进程

进程是什么?

一个已经加载到内存中的程序，叫做进程

正在运行的程序，叫做进程

理解OS管理进程的一般思路

OS需要将多个进程管理起来，先描述后组织

任何一个进程，在加载到内存时，形成进程时，OS要先创建描述进程（属性）的结构体对象——PCB（进程控制块，即进程属性的集合，本质是struct，因为OS是用C实现的）

属性包括：进程编号、进程状态、优先级。。。

OS要干啥?

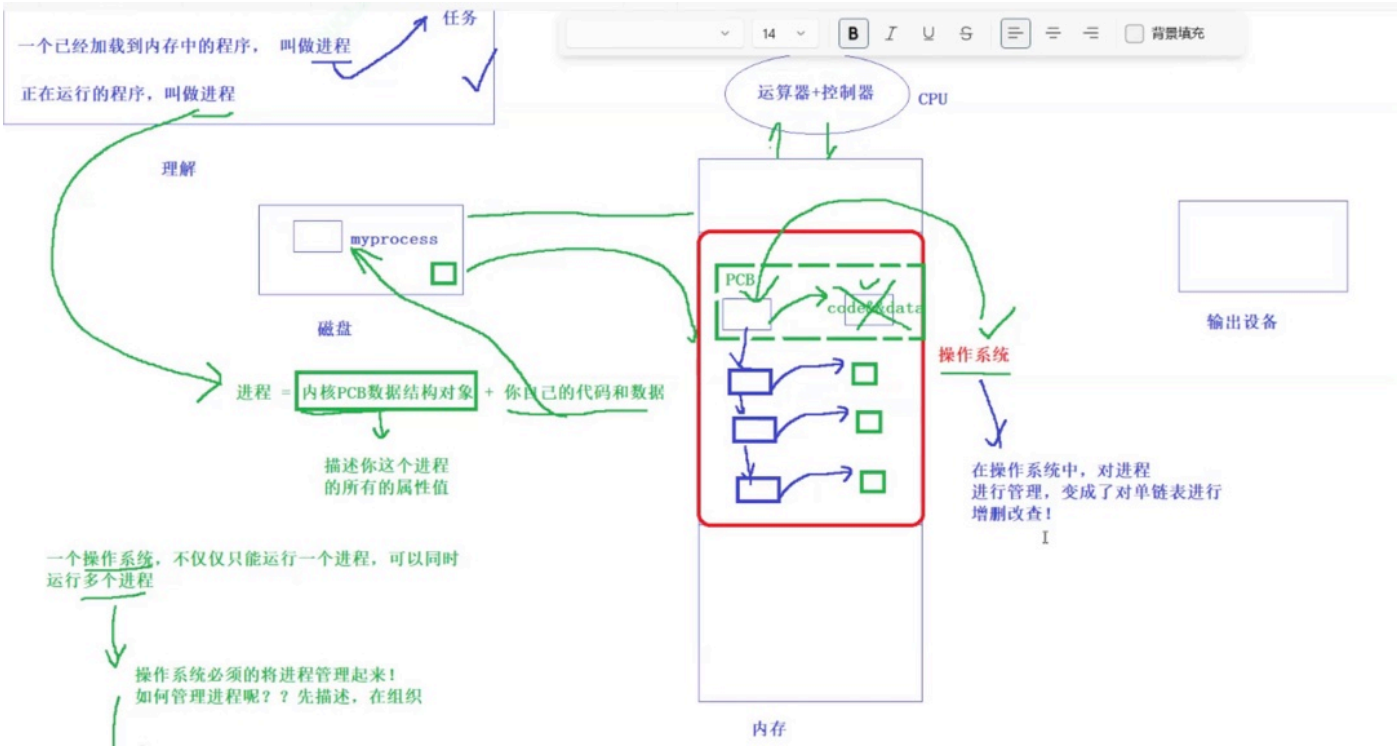
根据PCB类型，创建PCB对象——教务信息

进程数据加载到内存——人来学校

两件事和起来形成一个进程

进程 = 内核PCB数据结构对象 + 自己的代码和数据

在OS中，管理进程本质就是对PCB对象形成的单链表进行管理



Linux具体是怎么做的?

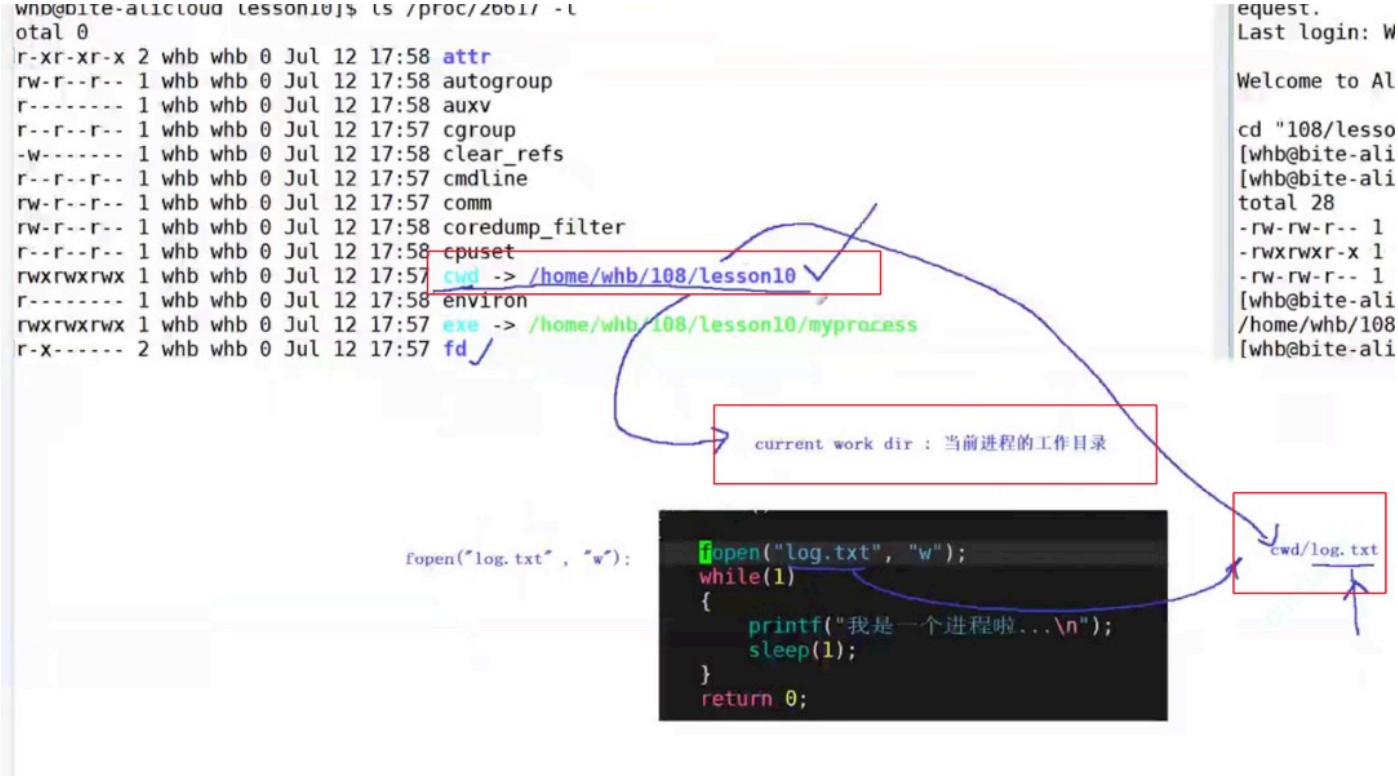
Linux下的PCB是：task struct，里面包含进程的所有属性

- 标示符: 描述本进程的唯一标示符，用来区别其他进程。
- 状态: 任务状态，退出代码，退出信号等。
- 优先级: 相对于其他进程的优先级。
- 程序计数器: 程序中即将被执行的下一条指令的地址。
- 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。
- 其他信息

linux内核中，最基本的组织进程task struct的方式，采取双向链表组织的

为什么touch文件时，默认在当前目录下创建文件?

因为命令本身就是一个进程，进程PCB中留存了进程的工作目录



进程概念

PID与PPID

查进程

ps ajx (| grep ...)

终止进程

kill –9 PID

系统调用接口：getpid(),获取当前进程的pid

PPID ： 父进程的pid

监视命令：

```
[whb@bite-alicloud lesson11]$ while :; do ps axj |head -1 && ps axj | grep pr
oc | grep -v grep ; sleep 1 ;done
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
```

bash进程是命令行的父进程

bash通过创建子进程来进行命令解释，自己又去接受用户的新命令

创建子进程

验证父进程子进程同时存在

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    printf("begin: 我是一个进程, pid: %d, ppid: %d\n", getpid(), getppid());

    pid_t id = fork();
    if(id == 0)
    {
        // 子进程
        while(1)
        {
            printf("我是子进程, pid: %d, ppid: %d\n", getpid(), getppid());
            sleep(1);
        }
    }
    else if(id > 0)
    {
        //父进程
        while(1)
        {
            printf("我是父进程, pid: %d, ppid: %d\n", getpid(), getppid());
            sleep(1);
        }
    }
    else
    {
        //error
    }
```

fprk()之后变成了两个进程

```
[whb@bite-alicloud lesson11]$ ./proc
begin: 我是一个进程, pid: 1849, ppid: 16815
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
我是父进程, pid: 1849, ppid: 16815
我是子进程, pid: 1850, ppid: 1849
```

```
[whb@bite-alicloud lesson11]$ ps axj  |grep 16815
16815  4889  4889 16785 pts/30    4889 R+    1003   0:00 ps axj
16815  4890  4889 16785 pts/30    4889 R+    1003   0:00 grep --color=auto 16815
16807 16815 16815 16785 pts/30    4889 S     1003   0:00 bash
```

一开始bash和普通函数一样加载成一个进程，执行fork之后形成一个分支

创建进程的方式

1. ./ *普通进程*
2. fork() *创建子进程*

问题

1. 为什么fork要给子进程返回0，给父进程返回子进程的pid

为了区分，让不同的执行流执行不同的代码

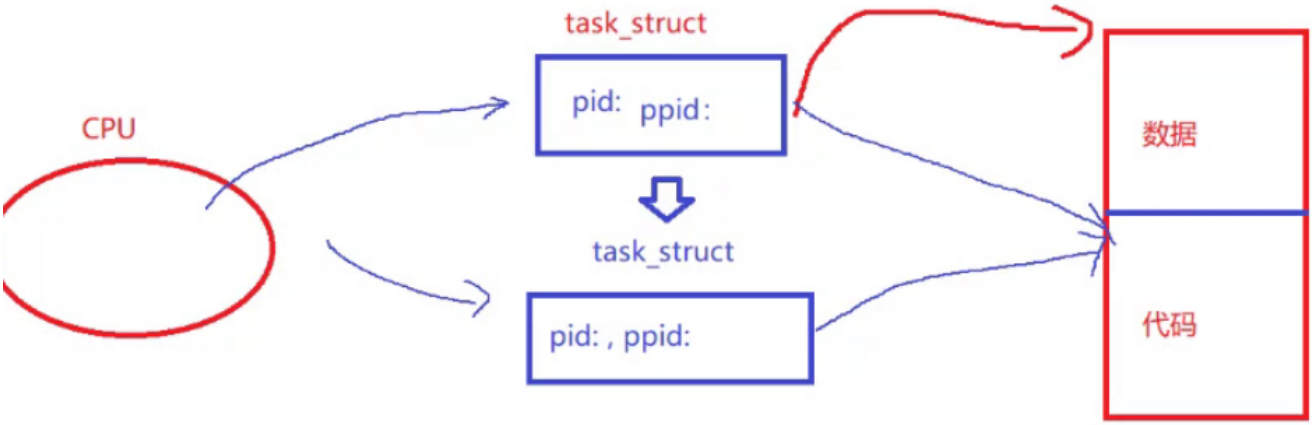
fork之后的代码父子进程共享

因为一个父有多个子，一个子只有一个父，想要通过父找到子，需要唯一标识，即返回子进程的pid

进程 = 内核数据结构+代码数据

创建子进程，本质就是系统中多了一个进程

子进程没有自己的数据和代码，只能跟父进程共享



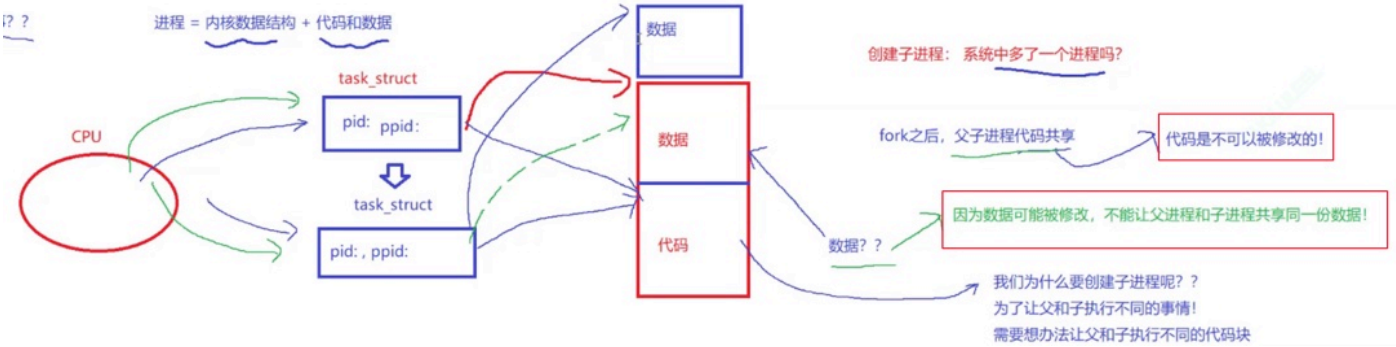
为什么要创建子进程：为了让父和子执行不同的代码块

2. 一个函数是如何做到返回两次的？如何理解？

fork也是个函数，有自己的代码块，return之前已经将子进程创建出来了！

3. 为什么id有两个值？

进程运行有独立性，一个变化不会影响另一个进程，这就决定了数据是不一样的



有相当一部分数据子进程不一定访问全部数据，甚至完全不做修改，所以全部拷贝优点浪费

要修改时，重新开一份空间，在新空间内写入，即数据层面的写时拷贝

当fork函数内部return时，也是写入，而id时父进程的数据，此时发生id的写时拷贝

fork之后，谁先运行，有调度器决定，不确定的

调度器保证进程运行时间相对平均

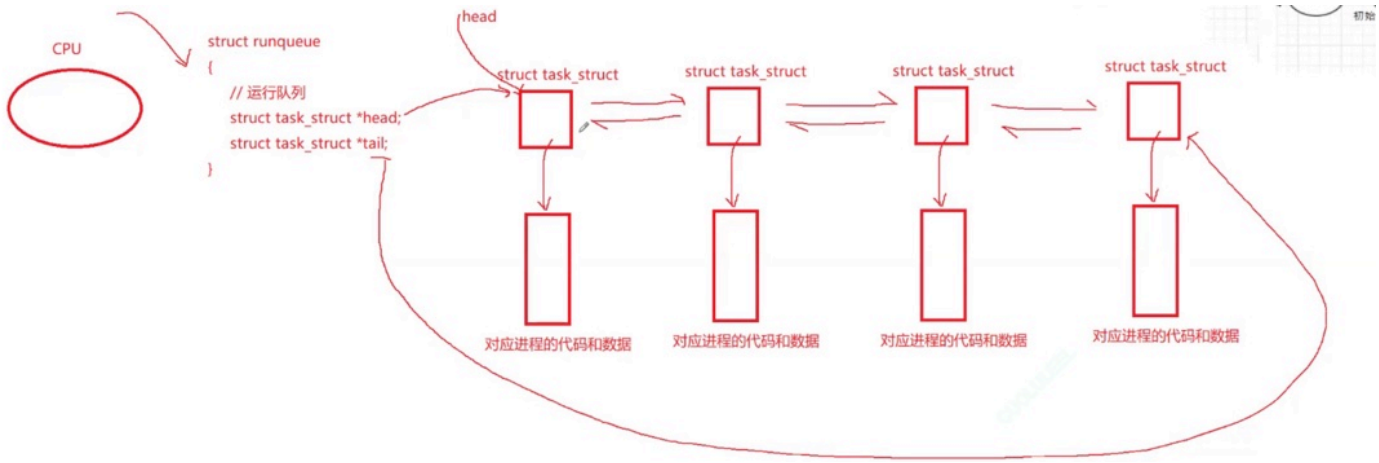
进程状态

操作系统学科下的进程状态

运行状态 阻塞状态 挂起状态

运行状态

CPU内部维护一个运行队列，在运行队列上的状态就是运行状态，即r状态

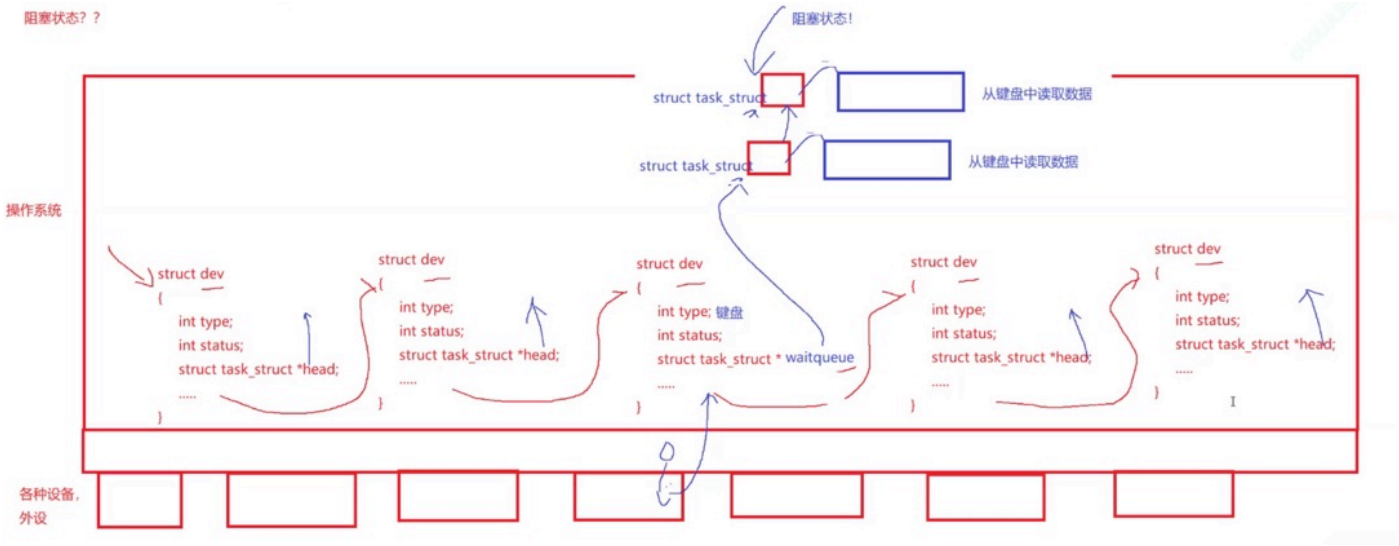


一个进程只要把自己放到CPU上开始运行了，不是执行完毕才把自己放下来！比如进行死循环时，其他进程仍在执行。如何做到的？每个进程都有一个时间片的概念，到了一定时间换下一个。

在一个时间段内，所有进程都会被执行，叫做并发执行。发生频繁的进程切换

阻塞状态

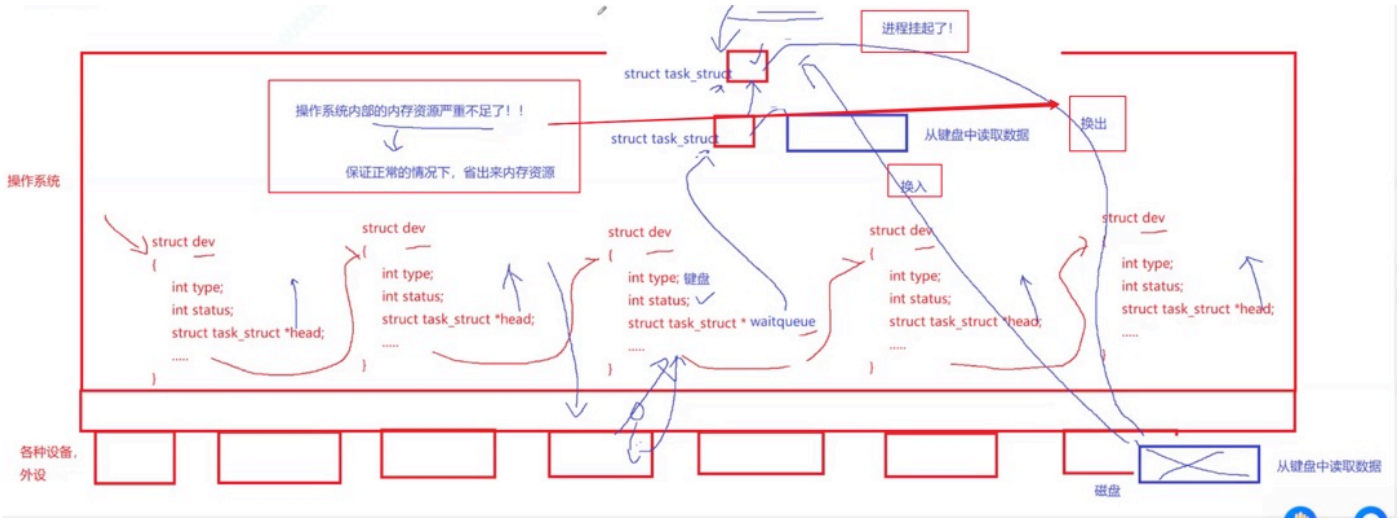
管理外设，对每个devic,都有一个等待队列，拿键盘举例，当一个进程要从键盘中读取数据，但我迟迟没输入，改进程进入键盘设备对应的等待队列，此时其处于阻塞状态，输入之后进入运行队列，执行后从硬件中读取数据



等待队列很对，但运行队列一个CPU只有一个

挂起状态

当进程处于阻塞状态时，数据与代码是空闲的，如果此时操作系统内的内存资源严重不足了，进程发生换出，变为挂起状态



具体的Linux状态如何维护?