多线程与多进程编程
Part II

马栩杰

2016 年 7 月 3 日

讲义 Slides 版见 `https://cdn.rawgit.com/ma-xujie/reveal.js/multiprocessing/index.html`

1 多进程编程

1.1 Python 中的多进程实现

同样介绍一下 Python 的多进程编程。

在 Python 中，多进程的使用也非常容易，接口与 Thread 几乎一致[1]。

```
1  import multiprocessing
2
3  # 定义 foo
4
5  p = multiprocessing.Process(target=foo, name='hehe')
6  p.start()
7
8  # 做别的事情
9
10 p.join()
```

单纯看接口的话，会觉得多线程和多进程没有什么差别的样子。

在实际运行的过程中，Python 的多进程运行不受 GIL 的限制，在计算密集的情况下可以大大提高多核 CPU 的利用率[2] 。

如果希望在多进程之间传递数据的话，`multiprocessing.Queue` 是一个很好的选择。Python 的多进程队列和 `queue.Queue` 的接口是一致的，并且可以在多进程的情况下工作[3]。

启动一个新的进程的时间和内存开销要远远大于启动新线程。在 Python 中如果需要让大量独立的进程并行执行很多任务的话，可以考虑使用进程池（`multiprocessing.Pool`）实现。

[1] 大概是为了弥补 GIL 的影响，让大家在需要多任务并行的时候可以方便地从 threading 移动到 multiprocessing

[2] 队式 17 虽然是用多线程完成的，但是选手的代码运行并不会受到 GIL 的限制，原因是用 Python 的 ctypes 库调用 dll 的时候会自动释放全局锁。

[3] 为什么不能用 `queue.Queue` 呢？因为如果将队列作为参数传入新进程中的话你会发现它们已经不再是同一个对象了..

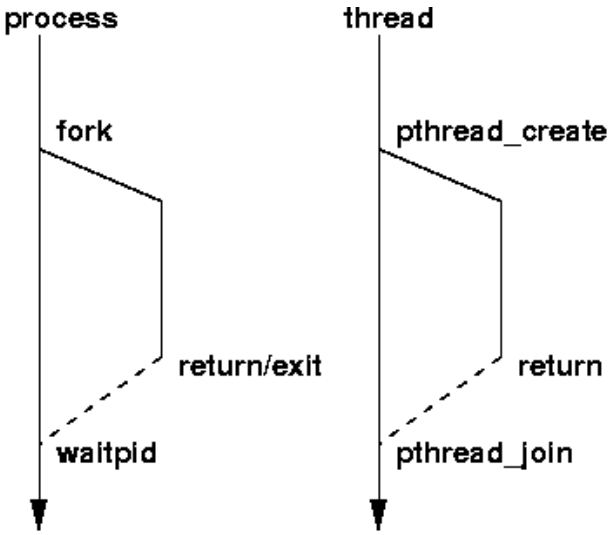
1.2 C++ 中的多进程

C++ 标准库不支持多进程编程。

在 *nix 系统中，可以使用 unistd.h 中的 fork 来创建新的进程。

```
#include <stdio.h>
#include <unistd.h>

int main() {
  printf("Hello!\n");
  pid_t pid = fork();
  printf("%d\n", pid);
  if (pid == 0) {
    printf("I'm child process.\n");
  } else {
    printf("I'm main process, child pid = %d.\n", pid);
  }
}
```

在上述代码中，使用 fork 创建了一个新的进程。



在调用 fork 时，会在 fork 处复制当前的进程，产生一个子进程。fork 的返回值对两个进程不同：在父进程中返回子进程的 pid，在子进程中返回 0。然后父进程和子进程同时在 fork 后继续运行。此时，父进程和子进程的堆栈中的数据内容是相同的，但是此后两个进程对变量的修改都互相不会产生影响。

通常在 fork 后需要两个进程做不同的工作，因此 Linux 提供了一系列 exec 函数[4][5]，让一个进程可以切换到另一项工作。

在 Windows 中，引用 windows.h 并调用 CreateProcess 等函数也可以进行多进程编程[6]。

[4]我没用过..

[5]http://man7.org/linux/man-pages/man3/exec.3.html

[6]https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx

由于 C++ 多进程的实现与平台相关，而且与多线程程序相比很不直观，因此并不推荐用系统原生方法写多进程程序[7]。

1.3 多线程与多进程的比较

多线程

- 共享内存地址，线程间交换数据容易
- 同步需要考虑线程安全问题
- 创建、销毁时的开销较小
- 运行时占用资源小
- 一个线程挂掉会导致整个进程挂掉
- （CPython）受 GIL 的限制

多进程

- 内存地址分离，进程间交换数据困难
- 数据相对独立，一般不用担心安全性
- 创建、销毁时的开销较大
- 运行时占用资源大
- 多个进程相对独立
- （CPython）不受 GIL 限制

通常情况下，IO 密集 / 多任务计算量大，任务间切换频繁 / 大量创建和销毁子任务的程序优先选择多线程编程。

如果多个任务之间的相关性较弱，或者是有多机分布计算需求的话，考虑使用多进程。

Python 要并行进行大量计算也优先考虑不受 GIL 限制的多进程。

2 进程间通信与跨语言通信

进程间通信（IPC）指至少两个进程或线程间传送数据或信号的一些技术或方法。进程是计算机系统分配资源的最小单位 (严格说来是线程)。每个进程都有自己的一部分独立的系统资源，彼此是隔离的。为了能使不同的进程互相访问资源并进行协调工作，才有了进程间通信。[8]

再以队式 17 为例：

[7] 我并没有写过，如果要写的话请自行查找方法

[8] https://zh.wikipedia.org/wiki/%E8%A1%8C%E7%A8%8B%E9%96%93%E9%80%9A%E8%A8%8A

- 进程 1：平台
  - 线程 1：平台主程序
  - 线程 2：逻辑
  - 线程 3：回放生成器
  - 线程 4：选手 1
  - 线程 5：选手 2
  - 线程 6：socket 发送
  - 线程 7：socket 接收
- 进程 2：3D 播放器
  - 线程 8：播放器主程序
  - 线程 9：socket 收发
  - 线程 10：Unreal 启动的各种线程
  - 线程 11：...

由于选手程序被编译成了 dll，因此这一部分被合并到平台进程中，简化了选手和平台通信的过程。在实际实现中，采用了 Python ctypes 库提供的封装回调函数的办法，让选手的 C++ 程序可以在 C 线程内直接调用 Python 中实现的通信函数。而 Python 中通信函数使用队列来实现，每个选手的线程可以向队列中放入指令，由平台主线程拿出来处理。

不过由于 3D 播放器使用虚幻引擎制作生成，因此播放器和平台的两个进程之间只能通过耦合度更低的方式通信。在实现时，使用 Socket 在两个进程间传递数据。

2.1 Socket 通信

现代互联网通信协议统称为 TCP/IP 协议[9][10]。

接入互联网的每台计算机有惟一的 IP 地址[11]，作为计算机在网络中的标识。一台计算机上同时有很多不同应用程序都有网络访问的需求，同一计算机上不同的程序和功能在收发信息时使用不同端口进行区分[12]。

Socket 是一种常用的网络中进程通信的工具，指定 **IP 地址**、**协议**和**端口**，就可以实现网络中两个进程的点对点通信（当然也可以实现本机的进程间通信）。

我们还是以 Python 为例来实现一个本地的进程间 Socket 通信的例子：

服务器端：

```
1  # -*- coding: utf-8 -*-
2
3  import socket
4  import threading
5
```

[9] https://zh.wikipedia.org/wiki/TCP/IP%E5%8D%8F%E8%AE%AE%E6%97%8F

[10] http://www.liaoxuefeng.com/wiki/001431608955726 4a6b348958f449949df42a6d3a2e542c000/0014320037768360d53e4e935ca4a1f96eed1c896ad12

[11] *nix 命令行输入 ifconfig 、Windows 命令行输入 ipconfig 可以查看

[12] 端口一览 https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

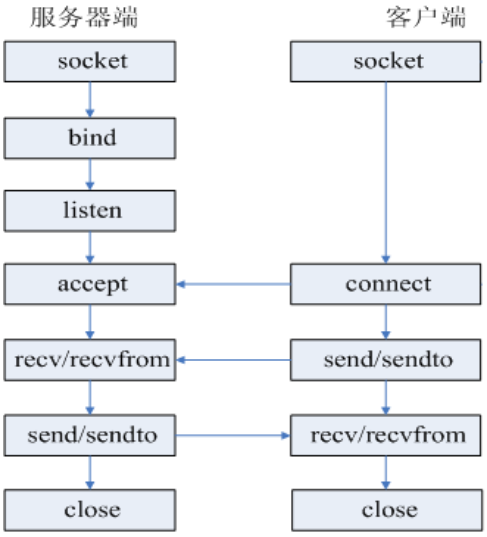
```

def socket_link(sock, addr):
    print('Accept connection from %s:%s' % addr)
    try:
        sock.send(b'Hello!')  # 服务端和客户端都可以收发数据，必须是 ascii 字符
        while True:
            msg = sock.recv(4096)  # 接收信息，buffer 大小为 4096 字节
            if(len(msg) > 0):
                print('Message from %s:%s : %s' % (*addr, msg.decode('ascii')))
            else:  # 没有收到消息就断开连接
                break
    finally:
        sock.close()  # 要保证在使用后关闭 Socket，否则该端口会一直被占用


# 指定协议为 ipv4, TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 绑定到本机（127.0.0.1）的 6666 端口（端口可以自行指定）
s.bind(('127.0.0.1', 6666))

# 开始监听
backlog = 1  # 指定最大的连接数量
s.listen(backlog)
print('Waiting for connection.')

# 接收一个新的连接
sock, addr = s.accept()

# 创建一个新的线程处理连接
t = threading.Thread(target=socket_link, args=(sock, addr))
t.start()

# 然后主线程处理其他工作
```

客户端：

```

# -*- coding: utf-8 -*-

import socket
import time

addr = ('127.0.0.1', 6666)   # 与 server 相同
```

```
s = socket.socket()

s.connect(addr)  # 如果此时连接失败会抛出异常


try:
    msg_get = s.recv(4096)
    print(msg_get.decode('ascii'))
    for i in range(10):
        msg = b'hehe'
        s.send(msg)
        print('Message to %s:%s : %s' % (*addr, msg.decode('ascii')))
        time.sleep(1)
finally:
    s.close()  # 要保证在使用后关闭 Socket，否则该端口会一直被占用
```

以上是一段可以正常工作的 Socket 通信示例代码。首先启动 server ，然后 server 会开始等待 client 连接，再启动 client，连接成功后双方就可以收发数据了。



需要注意的是如果一方一直在发送数据而另一方没有及时接收的话，那么待接收的数据就会堆积起来越来越多，混在一起。所以为了保证正确接收每一条数据，发送端在发送的时候应该主动标记每条信息的起止。

另外如果 recv 指定的 buffer 大小小于实际接收到的字符数量的话，会从接收的缓冲区中取出 buffer 数量的字符，剩下的留在缓冲区中，可以下次再接收[13]。

在标准 C++ 中，Socket 的实现是平台相关的，而且实现起来比起 Python 要复杂很多[14]。如果有需要的话，Windows 下可参考 winsock 相关内容，*nix 下可以自行查询 Linux Socket 相关内容。

[13] 也可以每次只取一个字符...

[14] 要用的时候在网上现抄代码..

2.2 JSON

由于几乎所有语言都支持 Socket，所以只要能在收发端将数据转换为字符串，就可以在任意语言实现的多个进程之间传递任何数据。

一种常用的纯文本数据格式是 JSON （JavaScript Object Notation）。

示例：

```
{
     "firstName": "John",
     "lastName": "Smith",
     "sex": "male",
     "age": 25,
     "address":
     {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber":
     [
         {
           "type": "home",
           "number": "212 555-1234"
         },
         {
           "type": "fax",
           "number": "646 555-4567"
         }
     ]
 }
```

在 Python 中只需要使用模块 `json`，然后调用 `json.dumps` 和 `json.loads` 就可以实现 Python 数据结构与 JSON 字符串之间的转换。其他语言也有很多 JSON 相关的轮子可用。

如果需要使用可以自行查阅：

w3school 的 JSON 教程 `http://www.w3school.com.cn/json/index.asp`

learn x in y minutes 的 JSON 教程 `https://learnxinyminutes.com/docs/zh-cn/json-cn/`

作业

自行查 & 抄 & 改代码，编译对应自己平台的 C++ Socket Server。用示例代码中的 Python Socket Client 与之建立连接。（作业不收，成功接收到数据就好）

（选做）通过 Socket 通信，在 Python 中向 C++ 程序发出计算斐波那契数列的请求（比如通过

Socket 发送一个数字 n，C++ 接收），C++ 程序收到后完成计算并通过 Socket 返回生成的数列[15]，并在 Python 中转换为 int 型 list。（当然还是不收...）

[15] 1, 1, 2, 3, 5, $\cdots$, Fib_n