Quiz 3

Test what you have learnt so far.

Question # 1

Consider the ping pong program from the Semaphore section. Will the program work if both the semaphores `sem1` and `sem2` are initialized to 1 instead of 0? The code is reproduced below for reference:

```
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) # change from 0 to 1 for this question
    sem2 = Semaphore(0) # change from 0 to 1 for this question

    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()
```

Given the current structure of the code, the program may not produce the desired output always. Consider the following sequence:

- Imagine process A gets scheduled but not process B.
- Process A prints **ping** and `releases()` the `sem1`.
- Process A tries to `acquire()` the semaphore `sem2` and is successfully able to do so because it started with an initial value of 1.
- Process A goes to sleep but wakes up before Process B gets scheduled on any of the processors and prints **ping** again thus violating the constraint that two pings can't be printed in succession.

Question # 2

Jenny has just joined your team and sees the "ping-pong" from the semaphore section and reasons she could make the program work using a single semaphore. She sends the following snippet for code-review. Remember the requirement of the program is to alternate printing "ping" and "pong". Two pings or pongs should never be

printed together.

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


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


if __name__ == '__main__':
    sem = Semaphore(0)

    exit_prog = Value(c_bool, False)

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

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

    # Let the threads run for 3 seconds
    time.sleep(3000)

    exit_prog.value = True

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

Q As the lead on the team, what is the feedback you'll give?

- A) Great all looks good, check-it in.
- B) The code will fail in all scenarios
- C) The code will sometimes output the right result and at other times not

Submit Answer

Reset Quiz

Explanation

This is a classic newbie mistake. If you look closely at the code one can reason that the code can possibly print two pings in some scenarios. Consider the following sequence:

1. Process A prints **ping**
2. Then releases the semaphore
3. At this point either process A can continue execution and re-acquire the just released semaphore or process B can be woken up to

resume execution. If process A is chosen to continue then **ping** will be printed in succession. And if process B is chosen then we see **pong** get printed which is the desired outcome.

When dealing with semaphores one can mistakenly assume that a `release()` call would suspend the current thread (or process) and wake up the thread (or process) already waiting on the semaphore. In fact, this question can't be solved correctly with a single semaphore.

Question # 3

Can you solve the ping-pong problem using a condition variable?

Yes, the ping-pong problem can be solved using a condition variable and an associated boolean predicate. Let's call this predicate `flag`. If `flag` is true process A prints ping else it waits on the condition variable. Process B on the other hand prints pong when the predicate `flag` is false. The complete solution appears below:

Solving ping-pong problem using condition variables

```
def process_A(cv, flag, exit):
    while not exit.value:

        with cv:
            while flag.value is False:
                cv.wait()

            print("ping")
            flag.value = False

        time.sleep(0.05)


def process_B(cv, flag, exit):
    while not exit.value:

        with cv:
            while flag.value is True:
                cv.wait()

            print("pong")
            flag.value = True


if __name__ == '__main__':
    cv = Condition()

    exit_prog = Value(c_bool, False)
    flag = Value(c_bool, True)

    processA = Process(target=process_A, args=(cv, flag, exit_
prog))
    processA.start()

    processB = Process(target=process_B, args=(cv, flag, exit_
prog))
    processB.start()
```

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


    exit_prog.value = True

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

You can run the above code in the widget below:

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


def process_A(cv, flag, exit):
    while not exit.value:

        with cv:
            while flag.value is False:
                cv.wait()

            print("ping")
            flag.value = False

        time.sleep(0.05)


def process_B(cv, flag, exit):
    while not exit.value:

        with cv:
            while flag.value is True:
                cv.wait()

            print("pong")
            flag.value = True


if __name__ == '__main__':
    cv = Condition()

    exit_prog = Value(c_bool, False)
    flag = Value(c_bool, True)

    processA = Process(target=process_A, args=(cv, flag, exit_prog))
    processA.start()

    processB = Process(target=process_B, args=(cv, flag, exit_prog))
    processB.start()

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

    exit_prog.value = True

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

Question # 4

Can you solve the ping-pong problem using only a Lock?

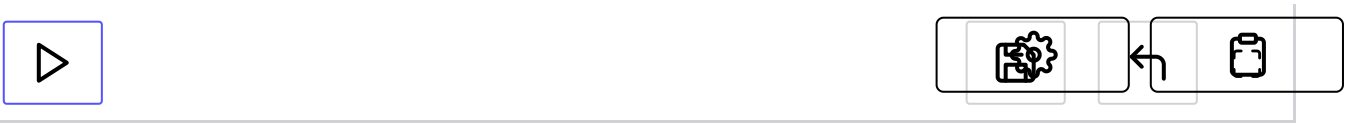
Interestingly enough, we can solve the ping-pong problem using a `Lock` object too. However, `Lock` has no notion of waiting or notification and in this case, we'll need to build a busy-waiting solution that constantly checks if the `flag` variable has flipped. The code is self-explanatory and presented below:

```
def process_A(lock, flag, exit):
    while not exit.value:

        lock.acquire()
        while flag.value is False:
            lock.release()
            lock.acquire()

        print("ping")
        flag.value = False
        lock.release()


def process_B(lock, flag, exit):
    while not exit.value:

        lock.acquire()
        while flag.value is True:
            lock.release()
            lock.acquire()

        print("pong")
        flag.value = True
        lock.release()


if __name__ == '__main__':
    lock = Lock()

    exit_prog = Value(c_bool, False)
    flag = Value(c_bool, True)

    processA = Process(target=process_A, args=(lock, flag, exi
t_prog))
    processA.start()

    processB = Process(target=process_B, args=(lock, flag, exi
t_prog))
```

```
    processB.start()


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

    exit_prog.value = True

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

Pay attention to how the while loops are structured. We always test the value of the `flag` whilst holding the lock.

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


def process_A(lock, flag, exit):
    while not exit.value:

        lock.acquire()
        while flag.value is False:
            lock.release()
            lock.acquire()

        print("ping", flush=True)
        flag.value = False
        lock.release()


def process_B(lock, flag, exit):
    while not exit.value:

        lock.acquire()
        while flag.value is True:
            lock.release()
            lock.acquire()

        print("pong", flush=True)
        flag.value = True
        lock.release()


if __name__ == '__main__':
    lock = Lock()

    exit_prog = Value(c_bool, False)
    flag = Value(c_bool, True)

    processA = Process(target=process_A, args=(lock, flag, exit_prog))
    processA.start()

    processB = Process(target=process_B, args=(lock, flag, exit_prog))
    processB.start()

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

    exit_prog.value = True

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



Back

Quiz 2

Next

Pool Executors

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/quiz-3__multiprocessing__python-concurrency-for-senior-engineering-interviews)