Quiz 1

Test what you have learnt so far.

Question # 1

Consider the snippet below:

```
def child_task(str_obj):
    print(str_obj.__hash__())

if __name__ == '__main__':
    str_obj = "Educative!"
    multiprocessing.set_start_method('fork')

    print(str_obj.__hash__())

    process1 = Process(target=child_task, args=(str_obj,))
    process1.start()

    process2 = Process(target=child_task, args=(str_obj,))
    process2.start()

    process1.join()
    process2.join()
```

Q What will be the output of all the print statements?

A) Exactly the same

B) All are different

C) They may be different

Submit Answer

Reset Quiz

```
1  from multiprocessing import Process, current_process
2  import multiprocessing
3
4  def child_task(str_obj):
5      print(id(str_obj))
6
7  if __name__ == '__main__':
8      str_obj = "Educative!"
9      multiprocessing.set_start_method('fork')
10
11     print(id(str_obj))
12
13     process1 = Process(target=child_task, args=(str_obj,))
14     process1.start()
15
16     process2 = Process(target=child_task, args=(str_obj,))
17     process2.start()
18
19     process1.join()
20     process2.join()
21
```

Question # 2

(/learn)

Consider the same code from the previous question and assume we change the start method from fork to spawn.

Q Will the output of the print statements be same?

- (●) **A)** Yes
- ( ) **B)** No
- ( ) **C)** Maybe

Submit Answer

Reset Quiz

```
1  from multiprocessing import Process, current_process
2  import multiprocessing
3
4  def child_task(str_obj):
5      print(id(str_obj))
6
7  if __name__ == '__main__':
8      str_obj = "Educative!"
9      multiprocessing.set_start_method('spawn')
10
11     print(id(str_obj))
12
13     process1 = Process(target=child_task, args=(str_obj,))
14     process1.start()
```

```
15
16      process2 = Process(target=child_task, args=(str_obj,))
17      process2.start()
18
19      process1.join()
20      process2.join()
21
```

Observe that if the start method is changed to forkserver then the id for **`str_obj`** variable will be the same for all child processes and different for the main process as shown below:

```
from multiprocessing import Process, current_process
import multiprocessing

def child_task(str_obj):
    print("hash code in child process : " + str(id(str_obj)))

if __name__ == '__main__':
    str_obj = "Educative!"
    multiprocessing.set_start_method('forkserver')

    print(id(str_obj))

    process1 = Process(target=child_task, args=(str_obj,))
    process1.start()

    process2 = Process(target=child_task, args=(str_obj,))
    process2.start()

    process1.join()
    process2.join()
```

Question # 3

Consider the RLock example that was presented earlier. Instead of using `multiprocessing.RLock` we replace it with `threading.RLock`.

```
from multiprocessing import Process, current_process
from threading import RLock
import multiprocessing
import time


def child_task(rlock):
    for _ in range(0, 10):
        rlock.acquire()
        print("I am child process {0}".format(current_process
().name))

    for _ in range(0, 10):
        rlock.release()

if __name__ == '__main__':
    multiprocessing.set_start_method('fork')

    rlock = RLock()
    rlock.acquire()

    process1 = Process(target=child_task, args=(rlock,))
    process1.start()

    process2 = Process(target=child_task, args=(rlock,))
    process2.start()

    # sleep 3 seconds before releasing the lock
    time.sleep(3)
    rlock.release()

    process1.join()
    process2.join()
```

Q What will be the outcome of the above program when ran?

A) Exactly the same i.e. the output is printed once the main process releases the lock

B) Error is thrown as we are mixing threads and multiprocessing

C) All the three processes (2 child and 1 main) work in parallel without being blocked.

Submit Answer

Reset Quiz

```
from multiprocessing import Process, current_process
from threading import RLock
import multiprocessing
import time


def child_task(rlock):
    for _ in range(0, 10):
        rlock.acquire()
        print("I am child process {0}".format(current_process().name))

    for _ in range(0, 10):
        rlock.release()

if __name__ == '__main__':
    multiprocessing.set_start_method('fork')

    rlock = RLock()
    rlock.acquire()

    process1 = Process(target=child_task, args=(rlock,))
    process1.start()

    process2 = Process(target=child_task, args=(rlock,))
    process2.start()

    # sleep 3 seconds before releasing the lock
    time.sleep(3)
    rlock.release()

    process1.join()
    process2.join()
```

Explanation

This is a very interesting question if you correctly understand why the above code works. First, realize that we are forking the process, that is we end up with two copies of **RLock** objects, one in each of the processes. Since they are separate objects, the processes work on their

personal copy of `RLock` object and don't block on each other.

Secondly, recognize that the `RLock` object from the main process is copied in the locked state. Even then, the two processes are able to recursively acquire the lock. The owner of the `RLock` will be the main thread of the main process. This should become clear with the next question.

Lastly, each `RLock` object in possession by each of the processes has no awareness of other processes and can't be used to coordinate among the processes. The right construct is to use `multiprocessing.RLock` object.

Question # 4

Assume in the previous question, we change the start method to spawn or forkserver.

Q What would be the outcome of running the snippet with the aforementioned changes?

- A) Error is raised
- B) Output is same as previous question

Submit Answer

Reset Quiz

Question # 5

Consider the below snippet:

```
from multiprocessing import Process, current_process
from threading import RLock, current_thread, Thread
import multiprocessing


def child_task_thread(rlock):
    rlock.release()


def child_task(rlock):
    thread = Thread(target=child_task_thread, args=(rlock,), n
ame="child_process_thread")
    thread.start()
    thread.join()


if __name__ == '__main__':
    multiprocessing.set_start_method('fork')

    rlock = RLock()
    print(rlock.__hash__())
    rlock.acquire()

    process1 = Process(target=child_task, args=(rlock,))
    process1.start()
    process1.join()
```

Q What would be the outcome of running the above snippet?

A) Program runs to completion without errors

B) Unacquired lock release error

Submit Answer

Reset Quiz

```
from multiprocessing import Process, current_process
from threading import RLock, current_thread, Thread
import multiprocessing


def child_task_thread(rlock):
    rlock.release()


def child_task(rlock):
    thread = Thread(target=child_task_thread, args=(rlock,), name="child_process_th
    thread.start()
    thread.join()


if __name__ == '__main__':
    multiprocessing.set_start_method('fork')

    rlock = RLock()
    rlock.acquire()

    process1 = Process(target=child_task, args=(rlock,))
    process1.start()
    process1.join()
```

Explanation

Explanation

The code for this question is conceptually similar to the code in the previous question. We have moved the **`acquire()`** call for the child process in a spawned thread different than the main thread. When the **`RLock`** object is copied for the child process, its owner is the main thread and it is in the locked state. The child process attempts to acquire the **`RLock`** object in a child thread and fails. If the child process acquired the **`RLock`** in its own main thread, the call would succeed because the **`RLock`** object lists the name of the thread that is its owner, which is the main thread. It can't differentiate if the main thread is of the child process or the main process.

This also explains why the code in the previous question worked where the **`RLock`** is acquired in the main thread of the child process.

```
from multiprocessing import Process, current_process
from threading import RLock
import multiprocessing
import threading
import time


def child_task(rlock):
    for _ in range(0, 10):
        rlock.acquire()
        print("I am child process {0}".format(current_process().name))

    for _ in range(0, 10):
        rlock.release()

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

    rlock = threading.RLock()
    # rlock.acquire()

    process1 = Process(target=child_task, args=(rlock,))
    process1.start()

    process2 = Process(target=child_task, args=(rlock,))
    process2.start()

    # sleep 3 seconds before releasing the lock
    time.sleep(3)
    rlock.release()

    process1.join()
    process2.join()
```

← Back

Next →

Namespace

Quiz 2

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/quiz-1__multiprocessing__python-concurrency-for-

issue

senior-engineering-interviews)