2020年9月18日 16:41

目录 - 多进程

Threading by 莫凡

2020年9月17日 14:45

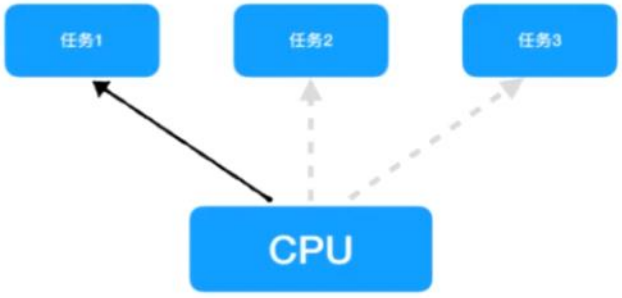
多进程 IT黑马

2020年9月18日 10:58

1. 多任务的介绍
    a. 多任务的概念
    多任务是指在同一时间内执行多个任务
    b. 多任务的两种表现形式
    - 并发
    - 并行
    c. 并发
    在一段时间内交替去执行对个任务

例子：

对于单核cpu处理多任务,操作系统轮流让各个**任务交替执行。**



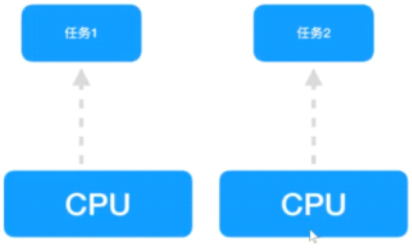
并发：任务数量大于CPU的核心数

1.4 并行

在一段时间内**真正的同时一起**执行多个任务。

例子：

对于多核cpu处理多任务，操作系统会给cpu的每个内核安排一个执行的任务，多个内核是真正的**一起同时执行多个任务**。这里需要注意多核cpu是并行的执行多任务，始终有多个任务一起执行。

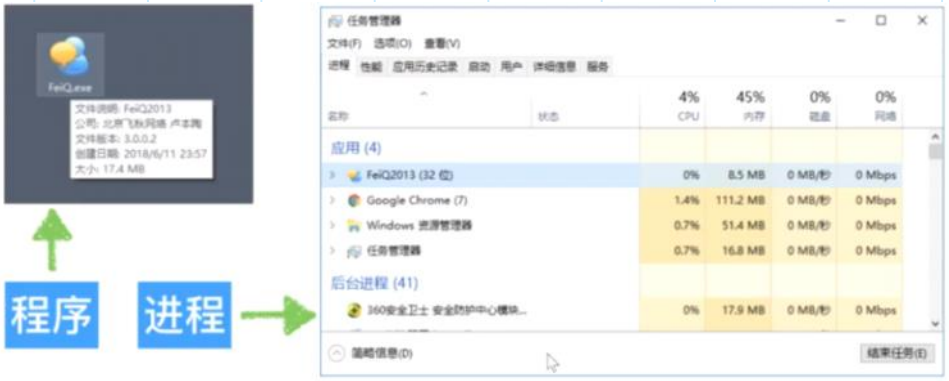


并行：任务数量小于或等于CPU的核心数

第一课：进程介绍

2020年9月18日 11:09

1. 程序中实现多任务的方式
   在python中，想要实现多任务可以使用多进程来完成
2. 进程的概念
   进程(process) 是资源分配的最小单位，它是操作系统进行资源分配和调度运行的基本单位，通俗理解：一个正在运行的程序就是一个进程。例如：正在运行的qq,微信等 他们都是一个进程



2.3 多进程的作用

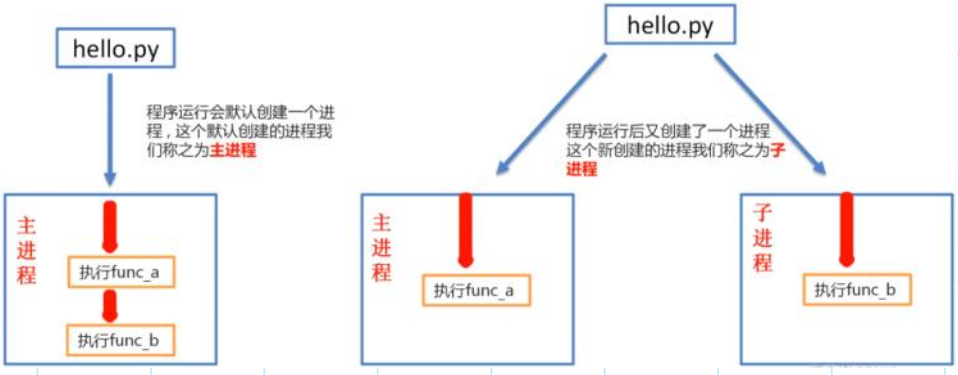
```python
def func_a():
    print("任务A")

def func_b():
    print("任务B")

func_a()
func_b()
```

思考：
图中是一个非常简单的程序，一旦运行hello.py这个程序，按照代码的执行顺序，func_a函数执行完毕后才能执行func_b函数，如果可以让func_a和func_b同时运行，显然执行hello.py这个程序的效率会大大提升。

2.4 多进程的作用



第二课：多进程完成多任务

2020年9月18日 11:19

3.1进程的创建步骤

1.导入进程包

Import multiprocessing

2.通过进程类创建进程对象

进程对象 = multiprocessing.Process()

3.启动进程执行任务

进程对象.Start()

3.2 通过进程类创建进程对象

进程对象 = multiprocessing.Process(target=任务名)

参数名	说明
target	执行的目标任务名,这里指的是函数名(方法名)
name	进程名，一般不用设置
group	进程组，目前只能使用None

3.3 进程创建与启动的代码

```
# 创建子进程
sing_process = multiprocessing.Process(target=sing)
# 创建子进程
dance_process = multiprocessing.Process(target=dance)
# 启动进程
sing_process.start()
dance_process.start()
```

01-单任务.py × 02-使用多进程实现多任务.py ×

```python
import time


# 唱歌
def sing():
    for i in range(3):
        print("唱歌...")
        time.sleep(0.5)


# 跳舞
def dance():
    for i in range(3):
        print("跳舞...")
        time.sleep(0.5)


if __name__ == '__main__':
    sing()
    dance()
```

```python
# 跳舞
def dance():
    for i in range(3):
        print("跳舞...")
        time.sleep(0.5)


if __name__ == '__main__':
    # 2. 使用进程类创建进程对象
    #   target: 指定进程执行的函数名
    sing_process = multiprocessing.Process(target=sing)
    dance_process = multiprocessing.Process(target=dance)

    # 3. 使用进程对象启动进程执行指定任务
    sing_process.start()
    dance_process.start()
```

```python
# 1. 导入进程包
# 2. 使用进程类创建进程对象
# 3. 使用进程对象启动进程执行指定任务

# 1. 导入进程包
import multiprocessing
import time
```

第三课：进程执行带有参数的任务

2020年9月18日 14:43

4.1 进程执行带有参数的任务

参数名	说明
args	以元组的方式给执行任务传参
kwargs	以字典方式给执行任务传参

4.2 args参数的使用

```
# target: 进程执行的函数名
# args: 表示以元组的方式给函数传参
sing_process = multiprocessing.Process(target=sing, args=(3,))
sing_process.start()
```

4.3 kwargs参数的使用

```
# target: 进程执行的函数名
# kwargs: 表示以字典的方式给函数传参
dance_process = multiprocessing.Process (target=dance, kwargs={"num": 3})
# 启动进程
dance_process.start()
```

```python
# 唱歌
def sing(num):
    for i in range(num):
        print("唱歌...")
        time.sleep(0.5)


# 跳舞
def dance():
    for i in range(3):
        print("跳舞...")
        time.sleep(0.5)


if __name__ == '__main__':
    # 2. 使用进程类创建进程对象
    #   target: 指定进程执行的函数名
    #   args: 使用元组方式给指定任务传参
    sing_process = multiprocessing.Process(target=sing, args=(3,))
    dance_process = multiprocessing.Process(target=dance)

    # 3. 使用进程对象启动进程执行指定任务
    sing_process.start()
    dance_process.start()
```

```python
# 跳舞
def dance(num):
    for i in range(num):
        print("跳舞...")
        time.sleep(0.5)


if __name__ == '__main__':
    # 2. 使用进程类创建进程对象
    #   target: 指定进程执行的函数名
    #   args: 使用元组方式给指定任务传参
    #   kwargs: 使用字典方式给指定任务传参
    sing_process = multiprocessing.Process(target=sing, args=(3,))
    dance_process = multiprocessing.Process(target=dance, kwargs={"num": 2})
```

第四课：获取进程的编号

2020年9月18日 14:56

进程编号的作用:

当程序中进程的数量越来越多时，如果没有办法区分主进程和子进程还有不同的子进程，那么就无法进行有效的进程管理，为了方便管理实际上每个进程都是有自己编号的.

获取进程编号的两种方式：

1. 获取当前进程编号

os.getpid()

2. 获取当前父进程编号

os.getppid()

5.2 os.getppid()的使用

```
def work():
    # 获取当前进程的编号
    print("work进程编号:", os.getpid())
    # 获取父进程的编号
    print("work父进程的编号:", os.getppid())
```

```
# 唱歌
def sing(num, name):
    print("唱歌进程的pid: ", os.getpid())
    for i in range(num):
        print(name)
        print("唱歌...")
        time.sleep(0.5)


# 跳舞
def dance(num, name):
    print("跳舞进程的pid: ", os.getpid())
    for i in range(num):
        print(name)
        print("跳舞...")
        time.sleep(0.5)


if __name__ == '__main__':
    ...
    sing_process = multiprocessing.Process(target=sing, args=(3, "xiaoming"))
    dance_process = multiprocessing.Process(target=dance, kwargs={"name": "传智",
 "num": 2})

    # 3. 使用进程对象启动进程执行指定任务
    sing_process.start()
    dance_process.start()
```

```python
if __name__ == '__main__':                                    主进程
    1. 创建子进程对象并指定执行 的任务名
    sing_process = multiprocessing.Process(target=sing, args=(3, "xiaoming"))
    dance_process = multiprocessing.Process(target=dance, kwargs={"name": "传智",
"num": 2})

    2. 启动子进程并执行任务 动进程执行指定任务
    sing_process.start()
    dance_process.start()
```

第五课：进程的注意点

2020年9月18日 15:23

主进程会等待所有的子进程执行结束再结束

6.1 主进程会等待所有的子进程执行结束再结束

```
def work():
    for i in range(10):
        print("工作中...")
        time.sleep(0.2)

if __name__ == '__main__':
    # 创建子进程
    work_process = multiprocessing.Process(target=work)
    work_process.start()
    # 让主进程等待1秒钟
    time.sleep(1)
    print("主进程执行完成了啦")
    # 总结： 主进程会等待所有的子进程执行完成以后程序再退出
```

6.2 设置守护主进程

```
def work():
    for i in range(10):
        print("工作中...")
        time.sleep(0.2)
if __name__ == '__main__':
    # 创建子进程
    work_process = multiprocessing.Process(target=work)
    # 设置守护主进程，主进程退出后子进程直接销毁，不再执行子进程中的代码
    work_process.daemon = True
    work_process.start()
    # 让主进程等待1秒钟
    time.sleep(1)
    print("主进程执行完成了啦")
```

6.3 知识要点

为了保证子进程能够正常的运行，主进程会等所有的子进程执行完成以后再销毁，设置守护主进程的目的是**主进程退出子进程销毁**，不让主进程再等待子进程去执行。

设置守护主进程方式： **子进程对象.daemon = True**

第六课：传智教学视频文件夹高并发copy器

2020年9月18日 15:48

案例：需求分析

① 目标文件夹是否存在，如果不存在就创建，如果存在则不创建
② 遍历源文件夹中所有文件，并拷贝到目标文件夹
③ 采用进程实现多任务，完成高并发拷贝

1、定义源文件夹所在的路径、目标文件夹所在路径

```
# 1、定义源文件目录和目标文件夹的目录
source_dir = "python教学视频"
dest_dir = "/home/python/桌面/test"
```

2、创建目标文件夹

```
try:
    # 2、创建目标文件夹目录
    os.mkdir(dest_dir)

except:
    print("目标文件夹已经存在，未创建~")
```

3、通过os.listdir 获取源目录中的文件列表

```
# 3、列表得到所有的源文件中的文件
file_list = os.listdir(source_dir)
print(file_list)
```

4、遍历每个文件，定义一个函数，专门实现文件拷贝

```
    # 4、for 循环，依次拷贝每个文件
    for file_name in file_list:
        copy_work(file_name, source_dir, dest_dir)
```

5、采用进程实现多任务，完成高并发拷贝

```
    # 4、for 循环，依次拷贝每个文件
    for file_name in file_list:
        # copy_work(file_name, source_dir, dest_dir)
        sub_process = multiprocessing.Process(target=copy_work,
args=(file_name, source_dir, dest_dir))
        sub_process.start()
```

案例：文件拷贝函数实现步骤

1、拼接源文件和目标文件所在的路径

```
def copy_work(file_name, source_dir, dest_dir):

    # 拼接路径
    source_path = source_dir+"/"+file_name

    dest_path = dest_dir+"/"+file_name
```

2、打开源文件、创建目标文件

```
def copy_work(file_name, source_dir, dest_dir):
    # 拼接路径
    source_path = source_dir+"/"+file_name
    dest_path = dest_dir+"/"+file_name

    print(source_path, "----->", dest_path)
    # 打开源文件、创建目标文件
    with open(source_path,"rb") as source_file:
        with open(dest_path,"wb") as dest_file:
```

3、读取源文件的内容并且写入到目标文件中（循环）

```
def copy_work(file_name, source_dir, dest_dir):
            .....
            while True:
                # 循环读取数据
                file_data = source_file.read(1024)
                if file_data:
                    # 循环写入到目标文件
                    dest_file.write(file_data)
                else:
                    break
```

```
import os
import multiprocessing


def copy_file(file_name, source_dir, test_dir):
    # 拼接源文件路径和目标文件路径
    source_path = source_dir + "/" + file_name
    test_path = test_dir + "/" + file_name
    # 打开源文件和目标文件
    with open(source_path, 'rb') as source_file:
        with open(test_path, "wb") as test_file:
            # 循环读取源文件到目标文件
            while True:
                data = source_file.read(1024)
                if data:
                    test_file.write(data)
                else:
                    break
```

```
if __name__ == '__main__':
    # 定义源文件夹和目标文件夹
    source_dir = "python教学视频"
    name = input("请输入创建的文件夹名：")
    test_dir = "C:/Users/Charles/Desktop/" + name
    # 创建目标文件夹
    try:
        os.mkdir(test_dir)
    except Exception as e:
        print(e)
    # 读取源文件夹的文件列表
    file_list = os.listdir(source_dir)
    # 遍历文件夹的文件列表
    for file_name in file_list:
        # copy_file(file_name,source_dir,test_dir)
        # 使用多进程实现多任务
        sub_process = multiprocessing.Process(target=copy_file, args=(file_name, source_dir, test_dir))
        sub_process.start()
```