SUPERVISOR的实现原理

为什么会出现SUPERVISOR这种进程守护工具

一、期望运行程序能够一直运行(除了本身代码错误外)

1.长期运行的进程内存泄漏导致程序退出

2.系统资源紧张导致正常运行的进程退出

3.误杀进程

二、能够方便控制任务的运行启动

1.在修改了任务代码的情况下，期望能够方便的重新启动

2.期望在不修改任务代码的情况下，收集任务的标准输入输出

3.能够获取实时运行任务的运行状态信息

需求出现后分析该问题

思考过程

1.程序运行于计算机之上，计算机可大致分为三个部分组成：计算机硬件、操作系统和应用程序。

2.待开发的工具运行于应用程序一层，此时可考虑操作系统可提供的服务。

操作系统

操作系统：作为管理和控制计算机硬件和软件资源的计算机程序，是用户和计算机的接口，也是计算机硬件和其他软件的接口。

待开发的软件需要运行在操作系统上，就必须满足操作系统作为运行任务的基本要求，即最终都是已进程的形式存在，进程就是系统进行资源分配和调度的基本单位。

对于Linux系统而言，进程在操作系统中运行时，会将待运行的进程的信息初始化到内核的PCB，PCB中包括进程状态、PID、优先级、父进程等。

操作系统在启动初始化的过程中，会初始化init进程作为所有进程的子进程或祖进程，进程都是通过系统调用fork来生成子进程。通过生成子进程的方式来独立执行相关的任务。

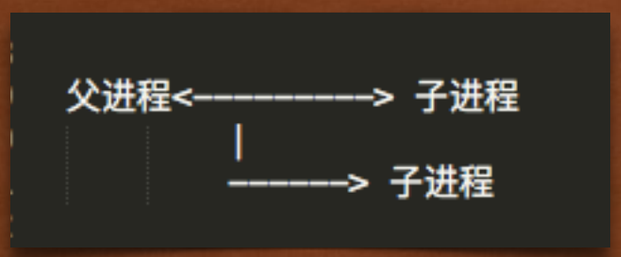
初步架构

1.此时通过fork系统调用，让子进程执行任务，达到任务代码与监控工具解耦。

2.此时任务情况划分：

父进程：主要完成对任务子进程的控制，如启动、停止、任务运行状态。

子进程：主要执行具体的任务。




```
#开发程序的架构实现

...
pid = os.fork()
if pid:
    #父进程
    #主要负责监控子进程运行工作，与客户端通信
    pass
else:
    #子进程
    #执行具体待执行的任务
    pass
...
```

父进程对子进程运行状态监控

此时已经能够在子进程中运行任务，此时如何对子进程状态进行监控呢?

参考操作系统中，父进程对子进程的监控方式，在操作系统中，通过系统调用waitpid来实现对子进程是否退出做出监控。

架构演化:

```
#开发程序的架构实现

...
pid = os.fork()
if pid:
    #父进程
    #主要负责监控子进程运行工作
    pid = os.waitpid()
    if pid:
        #重新开启一个子进程执行该任务
    #客户端通信
    pass
else:
    #子进程
    #执行具体待执行的任务
    pass
...
```

客户端通信

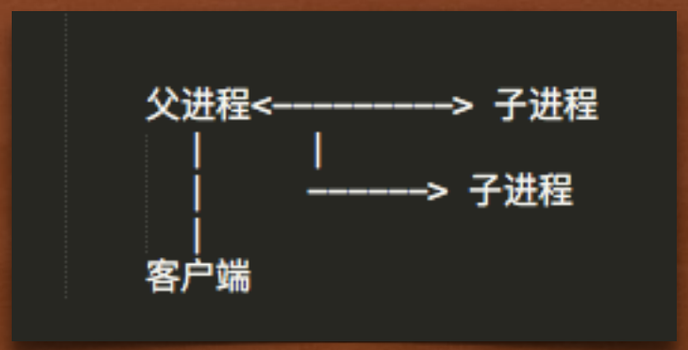
此时实现对父进程的通信，就涉及到进程间的通信。

在Linux中，进程间的通信的方式：管道、消息队列、信号和套接字等，

此时，选用比较常用的套接字通信。在套接字通信中，有很大应用层的协议可供选择如http，rpc等。

此时客户端与父进程通信，以此来控制子进程的启动、停止和获取状态信息。

此时父进程中需要实现提供套接字服务的服务器。



套接字服务器设计

1.类比于web服务器，常用的一些服务模式为多线程、多进程和异步IO。

2.在这三种方式中，异步IO凭借较好的性能和较好的性能是目前比较主流的服务端实现模式。

3.异步IO的常见的工作模式如下图所示。

```
while True:
    调用系统select函数获取当前触发对象
    r,_,_ = select.select(r,[],[],1)
    if 如果触发的事件是新请求连接:
        处理连接事件请求,
        并将新建立的连接加入到r监听列表中
    elif 如果触发是读事件:
        处理读事件
    elif 如果触发是写事件:
        处理写事件
    else:(如果错误等其他事件)
        其他事件处理
```

软件框架改进

1.此时主要的思路已经如右图所示。

2.添加使程序成为后台程序的函数daemonize。

3.客户端程序不再详细描述。

```
#开发程序的架构实现
...

# 该函数需自行实现，实现将该任务设置成后台任务
daemonize()

pid = os.fork()
if pid:
    #父进程
    while True:
        #主要负责监控子进程运行工作
        pid = os.waitpid()
        if pid:
            #重新开启一个子进程执行该任务

        #客户端通信
        #调用系统select函数获取当前触发对象
        r,_,_ = select.select(r,[],[],1)
        if 如果触发的事件是新请求连接:
            处理连接事件请求,
            并将新建立的连接加入到r监听列表中(客户端连接)
        elif 如果触发是读事件:
            处理读事件(如启动、停止某个子进程)
        elif 如果触发是写事件:
            处理写事件(将处理结果返回)
        else:(如果错误等其他事件)
            其他事件处理

else:
    #子进程
    #执行具体待执行的任务
    pass
...
```

分析一下DEMO代码