Writing a Threaded Downloader

We'll write a downloader in python using multiple threads to learn more about the threads

WE'LL COVER THE FOLLOWING

The previous example wasn't very useful other than as a tool to explain how threads work. So in this example, we will create a Thread class that can download files from the internet. The U.S. Internal Revenue Service has lots of PDF forms that it has its citizens use for taxes. We will use this free resource for our demo. Here's the code:

```
# Python 2 version

import os
import urllib2

from threading import Thread

class DownloadThread(Thread):
    """
    A threading example that can download a file
    """

    def __init__(self, url, name):
        """Initialize the thread"""
        Thread.__init__(self)
        self.name = name
        self.url = url

    def run(self):
        """Run the thread"""
        handle = urllib2.urlopen(self.url)
        fname = os.path.basename(self.url)
        with open(fname, "wb") as f_handler:
            while True:
                chunk = handle.read(1024)
                if not chunk:
                    break
                f_handler.write(chunk)
        msg = "%s has finished downloading %s!" % (self.name,
```

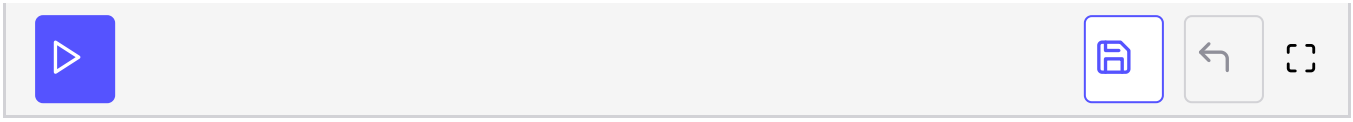
```
        msg = "%s has finished downloading %s!" % (self.name,
                                                   self.url)
        print(msg)

def main(urls):
    """
    Run the program
    """
    for item, url in enumerate(urls):
        name = "Thread %s" % (item+1)
        thread = DownloadThread(url, name)
        thread.start()

if __name__ == "__main__":
    urls = ["http://www.irs.gov/pub/irs-pdf/f1040.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040a.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040ez.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040es.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040sb.pdf"]
    main(urls)
```

This is basically a complete rewrite of the first script. In this one we import the os and urllib2 modules as well as the threading module. We will be using urllib2 to do the actual downloading inside the thread class. The os module is used to extract the name of the file we're downloading so we can use it to create a file with the same name on our machine. In the DownloadThread class, we set up the __init__ to accept a url and a name for the thread. In the run method, we open up the url, extract the filename and then use that filename for naming / creating the file on disk. Then we use a **while** loop to download the file a kilobyte at a time and write it to disk. Once the file is finished saving, we print out the name of the thread and which url has finished downloading.

The Python 3 version of the code is slightly different. You have to import **urllib** instead of **urllib2** and use **urllib.request.urlopen** instead of **urllib2.urlopen**. Here's the code so you can see the difference:

```
# Python 3 version

import os
import urllib.request

from threading import Thread

class DownloadThread(Thread):
    """
    A threading example that can download a file
    """
```

```
    def __init__(self, url, name):
        """Initialize the thread"""
        Thread.__init__(self)
        self.name = name
        self.url = url

    def run(self):
        """Run the thread"""
        handle = urllib.request.urlopen(self.url)
        fname = os.path.basename(self.url)
        with open(fname, "wb") as f_handler:
            while True:
                chunk = handle.read(1024)
                if not chunk:
                    break
                f_handler.write(chunk)
        msg = "%s has finished downloading %s!" % (self.name,
                                                   self.url)
        print(msg)

def main(urls):
    """
    Run the program
    """
    for item, url in enumerate(urls):
        name = "Thread %s" % (item+1)
        thread = DownloadThread(url, name)
        thread.start()

if __name__ == "__main__":
    urls = ["http://www.irs.gov/pub/irs-pdf/f1040.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040a.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040ez.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040es.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040sb.pdf"]
    main(urls)
```



Using Queues

A Queue can be used for first-in-first-out (FIFO) or last-in-last-out (LILO) stack-like implementations if you just use them directly. In this section, we're going to mix threads in and create a simple file downloader script to demonstrate how Queues work for cases where we want concurrency.

To help explain how Queues work, we will rewrite the downloading script from the previous section to use Queues. Let's get started!

```
import os
import threading
import urllib.request
```

```
from queue import Queue


class Downloader(threading.Thread):
    """Threaded File Downloader"""

    def __init__(self, queue):
        """Initialize the thread"""
        threading.Thread.__init__(self)
        self.queue = queue

    def run(self):
        """Run the thread"""
        while True:
            # gets the url from the queue
            url = self.queue.get()

            # download the file
            self.download_file(url)

            # send a signal to the queue that the job is done
            self.queue.task_done()

    def download_file(self, url):
        """Download the file"""
        handle = urllib.request.urlopen(url)
        fname = os.path.basename(url)
        with open(fname, "wb") as f:
            while True:
                chunk = handle.read(1024)
                if not chunk: break
                f.write(chunk)
        msg = "Finished downloading %s!" % (url)
        print(msg)

def main(urls):
    """
    Run the program
    """
    queue = Queue()

    # create a thread pool and give them a queue
    for i in range(5):
        t = Downloader(queue)
        t.setDaemon(True)
        t.start()

    # give the queue some data
    for url in urls:
        queue.put(url)

    # wait for the queue to finish
    queue.join()

if __name__ == "__main__":
    urls = ["http://www.irs.gov/pub/irs-pdf/f1040.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040a.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040ez.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040es.pdf",
            "http://www.irs.gov/pub/irs-pdf/f1040sb.pdf"]
    main(urls)
```

Let's break this down a bit. First of all, we need to look at the main function definition to see how this all flows. Here we see that it accepts a list of urls. The main function then creates a queue instance that it passes to 5 daemonized threads. The main difference between daemonized and non-daemon threads is that you have to keep track of non-daemon threads and close them yourself whereas with a daemon thread you basically just set them and forget them and when your app closes, they close too. Next we load up the queue (using its put method) with the urls we passed in.

Finally we tell the queue to wait for the threads to do their processing via the join method. In the download class, we have the line **self.queue.get()** which blocks until the queue has something to return. That means the threads just sit idly waiting to pick something up. It also means that for a thread to **get** something from the queue, it must call the queue's **get** method. Thus as we add or put items in the queue, the thread pool will pick up or **get** items and process them. This is also known as **dequeing**. Once all the items in the queue are processed, the script ends and exits. On my machine, it downloads all 5 documents in under a second.

Wrapping Up

Now you know how to use threads and queues both in theory and in a practical way. Threads are especially useful when you are creating a user interface and you want to keep your interface usable. Without threads, the user interface would become unresponsive and would appear to hang while you did a large file download or a big query against a database. To keep that from happening, you do the long running processes in threads and then communicate back to your interface when you are done.