等待disk磁盘---阻塞等待的一种—不可中段睡眠，深度睡眠主要用于磁盘io。

僵尸状态—有一个大爷，突然倒路边了，这时候你出于人道主义，你给120打了电话

给120打完电话之后，120发现人没了，给110打电话，110来了-

会有这个流程

1大爷倒下—2法医检测---3 110抬走---4后续。为啥要进行法医检测—要通过法医检测给家属（父进程）和社会（操作系统）一个交代。

1-3状态是僵尸状态，4是后续

为啥要有僵尸状态—维持进程退出信息，方便父进程和操作系统来进行查询！

为啥要有僵尸状态？

当linux操作系统创建一个进程，这个进程一定是一个子进程，进程被创建是为了完成任务，我们的父进程如何知道子进程完成任务的情况呢—---当一个进程退出，父进程或者操作系统通过获取进程退出信息，来检测子进程对于任务的完成情况---为了支持检测一个进程退出不能叫死亡—而是僵尸状态—当父进程或操作系统拿去他的退出信息之后，他就处于一种死亡状态

一个进程退出代码将不再执行---但这个进程不会被彻底释放掉，他会将自己的管理信息如pcb维护起来，以方便我们通过pcb来查看进程的退出信息，一般一共进程创建的时候进程=内核数据结构（task\_struct）+I代码+数据|可执行程序

一个进程退出，代码不会再执行，（一个进程在创建时会创建pcb 并从磁盘中加载代码和数据）所以代码和数据就不需要存在了，所以首先代码和数据可以立即释放，进程退出是要有退出信息的，（进程退出码,int code ,其他），进程退出时将进程退出信息，保存在自己的task\_struct 内部。一个进程再退出的时候会把main（）函数的返回值保存在task\_struct内部。

进程退出时，代码和数据可以释放，管理结构task\_struct必须被维护起来，方便用户未来进行获取进程的退出信息getpid(){return curr->pid}

一个进程在创建的时候是先加载代码和数据还是先创建内核数据结构？？

我觉得是先加载代码和数据，我觉得他的内核数据结构里包含了有关代码和数据的属性

妈的，年年考年年错

先创建内核数据结构-

类比高考后填完志愿—档案被学校提走先创建管理信息。

Task\_struct创建了这个进程就已经存在了，没加载代码和数据也没事，只是这个进程不会被调度

操作系统释放的时候是先释放代码和数据，维护task\_struct ---僵尸状态

我想看到僵尸状态如何设计 代码 创建子进程-》父子同时存在，让紫禁城退出，父进程还活着，但是什么也不做

Defunct无效的---

僵尸进程

如果没有人管僵尸进程，僵尸进程会一直僵尸下去，task\_struct会一直持续下去—一直消耗内存—就会产生内存泄露----后面：一般需要父进程读取子进程信息，一旦读取完子进程才会自动退出。

**#include<stdio.h>**

**2 #include<stdlib.h>**

**3 #include<unistd.h>**

**4**

**5 int main()**

**6 {**

**7 pid\_t pid=fork();**

**8 if(pid==-1)**

**9 {**

**10 perror("fork fail ");**

**11 exit(1);**

**12 }**

**13 if(pid==0)//子进程**

**14 {**

**15 int cnt=10;**

**16 while(cnt)**

**17 {**

**18 printf("我是子进程，我的pid是%d\n,ppid是%d",getpid(),getppid());**

**19 }**

**20 }**

**21 else{**

**22**

**23 while(1)**

**24 {**

**25 printf("我是父进程，我的pid是%d\n,ppid是%d",getpid(),getppid());**

**26 }**

**27 }**

**28 }**

如果用户层我malloc new 出来的空间我没free ,当进程退出后，是否会发生内存泄露问题？？new出来的空间是否还在

我觉得不会，不在了，因为我总干

因为程序分为代码和数据，我们自己new/malloc出的空间属于数据空间，属于将来要保存的数据，属于数据范畴，当一个进程退出时，这些数据空间就会被释放，包括在你的程序里malloc 出的空间，所以写一个死循环一直malloc,系统越来越卡了，直到我把程序给退了，系统就不卡了，内存占用率立马就降低了

此时有一个程序永远不退出，他一直malloc但是用户一直不手动free,这样就会导致申请内存空间不被释放，系统存在很多启动就不退出的软件（杀毒软件，听歌软件）常驻内存进程，如果这样软件一直malloc内存就会导致系统可用内存越来越少

语言层面的内存泄露，不怕一个进程一跑起来，就退出的，最怕语言层面的内存泄露在常驻内存的进程中出现，影响会比较大。

我们将来写的软件都是常驻内存的，所以要主要内存泄露。

6.26

孤儿进程

父进程退出，子进程在

进程A的父进程是bash，进程B的父进是A

当杀掉A后为啥A不会变成僵尸状态

因为A的父进程是bash, A被杀死后立刻被bash回收了

子进程B父进程pid变成1,因此不用担心，父进程退出后，子进程会被系统领养，待子进程退出后还会有人回收----被领养的进程B叫做孤儿进程

僵尸进程为啥不被领养？？  
因为僵尸进程的父进程还在，只是腾不出手来回收

父子两个进程的关系，如果父进程先退，子进程就会被领养

如果父进程还在，子进程退出，子进程就会退出。

一个进程被领养了，他就转到后台运行了，啃臭c就杀不掉他了

进程优先级

进程优先级是什么，为什么，怎么办

是什么？  
获得某种资源的先后顺序，打饭排队的本质就是在确认优先级，资源就是打饭大姨的饭

为啥要有优先级

本质其实就是目标资源比较少

如 cpu,磁盘，显示器，键盘虽然看着不少但是进程更多啊

优先级vs权限

权限解决的是能不嫩的问题

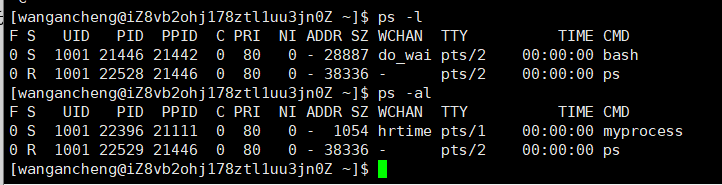
优先级：是已经能了，就是谁先谁后的问题。

优先级这个东西怎么做的

进程有task\_struct---task\_struct中有优先级的属性，在linux内核中我们用一个或者多个特定的整形变量来表示优先级。

在linux中有个特点，优先级数字越小，优先级越高—提升优先级，把数字变小.

Ps-l 可以看到进程优先级



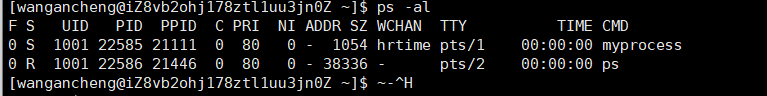
但是不用-al 为啥就看不到我们刚刚正在运行的myprocess

这是因为在用户层，我们启动了一个xshell，myprocess

在这个bash上，我们又启动了一个xhell进程在这个进程上我们只启动了一个bash,用ps -l 只能看到启动bash 和ps进程，要想看整个系统层面的进程要用ps -al

Pri当前进程的优先级

NI:nice:优先级的nice数据---优先级的修正数据



最终优先级pri=pri(最老的)+nice（细微的）---进程优先级重置

为啥不直接改pri而是要整一个修正数据，因为如果一个进程正在调度，把优先级调高来，他算高还是算低，不好判断

因此我们用nice值来修改优先级，等操作系统再次调度该进程的时候，就会更新该进程的进程优先级，然后再对该进程进行调度时，

UID表示当前进程的用户id

在linux系统里，系统除了给用户分配用户名还会分配数字（用户id）,操作系统标识用户用的是uid

操作系统不仅要记录文件是谁创建还要记录这个进程是谁启动的

权限控制

1文件会记录下拥有者，所属组，和对应的权限

2.linux下一切接文件

3．所有操作，都是进程操作，进程自己会记录谁启动的我，文件会记录谁创建的我

当进程进行与文件相关的操作的时候----就会对比文件的拥有者，所属组和进程uid从而使权限控制问题得到解决。

进程记录下来了我，知道我是谁，文件也记录下了我和其他人知道我和其他人是谁，你进程要访问文件，操作系统会检测uid是不是相等，在不在所属用户和用户组，

有无读写权限，无读没写直接删掉

进程的优先级往往指定的是进程竞争cpu的资源

如何修改优先级

1指令

2代码

修改进程优先级不是一个特别高频的动作，而且不建议修改

指令

Top ->r->按照提示修改即可（os禁止频繁修改，没有权限修改）nice>=-20,nice<=19

Nice可取40个数字

为啥nice值在可控范围内？

分时系统—进程调度时要尽量公平，因为在可控范围内才不会影响我系统的公平调度，如果不加控制的话，当用户把一个进程的优先级调的特别高就会产生进程饥饿现象

进程切换

1. 概念储备

A 在现代操作系统里每一个进程都有自己的时间片，时间片到了，进程就要被切换

B linux是基于时间片调度轮转的，

C 一个进程在时间片到来的时候，并不一定跑完了，可以在任何一个地方被重新调度切换。

进程如何切换？？

感性理解进程切换

在校入伍

给学校打招呼---保留学籍

在部队历练

回到学校---给学校打招呼—恢复学籍

进程就是我

选去当兵就是时间片到了

给学校打招呼---保留学籍—保留上下文

回到学校---给学校打招呼—恢复学籍---回复上下文

6/28打卡

你的进程在运行时，会有很多临时数据，临时数据和内存中的堆，栈，函数调用没关系，在内存里就和调度没关系，内存是保存数据的地方和调度没关系，和调度强相关的是cpu,在进程运行的时候会有很多临时数据·，当前代码执行到了第几行，正在计算的是while判断还是加减乘除，这些都在cpu寄存器中保存。

Eax

Ebx

Edx

……

Eflag

Ecs

Eds

Egs

Cr0

Cr1

Cr2

Ebp

Esp

Eip(pc指针)

Cpu中会有很多寄存器，这些寄存器cpu在未来调度进程的时候，会把进程产生的临时数据全部放在寄存器里，然后在寄存器里帮你计算，再帮运算结果返回内存中的进程，

系统在调度这个进程的·时候，系统怎么知道进程代码执行到了第几行，怎么知道。

Cpu怎么知道我们要读取哪条指令呢？

栈帧的概念

为啥cpu能知道程序执行到哪里呢，是因为cpu中的一个寄存器eip,eip保存了当前正在执行指令的下一条地址，ir:指令寄存器，保存的就是正在执行的指令

**1. 程序计数器（Program Counter, PC）​​**

​**​作用​**​：​**​指向下一条待执行指令的内存地址​**​，是CPU执行流程的“向导”。  
​**​关键特性​**​：

* ​**​自动递增​**​：每执行一条指令后，PC值自动指向下一条指令（地址增加指令长度）。
* ​**​跳转控制​**​：遇到分支指令（如jmp、call）时，PC会被修改为新的目标地址。
* ​**​硬件实现​**​：由专用寄存器存储，通常在CPU时钟驱动下更新。

**指令寄存器（Instruction Register, IR）​​**

​**​作用​**​：​**​临时存储当前正在执行的指令机器码​**​，供CPU解码和执行。  
​**​关键特性​**​：

* ​**​临时存储​**​：保存从内存中取出的指令二进制数据（如0xC7 0xF8）。
* ​**​解码输入​**​：指令解码器（Decoder）读取IR内容，解析出操作码（Opcode）和操作数。
* ​**​只读性​**​：执行过程中IR内容不变，直到下条指令加载。

Cpu内部的寄存器的数据是进程执行时的瞬时状态信息数据--------进程的上下文数据

Cpu内有很多个寄存器我们称为一套寄存器，寄存器！=寄存器里的数据(当前进程的上下文)

如果当一个正在执行的进程被切走了，在当前系统当中cpu会保留进程的临时数据，

如果当一个正在执行的进程被切走了，在当前系统当中cpu会保留进程的临时数据，为啥要保存

当一个正在执行的进程被切走时，CPU保存其临时数据（主要是寄存器状态）​**​至关重要且必要​**​，原因如下：

1. ​**​维持进程执行的连续性：​**​
   * 进程在运行时需要依赖CPU内部的各种寄存器：
     + ​**​程序计数器：​**​ 指向下一条要执行的指令地址。
     + ​**​通用寄存器：​**​ 存储指令操作数、中间计算结果、地址等临时数据。
     + ​**​状态寄存器：​**​ 包含关于最近操作结果的信息（如零标志、进位标志）、中断使能状态、执行模式等。
   * 如果不保存这些寄存器的值，当另一个进程开始运行时，它会覆盖掉所有这些寄存器的原始状态。
   * 当操作系统​**​稍后​**​决定让被切走的进程​**​重新获得CPU时​**​，它需要能够让这个进程​**​从中断的地方继续精确无误地执行​**​。如果寄存器的值丢失了，该进程将：
     + 找不到正确的指令（程序计数器丢失）。
     + 丢失关键的中间计算结果（通用寄存器被覆盖）。
     + 丢失状态信息，可能导致后续逻辑出错（状态寄存器被覆盖）。
   * 保存这些“临时数据”使得操作系统能够完整恢复进程的​**​执行现场​**​。

进行切换的核心—》进程上下文数据的保存与恢复