操作系统实验二实验报告
快速排序问题

张子扬 无06 2020010790

一、 实验目的

1. 通过对进程间高级通信问题的编程实现，加深理解进程间高级通信的原理；
2. 对 Windows 或 Linux 涉及的几种高级进程间通信机制有更进一步的了解；
3. 熟悉 Windows 或 Linux 中定义的与高级进程间通信有关的函数。

二、 实验平台

本实验在 Python3.10.0 上运行。

三、 实验原理

本实验选用共享内存实现进程(线程)间通信，分别实现了进程与线程两个版本的代码。实验中使用一个队列实现多进程（线程）间的任务分配。

四、 算法设计思路

首先创建 20 个进程（线程）、任务队列、共享内存。其中任务队列中的任务由（起始下标，终止下标）元组构成，任务队列中开始只有待排序数组的起始下标与终止下标构成的元组。共享内存中包含待排序的乱序数组。

之后 20 个进程（线程）并行、互斥地从任务队列中取出任务，如果任务的长度大于等于 1000，则将数组的该部分划分为三个部分：左侧、锚点、右侧。其中左侧的数组一定小于等于锚点，右侧的数组一定大于等于锚点，但是不保证左侧和右侧是有序的。之后分别将左侧和右侧的起始下标、终止下标作为任务元组放到任务队列队尾。如果任务的长度小于 1000，则对数组的该部分应用快速排序算法。

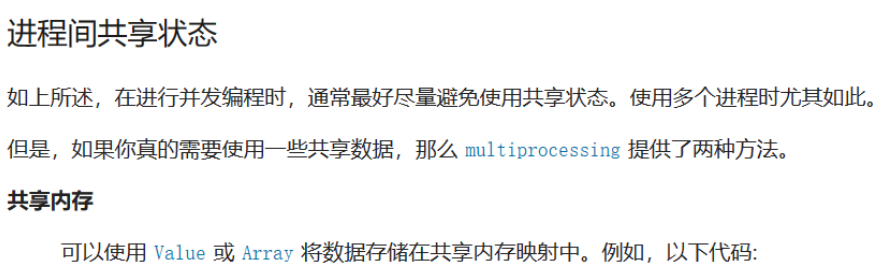
五、 算法实现

进程（线程）函数的调用实现如下所示。

```
def worker(segments, arr):
    while True:
        try:
            segment = segments.get(timeout=1)
        except queue.Empty:
            break
        # 长时间得到不任务证明不需要这么多进程（线程），则关闭该进程（线程）
```

```
        left, right = segment
        if right - left + 1 >= 1000:
            pivot = partition(arr, left, right) # 将任务划分为两个子任务
            segments.put((left, pivot - 1))
            segments.put((pivot + 1, right))
        else:
            quick_sort(arr, left, right) # 执行快速排序
```

由于线程之间本就是共享内存的，所以线程版本并没有为了“共享内存”进行额外的操作。如果认为这样不算做“共享内存”的话，那么我还实现了进程版本。在进程版本中使用了 Python multiprocessing 库中的 Array 函数，将数据存储在共享内存映射中，使得每个进程都可以访问这一共享内存。


进程间共享状态

如上所述，在进行并发编程时，通常最好尽量避免使用共享状态。使用多个进程时尤其如此。

但是，如果你真的需要使用一些共享数据，那么 multiprocessing 提供了两种方法。

共享内存

可以使用 Value 或 Array 将数据存储在共享内存映射中。例如，以下代码:

上图是 Python 官方文档对 multiprocessing 库实现共享内存的说明，详情见 https://docs.python.org/zh-cn/3.10/library/multiprocessing.html。

六、 运行结果与分析

运行多线程快速排序算法，结果如下：

```
Single Thread sorting Time: 12.79977011680603
Multi Thread creation Time: 6.562233209609985
Multi Thread sorting Time: 8.533598184585571
Total Time: 15.095831394195557
Sorted: True
```

可以看到虽然多线程的排序总时间相较于单线程更长，但是多线程的相当一部分时间用在了建立线程上，如果只比较排序时间，那么多线程更快一些。这也是可以解释的，因为多线程可以并行计算，而快速排序不同区域是互不影响的，适用于并行计算场景。因此使用多线程排序时间更短是合理的。

运行多进程快速排序算法，结果如下：

```
Single Process sorting Time: 12.916302680969238
Multi Process sorting Time: 1026.9219751358032
Sorted: True
```

可以看到多进程的排序总时间非常长，原因可能是计算机并没有足够的硬件支持 20 个进程，所以该 20 个进程必须不停“轮换”执行，所以有相当多时间花费在了加载和退出上。也有可能是 Python 对于共享内存的支持不好，使得共享内存的访问比较慢。

七、 思考题

1. 你采用了你选择的机制而不是另外的两种机制解决该问题，请解释你做出这种选择的理由。

在本次实验中，需要多个进程同时对同一序列进行操作。为了实现进程间的通信，共享内存是最适合的选择。相比于管道和消息队列，共享内存可以实现多个进程之间的高效通信。管道和消息队列主要用于单个线程与单个线程之间的通信，并不适用于本次实验的需求。如果在本次实验中使用管道或者消息队列，则每个线程都需要与主线程进行通信才能完成所需功能。然而，使用管道和消息队列来实现这种通信会更加复杂且效率较低。因此，共享内存是最合适且高效的通信方式。

2. 你认为另外的两种机制是否同样可以解决该问题？如果可以请给出你的思路；如果不能，请解释理由。

可以。每个线程在进行排序之前需要与主线程进行通信，以获取当前序列的信息。排序完成后，线程再次与主线程进行通信，将排序结果传回。这样可以实现每个线程与主线程之间的通信，进而实现不同线程之间的通信。