INTRODUCTION TO PARALLEL PROCESSING IN PYTHON

SHUI HUNG KWOK

W. M. KECK OBSERVATORY
2017-11-9

SEQUENTIAL – NON-SEQUENTIAL

Sequential

- Simple programming
- Deterministic

Non-sequential

- Parallel
- Concurrent
- Asynchronous
- Interrupt-driven

CONTEXT SWITCHING

Implicitly by	Operating system
	Programming language
	Timer or event
Explicitly by	Programmer (you)

INTER-PROCESS COMMUNICATION

Communication models	
	Signal
	Pipe
	Socket / Network
	Shared memory / memory mapped file
	Message queue / Message passing
	File

SYNCHRONIZATION

Basic concepts

- Race condition
- Atomic operation
- Critical section
- Lock / Semaphore
- Resource starvation

ASYNCHRONOUS TASKS

- Event loops (ie. GUIs)
- Callbacks (ie. on task completion)
- Timer objects (ie. Timer(t, worker).start())
- sched module

PROCESS VS THREAD

Process

- Execution of a program in a private context
- Allocated resources are freed after termination
- Complex inter-process communication

Thread

- Execution in context of a process
- Shared resources
- Needs synchronization

PROCESSES

How to create a process	Old style:	os.system, os.exec, os.Popen
	multiprocessing module	
	subprocess module	
	concurrent.futures module	
Examples	fork, spawn, exec	

THREADS

How to create a thread	Low level	_thread module _dummy_thread module
	threading module	
	Thread pool	
Examples	threads	

GENERATORS

- Provides a lazy evaluation
- Provides a stream of data (huge or endless)
- Compliant with iterator protocol
  - __iter__, next, StopIteration

```
def fib():
    a, b = 0, 1
    while True:
       yield b
       a, b = b, a + b
```

```
f = fib()
for f in range(10):
    print next(fib)
```

COROUTINES WITH GENERATORS

yield/send

coroutine A uses coroutine B

B calls ‘yield’ to return data to A

A calls ‘send’ to return data to B

Until A calls ‘close’ or B returns.