Multiprocessing, Multithreading

Embedded Interface Design

with **Bruce Montgomery**

Copyright © 2019 University of Colorado

Learning Objectives

- Students will be able to…
  - Understand the difference in multithreading and multiprocessing
  - Apply the tools present in Python and Node.js to concurrent tasks

Copyright © 2019 University of Colorado

Definitions

- Concurrent – two or more tasks running in overlapping time periods (doesn’t imply they’ll be running at the same instant)
  - For example, multitasking on a single-core machine
- Parallelism – two or more tasks executed simultaneously
- Thread – a sequence of instructions within a process, threads share the same memory space
- Process – an instance of a program with an independent memory space
- Multiprocessing – spawning processes (subprocesses) to take advantage of multiple processors, or CPU cores, on a single machine
- Multithreading – running multiple threads or separate control structures concurrently from a single program
- References [1], [2]

Copyright © 2019 University of Colorado

In Python

- In Python, only use of multiprocessing bypasses the Global Interpreter Lock that insures only one Python bytecode thread is running at any time – however, IO operations in threads release the lock
- So, using multiple threads is appropriate where they manage elements of I/O, network, or user interface transactions that need to be responsive
- Multiprocessing can help your code if it is CPU bound, allowing multiple cores to be exercised by the multiprocessing code
  - Note that using threads for Python CPU bound tasks can actually decrease performance

- Threading – I/O, network, UI bound
- Multiprocessing – CPU bound with multiple cores available

Copyright © 2019 University of Colorado

Python threading library

- threading, the Python thread library, did change a bit in Python 3.x vs. 2.x, mostly in naming methods
- Python threads can be tagged as daemons, which will stop when the main program exits
- You can also join a thread to wait for it to complete before continuing
- Detailed examples in [3] on thread pools, locks, queues, and semiphores

```
import logging
import threading
import time

def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")

    logging.info("Main    : before creating thread")
    x = threading.Thread(target=thread_function, args=(1,))
    logging.info("Main    : before running thread")
    x.start()
    logging.info("Main    : wait for the thread to finish")
    # x.join()
    logging.info("Main    : all done")
```

Copyright © 2019 University of Colorado

Python multiprocessing library

- The Python multiprocessing library provides similar functionality to the threading library for processes
- Processes can be tagged as daemons or can be joined
- Logging library is recommended for debugging concurrent code
- Detailed examples in [4] on process management, using locks, queues, events, managed state, and process pools

```
import multiprocessing
import logging
import sys


def worker():
    print('Doing some work')
    sys.stdout.flush()


if __name__ == '__main__':
    multiprocessing.log_to_stderr()
    logger = multiprocessing.get_logger()
    logger.setLevel(logging.INFO)
    p = multiprocessing.Process(target=worker)
    p.start()
    p.join()
```

Copyright © 2019 University of Colorado

In Node.js

- Node.js is single threaded, but uses multiple threads in the background to handle asynchronous operations
- The Node.js runtime is non-blocking so functions or callbacks are delegated to the event loop and can be executed by different threads
- Node.js does allow forking multiple processes (for multiple cores)
- State is not shared between processes, but the process send function can pass messages between processes
- Forking processes can increase execution speed

- Reference [5]

Copyright © 2019 University of Colorado

Forking in Node.js [5]

```
//server.js

const { fork } = require('child_process');
app.get('/endpoint', (request, response) => {
   // fork another process
   const process = fork('./send_mail.js');
   const mails = request.body.emails;
   // send list of e-mails to forked process
   process.send({ mails });
   // listen for messages from forked process
   process.on('message', (message) => {
     log.info(`Number of mails sent ${message.counter}`);
   });
   return response.json({ status: true, sent: true });
});
```

```
//send_mail.js

async function sendMultipleMails(mails) {
   let sendMails = 0;
   // logic for
   // sending multiple mails
   return sendMails;
}
// receive message from master process
process.on('message', async (message) => {
  const numberOfMailsSend = await
sendMultipleMails(message.mails);

  // send response to master process
  process.send({ counter: numberOfMailsSend });
});
```

Copyright © 2019 University of Colorado

New in Node.js: Worker Threads

- Experimental API in 10.5 or later, must be used with a flag when executed:
  `>> node --experimental-worker index.js`
- The worker threads can communicate through a postMessage method to a parentPort
- Reference [6]

```
// index.js
// run with node --experimental-worker index.js on Node.js 10.x
const { Worker } = require('worker_threads')

function runService(workerData) {
  return new Promise((resolve, reject) => {
    const worker = new Worker('./service.js', { workerData });
    worker.on('message', resolve);
    worker.on('error', reject);
    worker.on('exit', (code) => {
      if (code !== 0)
        reject(new Error(`Worker stopped with exit code ${code}`));
    }) }) }

async function run() {
  const result = await runService('world')
  console.log(result); }

run().catch(err => console.error(err))
```

Copyright © 2019 University of Colorado

Summary

- Python
    - threading – I/O, network, UI bound
    - multiprocessing – CPU bound with multiple cores available
- Node.js
    - Forked processes for CPU bound with multiple cores
    - Experimental new Worker threads in Node.js 10.5 or later

Copyright © 2019 University of Colorado

Next Steps

- If you're on the waitlist, and you want to join the class, keep attending and do all assignments. It is getting close to the time (end of the week) you may have to consider an alternative class…
  - See Adam Sadoff for questions, I have no control or influence over waitlists!
- Slack and Canvas sign ups, get the books…
- Python and Node.js practice
- Background surveys up shortly…
- First project for you and hardware handouts on 9/9, I'll be following up with distance students for shipping dev kits out
- SA office hours in ECEE 1B24 lab:
  - Shubham - Tues 12-2 PM, Fri 3-5 PM
  - Sharanjeet - Wed 2-3 PM, Mon 2-3 PM

Copyright © 2019 University of Colorado

References

[1] https://www.ploggingdev.com/2017/01/multiprocessing-and-multithreading-in-python-3/
[2] https://docs.python.org/2/library/multiprocessing.html
[3] https://realpython.com/intro-to-python-threading/
[4] https://pymotw.com/3/multiprocessing/basics.html
[5] https://itnext.io/multi-threading-and-multi-process-in-node-js-ffa5bb5cde98
[6] https://blog.logrocket.com/node-js-multithreading-what-are-worker-threads-and-why-do-they-matter-48ab102f8b10/

Copyright © 2019 University of Colorado