thread to read the subprocess output
        #   (put the subprocess output into the queue in a background thread,
        #    get output from the queue in the GUI thread.
        #    Output chain: process.readline -> queue -> label)
        q = Queue(maxsize=1024)  # limit output buffering (may stall subprocess)
        t = Thread(target=self.reader_thread, args=[q])
        t.daemon = True # close pipe if GUI process exits
        t.start()

        # show subprocess' stdout in GUI
        self.label = tk.Label(root, text="  ", font=(None, 200))
        self.label.pack(ipadx=4, padx=4, ipady=4, pady=4, fill='both')
        self.update(q) # start update loop

    def reader_thread(self, q):
        """Read subprocess output and put it into the queue."""
        try:
            with self.process.stdout as pipe:
                for line in iter(pipe.readline, b''):
                    q.put(line)
```

```
            finally:
                q.put(None)

        def update(self, q):
            """Update GUI with items from the queue."""
            for line in iter_except(q.get_nowait, Empty): # display all content
                if line is None:
                    self.quit()
                    return
                else:
                    self.label['text'] = line # update GUI
                    break # display no more than one line per 40 milliseconds
            self.root.after(40, self.update, q) # schedule next update

        def quit(self):
            self.process.kill() # exit subprocess if GUI is closed (zombie!)
            self.root.destroy()


    root = tk.Tk()
    app = DisplaySubprocessOutputDemo(root)
    root.protocol("WM_DELETE_WINDOW", app.quit)
    # center window
    root.eval('tk::PlaceWindow %s center' % root.winfo_pathname(root.winfo_id()))
    root.mainloop()
```

The essence of the solution is:

- put the subprocess output into the queue in a background thread
- get the output from the queue in the GUI thread.

i.e., call `process.readline()` in the background thread -> queue -> update GUI label in the main thread. Related `kill-process.py` (no polling -- a less portable solution that uses `event_generate` in a background thread).

Redirecting stdout to a write() method that updates your gui is one way to go, and probably the quickest - although running a subprocess is probably a more elegant solution.

Only redirect stderr once you're really confident it's up and working, though!

Example implimentation (gui file and test script):

test_gui.py:

```
from Tkinter import *
import sys
sys.path.append("/path/to/script/file/directory/")

class App(Frame):
    def run_script(self):
        sys.stdout = self
        ## sys.stderr = self
        try:
            del(sys.modules["test_script"])
        except:
            ## Yeah, it's a real ugly solution...
            pass
        import test_script
        test_script.HelloWorld()
        sys.stdout = sys.__stdout__
        ## sys.stderr = __stderr__

    def build_widgets(self):
        self.text1 = Text(self)
        self.text1.pack(side=TOP)
        self.button = Button(self)
        self.button["text"] = "Trigger script"
        self.button["command"] = self.run_script
        self.button.pack(side=TOP)

    def write(self, txt):
        self.text1.insert(INSERT, txt)

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.build_widgets()
```

```
root = Tk()
app = App(master = root)
app.mainloop()
```

test_script.py:

```
print "Hello world!"

def HelloWorld():
    print "HelloWorldFromDef!"
```

Tags: **Python** / **User Interface** / **Tkinter**

Related

Chemistry - How can I calculate the charge distribution of a water molecule?

AWS Cloud9 Building Docker Image Fail

Installing Shapely on Alpine docker

Best way to run python 3.7 on Ubuntu 16.04 which comes with python 3.5

How to get virtualenv for compiled python (missing pip/easy_install)?

How to connect to AWS ECR using python docker-py

How can I upgrade Python to 2.7.9 on Ubuntu 14.4?

Different Python versions under the same uwsgi Emperor?

PIP not installing to virtualenv directory

Non-responsive apache + mod_wsgi after installing scipy

Recent Posts

The calculation of the entropy of a single atom

PostgreSQL's libpq: Encoding for binary transport of ARRAY[]-data?

When is the infimum of an arbitrary family of measurable functions also measurable?

Checking microphone volume in Javascript

Counting the number of files in a directory using Java