# 多线程及其简单应用

1. **多线程定义及原理**

在操作系统层面上，在线程被定义出来之前，操作系统调度的最基本单位是进程。随着操作系统的不断完善和发展，线程诞生并逐渐替代原来进程在调度中的地位。现在，线程是操作系统任务调度的最小单位，进程是资源分配的最小单位，一个进程可以包含多个线程，线程共享进程中的资源。

每个正在系统上运行的程序都是一个进程。每个进程包含一到多个线程。进程也可能是整个程序或者是部分程序的动态执行。线程是一组指令的集合，或者是程序的特殊段，它可以在程序里独立执行。也可以把它理解为代码运行的上下文。所以线程基本上是轻量级的进程，它负责在单个程序里执行多任务。通常由操作系统负责多个线程的调度和执行。

实现多线程是采用一种并发执行机制，并发执行机制原理简单地说就是把一个处理器划分为若干个短的时间片，每个时间片依次轮流地执行处理各个应用程序，由于一个时间片很短，相对于一个应用程序来说，就好像是处理器在为自己单独服务一样，从而达到多个应用程序在同时进行的效果。多线程就是把操作系统中的这种并发执行机制原理运用在一个程序中，把一个程序划分为若干个子任务，多个子任务并发执行，每一个任务就是一个线程。.

· 线程是程序中一个单一的顺序控制流程.在单个程序中同时运行多个线程完成不同的工作,称为多线程.

· 线程和进程的区别在于,子进程和父进程有不同的代码和数据空间,而多个线程则共享数据空间,每个线程有自己的执行堆栈和程序计数器为其执行上下文.多线程主要是为了节约CPU时间,发挥利用,根据具体情况而定. 线程的运行中需要使用计算机的内存资源和CPU。

1. **多线程应用**

在人工智能领域，多线程被运用到很多地方，比如基于Django的全栈开发、模型的端侧部署、再比如基于python的爬虫技术等等，无一不用到了多线程的实现方式来大大提高响应速度或者是大大减少工作时间。举个例子，当某一个计算机视觉方面的项目需要海量数据（大约10万张）图像，如果按照传统的顺序执行方式来做，这个时间可能持续几个小时（相对于电脑的性能而言），但是如果使用多线程的方式来爬取这么庞大数量的图像数据，时间至少会减半。这是因为在网络爬虫程序是一种 IO 密集型程序，程序中涉及了很多网络 IO 操作以及本地磁盘 IO 操作，这些都会消耗大量的时间，从而降低程序的执行效率。但是使用多线程，线程会在遇到IO操作的时候会自动切换到其他需要占用CPU资源的线程，将原线程挂起，当等待的IO操作结束后，再将原线程唤醒，在这期间CPU一直在处理，利用率大大的提高了，自然就提高了爬虫的效率。

1. **多线程实现**

**3.1. 实验环境**

在本次实验中，我选择了Python语言为编程语言，主要原因是，Python提供了多线程工具包，封装完好，调用方便。

**3.2. 实验流程**

Python 提供了两个支持多线程的模块，分别是 \_thread 和 threading。其中 \_thread 模块偏底层，它相比于 threading 模块功能有限，因此本实验采用 threading 模块。

1. **多线程代码**
2. import requests
3. from lxml import etree
4. import threading, psutil, os
5. import time
6. from concurrent.futures import ThreadPoolExecutor
7. headers = {
8. 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.25 Safari/537.36 Core/1.70.3863.400 QQBrowser/10.8.4334.400',
9. }
10. *# 定义线程函数*
11. def call\_back(res):
12. *# 完成线程的操作，进行回调的函数*
13. res = res.result()  *# 获取结果*
14. *# print(res)*
15. def downmp4(video\_url, video\_name):
16. path = '包图网大国工匠视频'
17. *# 对当前目录下面的文件夹进行判断，如果没有自动创建一个文件夹拉存储*
18. if not os.path.exists(path):
19. os.mkdir(path)
20. *# 得到进程线程相关的信息*
21. thread = threading.current\_thread()  *# 得到当前的线程对象*
22. process = psutil.Process(os.getpid())  *# 得到当前的进程对象*
23. print(thread.ident, thread.name, process.pid, process.name())
24. start\_time = time.time()
25. video\_url = "https:" + video\_url
26. video\_name = video\_name.strip().replace("<strong>", "").replace("</strong>", "")
27. video\_content = requests.get(url=video\_url, headers=headers).content
28. with open(path + './%s.mp4' % video\_name, "wb") as f:
29. f.write(video\_content)
30. finish\_time = time.time() - start\_time
31. return "每个视频下载的时间" + str(finish\_time)
32. def parsePage():
33. *# 第一步，确定爬虫地址*
34. url = "https://ibaotu.com/tupian/gongjiangjingshen/7-0-0-0-0-0-0.html?format\_type=0"
35. *# 第二步：发送请求*
36. response = requests.get(url=url, headers=headers)
37. *# 第三步：获取数据*
38. html\_content = response.text
39. *# 第四部：保存在本地*
40. *# with open('baotuwang.html', 'w',encoding='utf-8') as f:*
41. *#     f.write(html\_content)*
42. *# 提取a标签*
43. tree = etree.HTML(html\_content)
44. video\_page\_url = tree.xpath("//ul/li/div/div/a/div[1]/video/@src")
45. print(video\_page\_url)
46. video\_name = tree.xpath("//ul/li/@pr-data-title")
47. print(video\_name)
48. start\_time = time.time()
49. *##################线程池的实现方法1###############*
50. *# 1.创建线程池，初始化线程数量*
51. executor = ThreadPoolExecutor(4)
52. for i in range(len(video\_page\_url)):
53. *# 进入到详情页面下载视频,封装成一个函数----相当于任务,call\_back是回调函数*
54. executor.submit(downmp4, video\_page\_url[i], video\_name[i]).add\_done\_callback(call\_back)  *# 提交任务*
55. *# 关闭线程*
56. executor.shutdown(True)
57. finish\_time = time.time()
58. print("总共下载的时间是" + str(finish\_time - start\_time))
59. if \_\_name\_\_ == '\_\_main\_\_':
60. parsePage()
61. **多线程结果**

普通爬取结果如下图1所示。

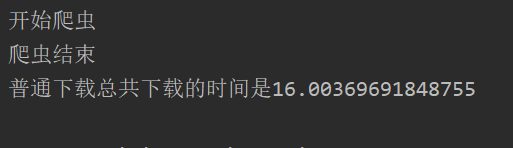


图1 普通爬取结果

利用多线程爬取结果如下图2示。

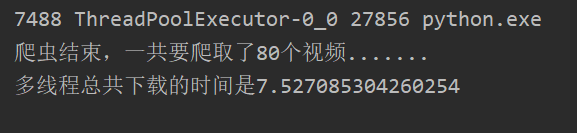


图2利用多线程爬取结果

1. **实验感悟与总结**

通过上述两种方法爬虫的结果（单线程或单进程与多线程）进行比较，不难发现，爬取相同大小的视频数据，利用单线程的时间要远远大于普通下载的时间，这是因为在通爬取的过程中，每一次都要申请磁盘和释放磁盘资源，这样大大降低了爬取的效率。但是在多线程中，设置四个线程的情况下，花费的时间仅为7秒，比普通下载所花费时间的一半少，这是因为每次释放和申请磁盘资源的时候会将其挂起，执行其他的CPU操作。

和线程相比进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

【注】 本报告中摘录的爬虫代码遵循robot协议，只用于学术交流，不允许做商业用途。