Barrier, Semaphore, Condition Variable

This lesson discusses the various synchronization constructs offered by the multiprocessing module which have counterparts in the threading module.

Barrier, Semaphore, Condition Variable

Semaphore

The **`multiprocessing.Semaphore`** is very similar to **`threading.Semaphore`**. The only difference is that the **`acquire()`** method's first argument is named *block* instead of *blocking* as is the case for threading.Semaphore. Below is a simple program between two processes that take turns to write "ping" and "pong" on the console. The script uses **`multiprocessing.Semaphore`** initialized to zero. Additionally, we also use a **`multiprocessing.Value`** boolean object to indicate to the two processes to exit.

Printing ping and pong

```
from multiprocessing import Semaphore, Process, Value
from ctypes import c_bool
import time
import multiprocessing


def process_A(sem1, sem2, exit):
    while not exit.value:
        print("ping")
        sem1.release()
        sem2.acquire()
        time.sleep(0.05)


def process_B(sem1, sem2, exit):
    while not exit.value:
        # wait for a prime number to become available
        sem1.acquire()
        print("pong")
        sem2.release()
        time.sleep(0.05)


if __name__ == '__main__':
    sem1 = Semaphore(0)
    sem2 = Semaphore(0)

    exit_prog = Value(c_bool, False)

    processA = Process(target=process_A, args=(sem1, sem2, exi
t_prog))
    processA.start()

    processB = Process(target=process_B, args=(sem1, sem2, exi
t_prog))
    processB.start()

    # Let the threads run for 3 seconds
    time.sleep(3)

    exit_prog.value = True

    processA.join()
    processB.join()
```

```
from multiprocessing import Semaphore, Process, Value
from ctypes import c_bool
import time
import multiprocessing


def process_A(sem1, sem2, exit):
    while not exit.value:
        print("ping", flush=True)
        sem1.release()
        sem2.acquire()
        time.sleep(0.05)


def process_B(sem1, sem2, exit):
    while not exit.value:
        # wait for a prime number to become available
        sem1.acquire()
        print("pong", flush=True)
        sem2.release()
        time.sleep(0.05)


if __name__ == '__main__':
    sem1 = Semaphore(0)
    sem2 = Semaphore(0)

    exit_prog = Value(c_bool, False)

    processA = Process(target=process_A, args=(sem1, sem2, exit_prog))
    processA.start()

    processB = Process(target=process_B, args=(sem1, sem2, exit_prog))
    processB.start()

    # Let the threads run for 3 seconds
    time.sleep(3)

    exit_prog.value = True

    processA.join()
    processB.join()
```

Barrier

Barrier

The **`multiprocessing.Barrier`** is similar in behavior to **`threading.Barrier`**. Below we rewrite the barrier program from the barrier section in the threading section:

```
from multiprocessing import Barrier, Process, current_process
import random
import time


def process_task():
    time.sleep(random.randint(0, 5))
    print("\nCurrently {0} processes blocked on barrier".format(barrier.n_waiting),
    barrier.wait()


def when_all_processes_released():
    print("\nAll processes released, reported by {0}".format(current_process().name


num_processes = 5
barrier = Barrier(num_processes, action=when_all_processes_released)
processes = [0] * num_processes

for i in range(num_processes):
    processes[i - 1] = Process(target=process_task)

for i in range(num_processes):
    processes[i].start()
```

Condition Variable

The **`multiprocessing.Condition`** works very similarly to **`threading.Condition`**. Below we rewrite our prime printing method using two processes. One process finds a prime and the other prints it. The main process lets the two child processes run for a few seconds before letting them know it's time to exit via a shared before letting them know it's time to exit via a shared `multiprocessing.Value` boolean variable.

Two process program to find and print primes

```
from multiprocessing import Process, Condition, Value
from ctypes import c_bool
import time
import multiprocessing

def printer_process(exit_prog, found_prime, prime, cond_var):
    while not exit_prog.value:

        cond_var.acquire()
        while not found_prime.value and not exit_prog.value:
            cond_var.wait()
        cond_var.release()

        if not exit_prog.value:
            print(prime.value)
            prime.value = 0

            cond_var.acquire()
            found_prime.value = False
            cond_var.notify()
            cond_var.release()


def is_prime(num):
    if num == 2 or num == 3:
        return True

    div = 2

    while div <= num / 2:
        if num % div == 0:
            return False
        div += 1

    return True


def finder_process(exit_prog, found_prime, prime, cond_var):
    i = 1

    while not exit_prog.value:

        while not is_prime(i):
            i += 1
            # Add a timer to slow down the thread
            # so that we can see the output
            time.sleep(.01)

        prime.value = i

        cond_var.acquire()
        found_prime.value = True
        cond_var.notify()
        cond_var.release()

        cond_var.acquire()
        while found_prime.value and not exit_prog.value:
            cond_var.wait()
        cond_var.release()

        i += 1


if __name__ == '__main__':
    multiprocessing.set_start_method("spawn")

    cond_var = Condition()
    prime = Value('i', 0)
    found_prime = Value(c_bool, False)
    exit_prog = Value(c_bool, False)

    printerProcess = Process(target=printer_process, args=(exit_prog, found_prime, prime, cond_var))
    printerProcess.start()

    finderProcess = Process(target=finder_process, args=(exit_prog, found_prime, prime, cond_var))
    finderProcess.start()
```

```
# Let the threads run for 3 seconds

time.sleep(3)

exit_prog.value = True

# Let the threads exit
cond_var.acquire()
cond_var.notify_all()
cond_var.release()

printerProcess.join()
finderProcess.join()
```

```
from multiprocessing import Process, Condition, Value
from ctypes import c_bool
import time
import multiprocessing

def printer_process(exit_prog, found_prime, prime, cond_var):
    while not exit_prog.value:

        cond_var.acquire()
        while not found_prime.value and not exit_prog.value:
            cond_var.wait()
        cond_var.release()

        if not exit_prog.value:
            print(prime.value)
            prime.value = 0

            cond_var.acquire()
            found_prime.value = False
            cond_var.notify()
            cond_var.release()


def is_prime(num):
    if num == 2 or num == 3:
        return True

    div = 2

    while div <= num / 2:
        if num % div == 0:
            return False
        div += 1

    return True


def finder_process(exit_prog, found_prime, prime, cond_var):
    i = 1

    while not exit_prog.value:

        while not is_prime(i):
            i += 1
            # Add a timer to slow down the thread
            # so that we can see the output
            time.sleep(.01)

        prime.value = i

        cond_var.acquire()
```

```
            found_prime.value = True
            cond_var.notify()

            cond_var.release()

            cond_var.acquire()
            while found_prime.value and not exit_prog.value:
                cond_var.wait()
            cond_var.release()

            i += 1


if __name__ == '__main__':
    multiprocessing.set_start_method("spawn")

    cond_var = Condition()
    prime = Value('i', 0)
    found_prime = Value(c_bool, False)
    exit_prog = Value(c_bool, False)

    printerProcess = Process(target=printer_process, args=(exit_prog, found_prime,
    printerProcess.start()

    finderProcess = Process(target=finder_process, args=(exit_prog, found_prime, pr
    finderProcess.start()

    # Let the threads run for 3 seconds
    time.sleep(3)

    exit_prog.value = True

    # Let the threads exit
    cond_var.acquire()
    cond_var.notify_all()
    cond_var.release()

    printerProcess.join()
    finderProcess.join()
```

← Back

Locks & Reentrant Lock

Next →

Pool

☑ Mark as Completed

Ask a Question

Report an Issue (https://discuss.educative.io/tag/barrier-semaphore-condition-variable__multiprocessing__python-concurrency-for-senior-engineering-interviews)