操作系统不相信任何任何上层用户，操作系统将整个操作系统的功能暴露了一个窗口，让我们调用---系统调用

Task\_struct中有哪些属性？

Task\_struct内存级数据对象—与磁盘和可执行程序关系不大。

Task\_struct是Linux内核的一种数据结构，它会被操作系统在内存创建

内存指针:包括程序代码和进程相关数据的指针，还有和其他进程共享的内存块的指针 • 上下⽂数据:进程执⾏时处理器的寄存器中的数据[休学例⼦，要加图CPU，寄存器]。

• I∕O状态信息:包括显⽰的I/O请求,分配给进程的I∕O设备和被进程使⽤的⽂件列表。

• 记账信息:可能包括处理器时间总和，使⽤的时钟数总和，时间限制，记账号等

运行起来的程序就是一个进程

1把程序运行起来，双击or./exe-🡪本质在系统中启动了一个进程

进程分类

执行完就退出---ls pwd等指令

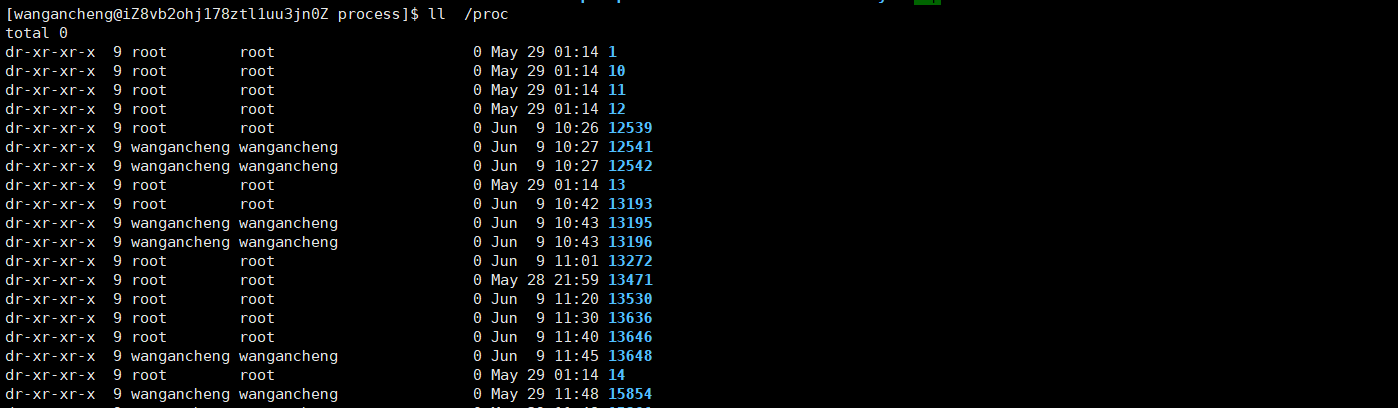
一直不退，直到用户退出---常驻进程，一直在后台挂着，不关就不退

同一个进程在不同时间点启动进程，进程的pid 会发生变化，因为操作系统维护进程pid是用一个累加计数器，来进行pid增长，你在启动进程的同时，操作系统中也有其他进程启动，所以pid变化且不连续很正常

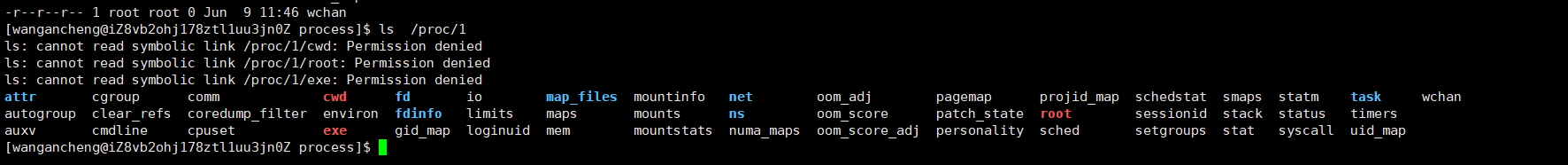
Pid保证进程的唯一性

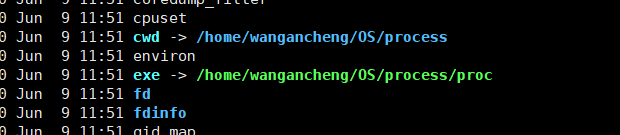
Kill -9 obj\_pid 杀死进程

Proc目录详细的介绍了所有进程的详细信息，当进程被创建的时候，那么在proc目录下j就会以他的pid为名创建一个目录，目录内是他详细的属性信息



每一个数字就是一个进程，每一个进程内部他的所有内容就是他的属性

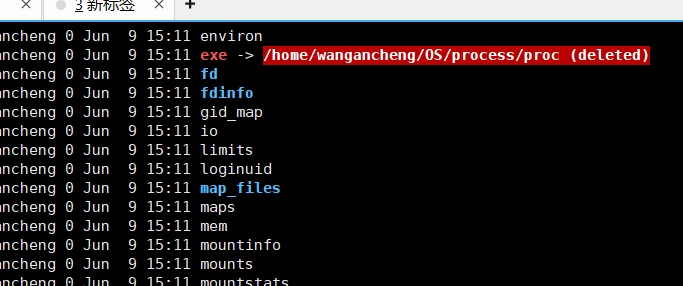




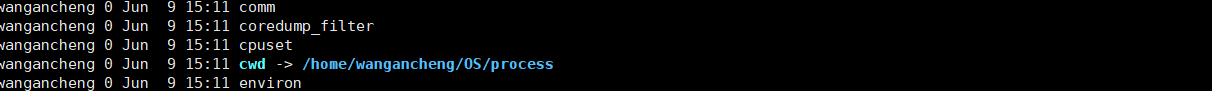
Exe🡪链接文件指向进程的可执行文件在磁盘的位置

为啥make clean后该进程还能运行🡪因为可执行程序已经被加载到内存了，要是杀死该进程在运行就运行不了了

删掉磁盘上的可执行程序exe会指向形成这个进程的原可执行文件+原路径并显示已经被删除

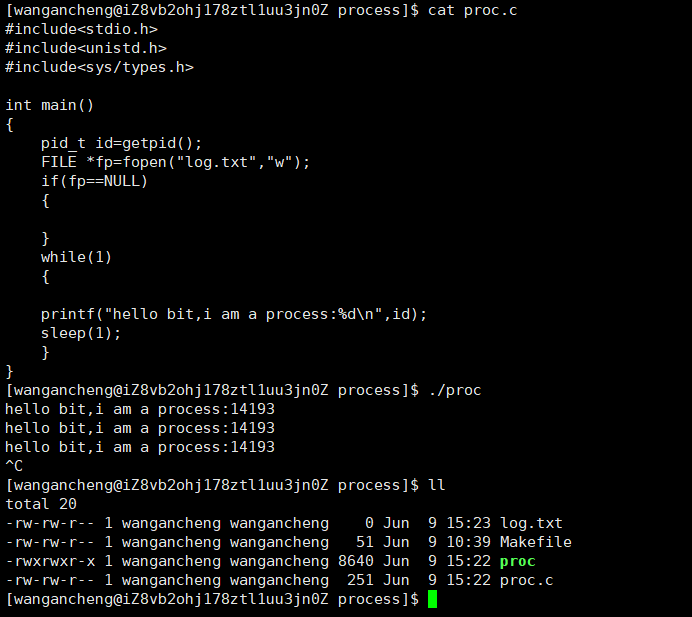


理解Cwd



cwd -> /home/wangancheng/OS/process

----当前工作目录

在执行fopen时在当前路径创建---所谓的当前路径指的是进程cwd指向的路径

Windows同理

操作系统为我们提供的查看进程接口就是proc

Ps命令底层就是打开proc目录，对于proc目录内容做文本分析实现 ps命令

Proc目录不是磁盘级的文件

也就是说当操作系统启动，创建很多进程，并遍历所有进程的pcb,边遍历，边构建内存级的文件系统。Proc是文件把内存中进程的属性以文件的形式展现出来了，让我们看到了，但是没有在磁盘保留任何信息—将内存中的数据以文件的形式呈现

Ppid(父进程)

在Linux系统中，启动之后，新创建任何进程时，都是有自己的父进程创建的

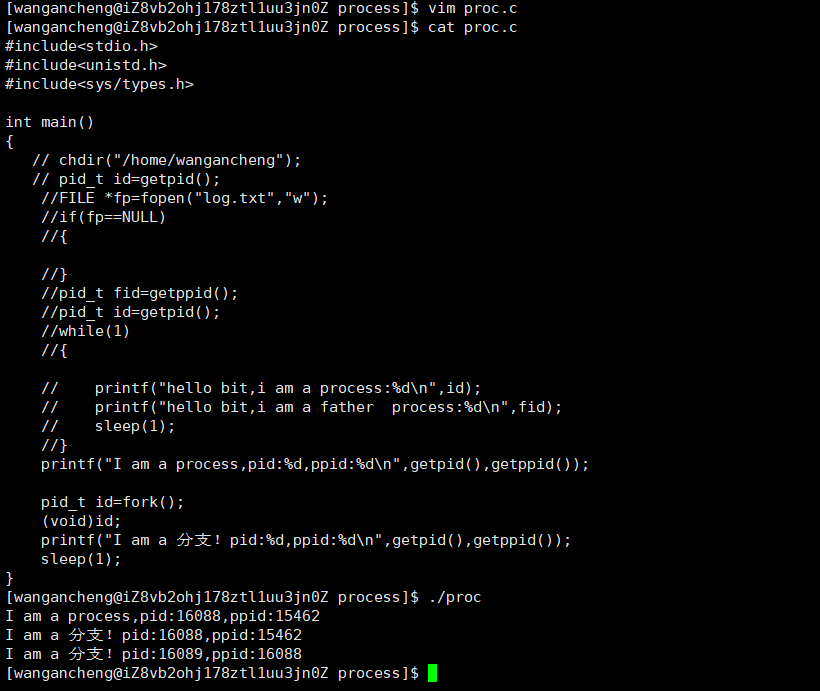
进程是父进程指导操作系统建的

命令行中，执行命令/执行程序，本质是bash的进程，创建的子进程，由子进程执行我们的代码

Bash-命令行解释器—shell 是所有命令行解释器的通称---linux的命令行解释器就叫bash

在命令行上启动的所有的进程的父进程都是 bash

Bash会对我们的命令进行解释—我们执行的程序都是bash的子进程由bash创建出后，bash 读取程序将程序加载到内存里，帮我们创建子进程由子进程执行我们的代码

用系统调用来创建进程

为啥第一条打印语句只有一条？

第二条打印语句有两条？

因为进程刚启动只有一个进程，经过fork(),程序的执行分支就不是一条执行分支了，经过fork()他有两条执行分支，在系统层面上两条执行分支都要跑

Fork之后有程序有两条分支一条是调用fork 的原进程（父进程），另一个通过新创建的执行流是子进程

Linux所有的进程都是树形结构

Fork()🡪两个进程🡪父子关系🡪一般而言，在fork()之后代码是会共享的，但是数据是各自私有一份的

子进程获得整个程序的代码副本，但​**​执行起点是fork()返回处​**​，不会重新执行fork()之前的代码。您当前的代码结构完全正确地区分了父子进程行为。

***为啥父子进程的数据是各自私有一份的***

进程有一个比较大的特点有很强的独立性

多个进程之间，运行时，互不影响，即便是父子

进程有

1. 代码（只读），进程实现独立性不影响
2. 数据上---如果改了就会产生影响---所以让各自私有一份

Pid\_t id=fork();

1 id是不是一个变量，是

2．返回的本质，不就是向指定变量进行写入返回值数据

3打印的本质是读取这个变量

理解创建子进程—结合系统接口上

一个函数怎么会有两个返回值

一般创建子进程的时候，子进程的task\_struct是拷贝自父进程的

并个性化一些属性

不是fork 之后代码就有两个分支了

其实当我们准备return 的时候，父子进程已经运行了，只是在系统调用没返回

Return 是一条代码

Fork()之后代码共享，数据各自有一份。

当执行了return ，父进程执行一次，子进程执行一次

1Fork() id的返回之前给父(subpid)子（0）

2，fork 会有两个返回值---为什么

3．一个变量？怎会会同时有不同的值呢？

为啥因为需要进行独立

--它是怎么做到的

Fork()有两个执行分支，他们两个谁先运行？？

谁先运行由操作系统自主决定