对于一个进程他刚被加载到内存，就处于一种新建状态---进程准备好了需要被调度，就处于一种就绪状态，进程在cpu上运行—运行状态，代码中有scanf 用户没有输入，--等待io请求-----阻塞状态-----用户输入数据---就绪状态

补充知识

并行和并发

电脑单核—操作系统同时可以启动多个进程

怎么做到的呢？？

时间片

Linux/windows民用级别的操作系统都是分时操作系统

—调度任务追求较为公平（常适用于任务没有明显的优先级）

进程具有独立性

等待的本质：连入目标设备

实时操作系统

新能源汽车实时和分时都打开

对于每一个CPU操作系统都要给CPU 提供一个运行队列（runqueue）

操作系统内核中提供的一个

Struct Runqueue

{  
 int nums

//其他属性

Task\_struct\*head;

}

什么叫一个进程处于运行状态呢，本质上就是一个进程在运行队列中，就称该进程处于运行状态，进程已经可以准备好了可以被CPU随时调度了

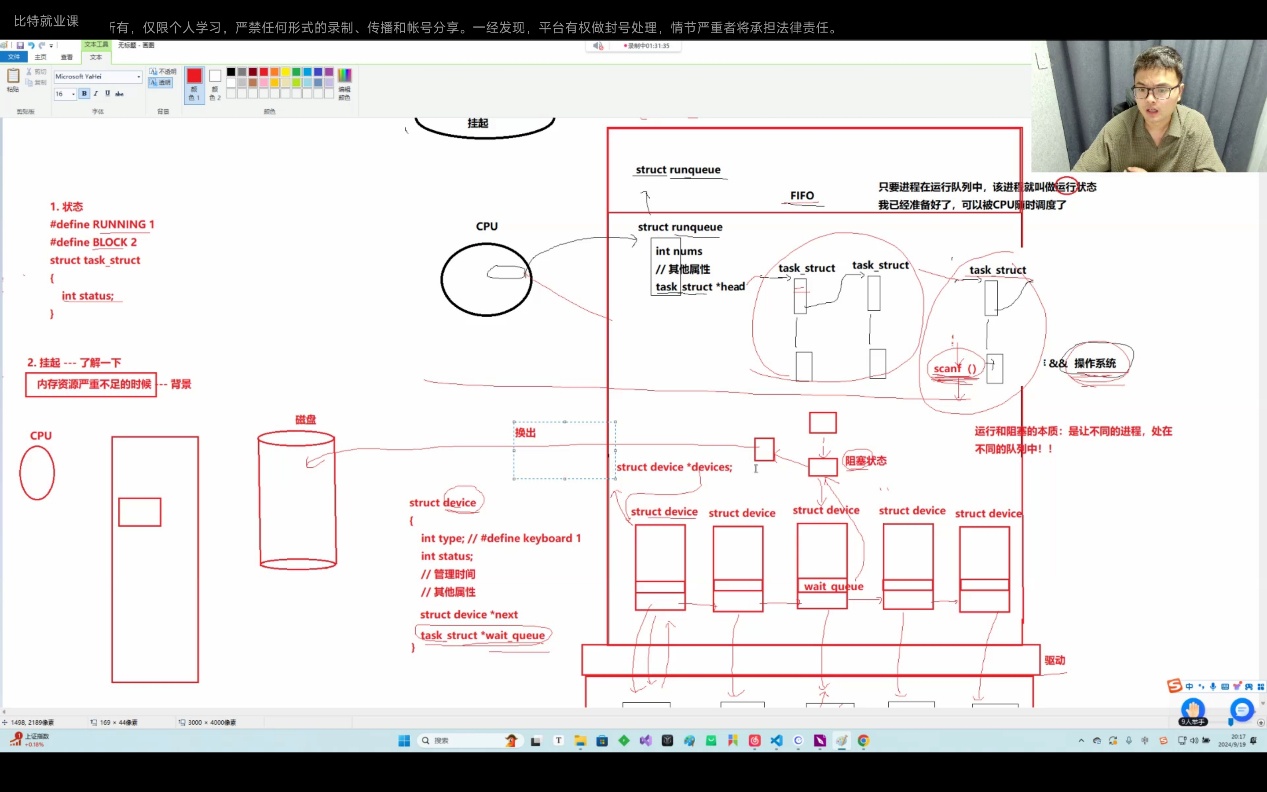
当然呢，操作系统底层包括硬件，cpu内存，底层设备是冯诺依曼结构构成的

运行和阻塞的本质是·1让进程处于不同的队列中

1，状态

阻塞挂起等待

挂起—了解

当进程处于阻塞状态且内存空间严重不足的时候，操作系统·会将阻塞进程的数据和代码全部换出到磁盘的swap 分区当进程获得资源后，再将进程的代码和数据换入到内存

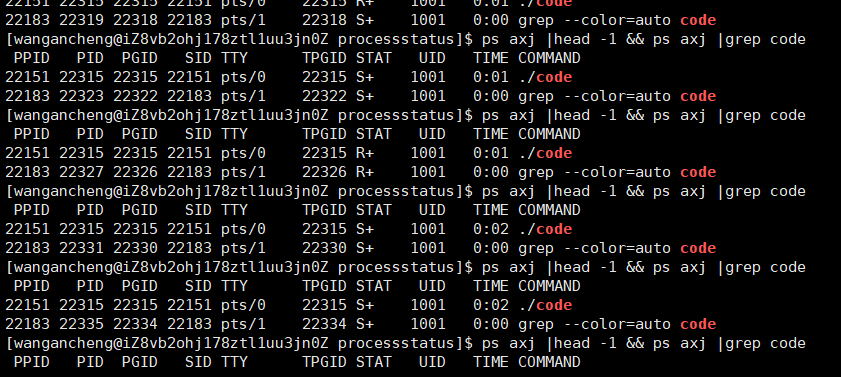
Swap分区 用时间换空间

一般swap分区的功能会被禁掉

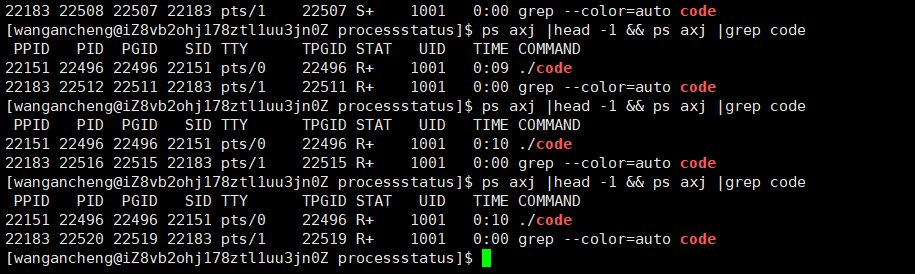
换入换出的本质是做IO，和外设做交换，就会慢

App闪退---被操作系统杀掉了

有打印



无打印



一个进程的时间片是非常小的根据冯诺依曼体系Printf不是直接将内容打印到显示屏上而是，将打印内容写到显示器的缓存，由缓存显示到显示器，如果我们疯狂去写，缓存被写满，会导致显示器不是时时刻刻就绪，虽然，每一次都从显示器看到了打印的具体内容，但是因为printf,--有io—所以这份代码看似在死循环，但是99%的情况，他大部分时间是在做io的，为啥—因为io速度非常慢，所以大部分时间都在做io,所以当他真正while检测条件，执行代码printf时才是r状态。

磁盘睡眠—深度睡眠

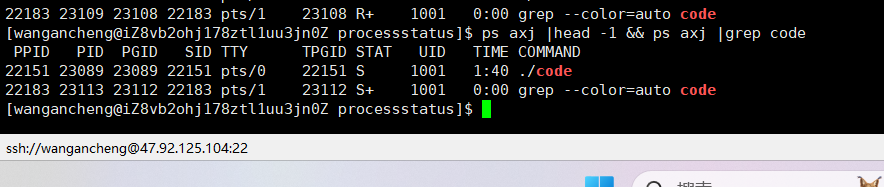
90%健康操作系统d状态很难出现---瞬时状态，未来在公司里查到一个d状态,一般查到d状态，磁盘/操作系统已经快挂掉了

Stop状态也是为了等待某个条就绪

有些情况下，有些进程，在特殊的应用场景下，不允许在显示器上打印

你非要打，操作系统也不能杀掉这个进程，就可能将这个进程暂停了

--也就是进程做了非法但是不致命的操作，被os 暂停了



前台进程可以用啃臭+c杀掉

但是后台进程只能通过 kill -9杀掉

./code &

为啥要把一个任务放在后台---在linux搞一个下载的任务i或者是一个比较耗时的任务，但是不想让他阻塞你的命令行，把这个任务启动起来，让系统自动完成，我继续进行命令行操作

Windows下载时将下载页面最小化，手机app不点退出，上划使app到后台运行。

后台进程如果和终端脱离，他在进行向终端写入操作的时候，不就暂停了吗？？

因为没有脱离

第二种暂停状态

给程序打断点的本质是让进程停下来

---追踪暂停

Gdb—将gdb进程启动，2一旦输入r gdb进程就开始追踪另一个进程了

遇到断点程序被暂停

进程死亡

进程最开始为啥被创建？--一定是基于某种应用，完成用户任务·

进程退出—系统特层面任务完成情况如何看 ？

通过进程执行的结果，告知父进程/操作系统，我把任务完成的如何

$?是最近一个程序执行完，退出的时的退出码

0表示成功

非0 程序执行出错

Ls C语言写的有main函数

Main函数有返回值

Main函数执行完函数退出---有返回值—为了告诉操作系统告诉父进程我的进程执行任务情况

进程退出不会立即死亡---先进入Z状态—操作系统和父进程来查看退出信息-再进入X

为啥要有Z状态---维持退出信息，方便父进程和操作系统来进行查询

Kill -19 pid ---使进程暂停

Kill -18 pid---使进程继续

Stop状态也是在等待某个条件就绪

在有些情况下，有些进程不允许在特殊应用场景下不允许在显示器上写入，把一个进程变成后台进程或者守护进程的时候，进程和终端脱离---不要这个终端，进程非得要用printf往终端上打印，操作系统把这个进程给暂停

进程暂停第一种情况—进程做了非法但是不致命的操作

暂停后重新启动---进程去后台运行—只能用kill -9 pid

./code & 任务自动到后台

为啥要有一些进程到后台执行

比如执行一些比较耗时的下载任务我不想让他阻塞我的命令行---因此把改进程转到后台

Gdb打断点的本质是进程暂停了

用gdb 进程是用gdb追踪代码进程

当一个进程被追踪时断点停下，进程状态就是t;

进程死亡—进程为什么被创建—进程创建出来是为了完成任务---进程退出任务完成情况怎么样—站在操作系统的角度----如何知道排序结果是正确的---通过进程执行的结果来告诉父进程我把任务完成的如何，

操作系统创建进程来完成任务，进程执行任务，操作系统如何知道进程将任务完成的怎么样了，进程退出时要把进程退出的痕迹记录下来方便我上层得知，比如在系统执行

Ls code

对于 ls进程的父进程就是bash 也要知道子进程执行情况

$? 最近一个退出程序他所对应的退出信息。----echo $?

Main函数执行完·程序结束---进程退出---MAIN函数返回值是为了告诉操作系统进程执行结果是否正确？？

在进程角度父进程要关心子进程的执行结果的，因为父进程创建子进程是为了执行任务—我需要知道进程完成的怎么样

**核心机制：等待队列 (Wait Queue)​​**

1. ​**注册等待项**​  
   当进程/线程需要等待某个事件（如数据到达、资源就绪、锁释放等）时：
   * 将自身加入与事件关联的**内核等待队列**​（wait\_queue\_head\_t）。
   * 设置自身状态为 TASK\_INTERRUPTIBLE（可中断睡眠）或 TASK\_UNINTERRUPTIBLE（不可中断睡眠）。

**操作系统中的实现依赖**

1. ​**交换守护进程 (kswapd)​**​  
   Linux内核线程，监控内存压力并触发挂起操作。
2. ​**调度器钩子函数**​  
   在 schedule() 中检测内存水位，决定是否调用 suspend\_process()。
3. ​**信号机制**​  
   SIGSTOP（强制暂停）/ SIGCONT（继续执行）实现人工挂起控制。

**总结：挂起等待的本质**

挂起等待是操作系统在**极端资源竞争下的一种生存策略**​：

* ​**目标**​：牺牲部分进程的响应延迟，换取系统整体稳定性（避免OOM崩溃）。
* ​**代价**​：恢复时的磁盘I/O导致显著延迟（不适合实时任务）。
* ​**核心价值**​：通过内存-磁盘的二级存储体系，​**扩展虚拟资源边界**，使系统能在有限硬件下承载更多任务。

在Linux中，​**可中断睡眠（TASK\_INTERRUPTIBLE）​**​ 和 ​**不可中断睡眠（TASK\_UNINTERRUPTIBLE）​**​ 是进程的两种核心休眠状态，其本质区别在于 ​**能否被外部信号（Signal）中断**。