操作系统实验报告——作业 3

张缤予 2018010982

实验题目：采用 Python 语言创建多进程

实验目的：基于 python 来对操作系统多进程的工作流程进行模拟

实验基本要求：采用 Python 内置工具包 multiprocessing

实验原理及步骤：

python 中的多线程其实并不是真正的多线程，如果想要充分地使用多核 CPU 的资源，在 python 中大部分情况需要使用多进程。Python 提供了非常好用的多进程包 multiprocessing，只需要定义一个函数，Python 会完成其他所有事情。借助这个包，可以轻松完成从单进程到并发执行的转换。multiprocessing 支持子进程、通信和共享数据、执行不同形式的同步，提供了 Process、Queue、Pipe、Lock 等组件。

查询资料得到如下 5 种情况

1. Linux 操作系统提供了 fork()函数来创建子进程。调用一次，返回两次，因为操作系统是将当前的进程（父进程）复制了一份（子进程），然后分别在父进程和子进程内返回。此函数位于 os 模块下。

2. 创建函数并将其作为单个进程。通过 Multiprocessing 模块中的 Process 类，创建 Process 对象，对象通过参数 target=”需要执行的子进程”

代码如下

```
def pro1(interval):
    for i in range(3):
        print("子进程", i)
        print("The time is {0}".format(time.ctime()))
        time.sleep(interval)


if __name__ == "__main__":
    p = Process(target=pro1, args=(2,))
    p.start()
    # p.join()  # 加入该语句是等子进程结束后再执行下面代码
    print("执行主进程内容")
    print("p.pid:", p.pid)
    print("p.name:", p.name)
    print("p.is_alive:", p.is_alive())
```

结果如下

```
执行主进程内容
p.pid: 19112
p.name: Process-1
p.is_alive: True
子进程 0
The time is Sun Jun 28 17:50:15 2020
子进程 1
The time is Sun Jun 28 17:50:17 2020
子进程 2
The time is Sun Jun 28 17:50:19 2020
```

加入 p.join()后，先执行子进程，完成后执行父进程

加入后代码

```
def pro1(interval):
    for i in range(3):
        print("子进程", i)
        print("The time is {0}".format(time.ctime()))
        time.sleep(interval)


if __name__ == "__main__":
    p = Process(target=pro1, args=(2,))
    p.start()
    p.join()  # 加入该语句是等子进程结束后再执行下面代码
    print("执行主进程内容")
    print("p.pid:", p.pid)
    print("p.name:", p.name)
    print("p.is_alive:", p.is_alive())
```

结果如下

```
子进程 0
The time is Sun Jun 28 18:42:59 2020
子进程 1
The time is Sun Jun 28 18:43:01 2020
子进程 2
The time is Sun Jun 28 18:43:03 2020
执行主进程内容
p.pid: 8680
p.name: Process-1
p.is_alive: False
```

3. 运用子类创建进程，通过继承 Process 类创建子进程并进行重写。

创建子类，继承父类，重写 run 方法，在执行 start()方法时会自动调用 run 方法

代码如下

```
class newProcess(Process):
    # 继承Process类，必须要调用Process中的init初始化参数，
    def __init__(self, interval):
        Process.__init__(self)
        self.interval = interval

    # 重写run方法
    def run(self):
        for i in range(3):
            print("子进程", i)
            print("The time is {0}".format(time.ctime()))
            time.sleep(self.interval)


if __name__ == "__main__":
    p = newProcess(1)
    p.start()
    print("执行主进程")
```

结果如下

```
执行主进程
子进程 0
The time is Sun Jun 28 19:12:19 2020
子进程 1
The time is Sun Jun 28 19:12:20 2020
子进程 2
The time is Sun Jun 28 19:12:21 2020
```

4. 使用进程池（非阻塞式）

当需要创建的子进程数量不多时，可以直接利用 multiprocessing 中的 Process 动态生成多个进程。但是如果是上百甚至上千个目标，手动的去创建进程的工作量巨大，此时就可以使用 multiprocessing 模块中的 Pool 方法。

初始化 Pool 时，可以指定一个最大进程数，当有新的请求提交到 Pool 时，如果池还没有满，那么就会创建一个新的进程来执行该请求；但如果池中的进程数已满，那么该请求就会等待，直到池中有进程结束，才会创建新的进程执行。

Pool 方法即进程池，需要说明有几个进程同时运行。在使用 apply_async 方法时第一个参数是函数名，第二个参数需要输入一个元组。实际上，在进程池的方式中，父进程基本上只需要等待子进程执行（使用 pool.join()进行等待），任务都是交给子进程执行的。

执行说明：创建一个进程池，设定进程的数量为 3，range(4)会生成四个对象[0,1,2,3]，被提交到 pool 进程池中，所以先 start 子进程 0，子进程 1，子进程 2，等到子进程 0 执行完之后，开始执行子进程 3，此时缓冲池里也是 3 个进程。由于为非阻塞形式，主函数会执行自己的函数，和进程的执行互不干扰，所以 for 循环执行完，直接输出“~~~~~~~~~~~~~”。主程序在 pool.join()处等待各个进程结束。

代码如下

```python
def pro2(msg):
    print("msg start:", msg)
    start_time = time.time()
    time.sleep(random.random())
    end_time = time.time()
    print('msg end:{} cost time:{} pid:{}'.format(msg, (end_time - start_time), os.getpid()))
    print('\n')


if __name__ == "__main__":
    pool = Pool(processes=3)
    for i in range(4):
        msg = "子进程 %d" % i
        # 维持执行的进程总数为processes，当一个进程执行完毕后会添加新的进程进去
        pool.apply_async(pro2, (msg, ))

    print("~~~~~~~~~~~~~~~~~~~~")
    pool.close()  # 关闭进程池，不允许继续添加进程
    pool.join()
    print("All processes done.")
```

结果如下

```
~~~~~~~~~~~~~~~~~~~~
msg start: 子进程 0
msg start: 子进程 1
msg start: 子进程 2
msg end:子进程 0 cost time:0.007670164108276367 pid:17404


msg start: 子进程 3
msg end:子进程 2 cost time:0.12033462524414062 pid:1964


msg end:子进程 1 cost time:0.7228708267211914 pid:14504


msg end:子进程 3 cost time:0.846930980682373 pid:17404


All processes done.
```

5. 使用缓冲池（阻塞式）

阻塞式特点：添加一个任务，就执行一个任务，如果一个任务不结束，下一个任务就不会被添加进来。与非阻塞的区别是使用 pool.apply 方法。

代码如下

```python
def pro3(msg):
    print("%s start:" % msg)
    start_time = time.time()
    time.sleep(random.random()*3)
    end_time = time.time()
    print('msg end:{} cost time:{} pid:{}'.format(msg, (end_time - start_time), os.getpid()))
    print("msg end:", msg)
    print('\n')


if __name__ == "__main__":
    pool = Pool(processes=4)
    for i in range(4):
        msg = "子进程 %d" % i
        # 维持执行的进程总数为processes，当一个进程执行完毕后会添加新的进程进去
        pool.apply(pro3, (msg, ))

    print("~~~~~~~~~~~~~~~~~~~~")
    pool.close()  # 执行完close后不会有新的进程加入到pool
    pool.join()   # join函数等待所有子进程结束
    print("All processes done.")
```

结果如下

```
子进程 0 start:
msg end:子进程 0 cost time:1.1772291660308838 pid:9808
msg end: 子进程 0


子进程 1 start:
msg end:子进程 1 cost time:1.790663719177246 pid:16088
msg end: 子进程 1


子进程 2 start:
msg end:子进程 2 cost time:2.806051731109619 pid:2912
msg end: 子进程 2


子进程 3 start:
msg end:子进程 3 cost time:2.924196481704712 pid:18648
msg end: 子进程 3


~~~~~~~~~~~~~~~~~~
All processes done.
```

实验结果或结论：

采用 python 语言创建多进程的方法有很多，最常用的还是 multiprocessing 的内置工具包，根据不同的进程实现要求，可以采用不同的方法，借助 multiprocessing 可以完成单进程到多进程并发执行的转换。缓冲池除了有阻塞时，非阻塞式，还可以通过多个缓冲池进行实现