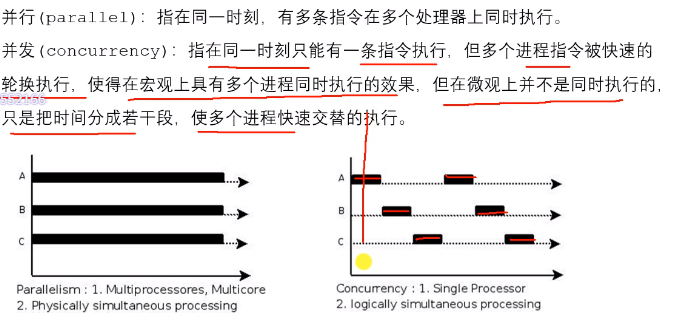
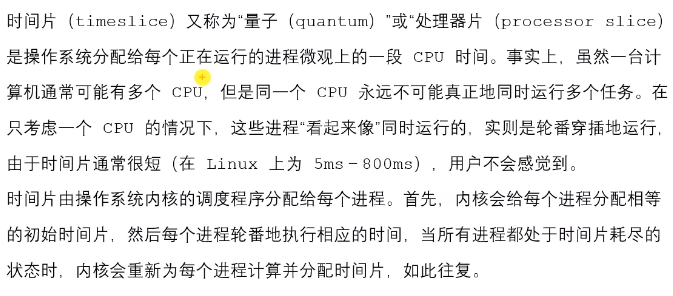
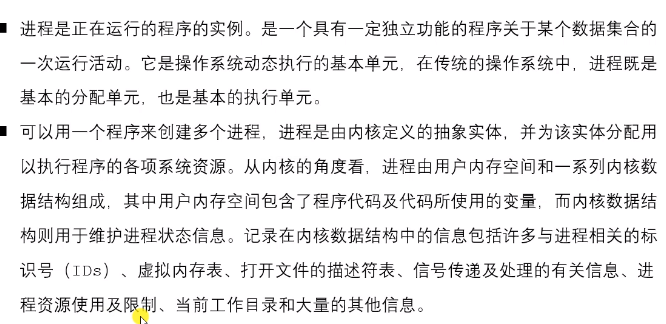
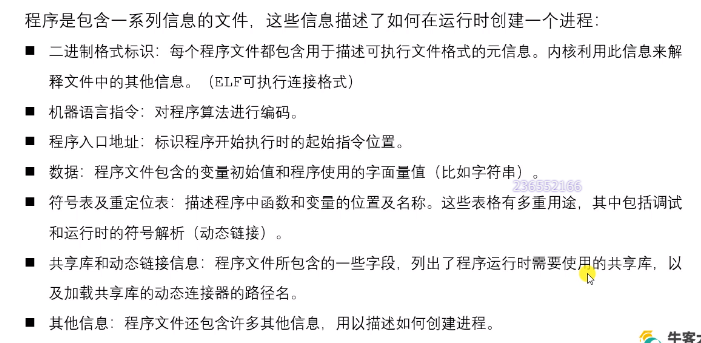
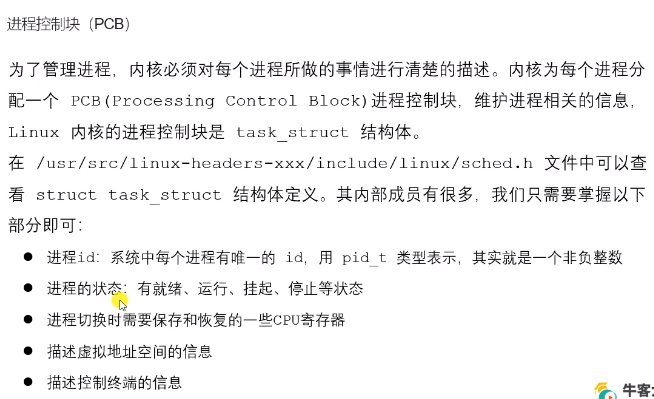
一、进程概述



并行：不同实体（多个CPU上）；并发：同一实体（单个CPU上）



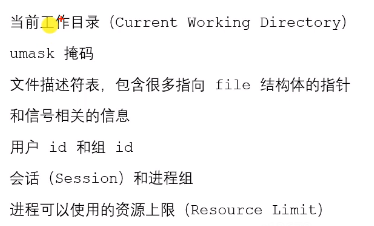
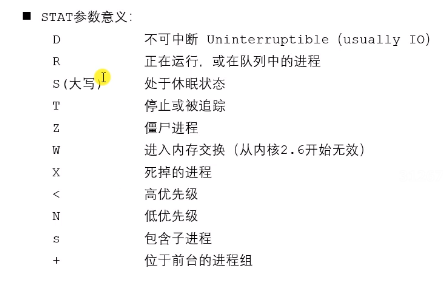
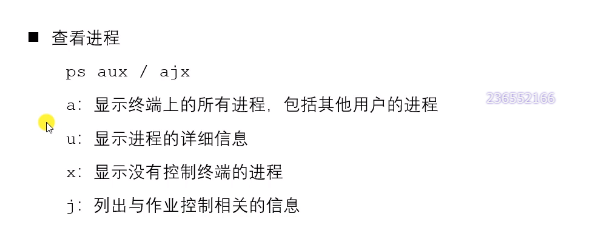
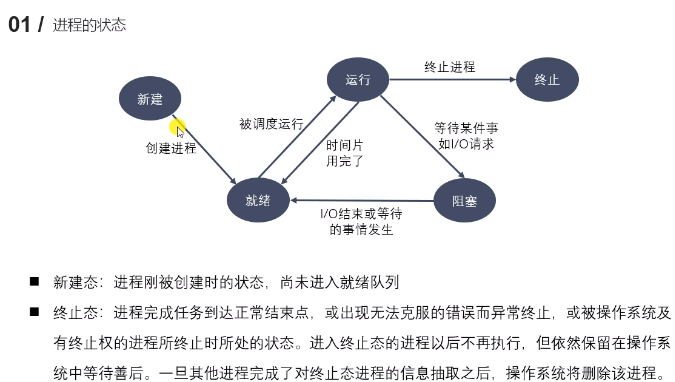
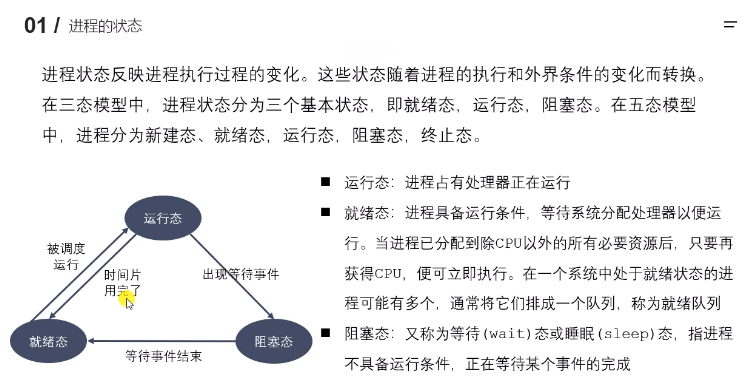
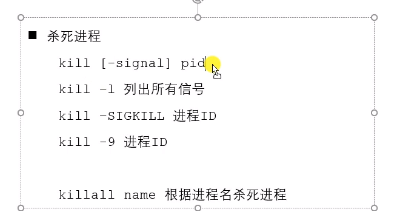
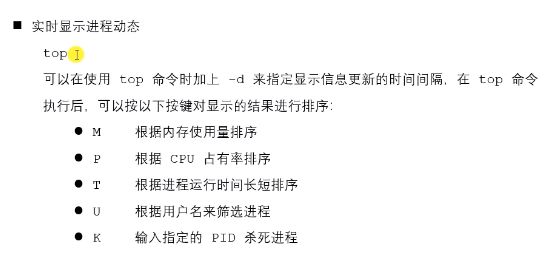


Table 1：ulimit指令 ——> ulimit –a：资源上限

二、进程状态转换

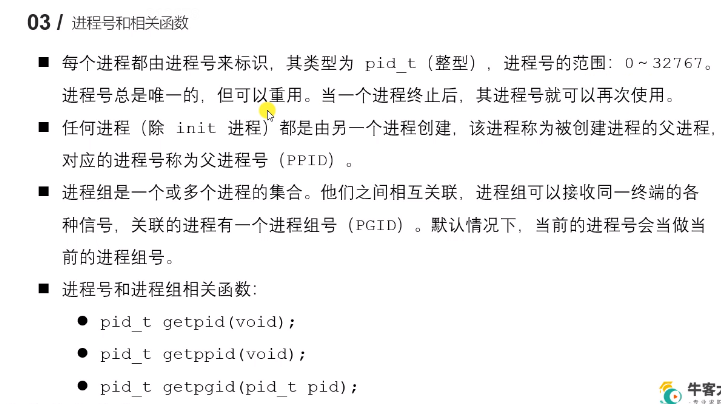


指令：tty：查看当前终端。

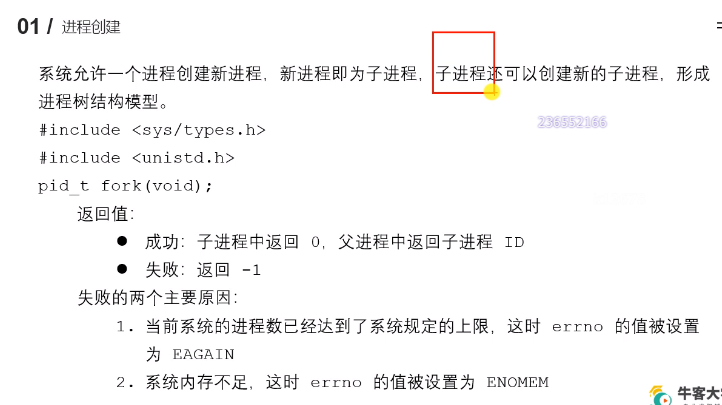


后台运行：./test &。

./test进程的父进程是当前终端。



三、创建进程



Q：为什么父子进程都要执行一次main()函数里的代码？

在一个典型的多进程程序中，当父进程调用fork()系统调用创建子进程时，子进程将会继承父进程的代码段、数据段和堆栈等信息。这意味着子进程在创建后会具有与父进程相同的代码和执行流程。（内核的PID不被改变；父子进程拥有相同的内存；栈空间里的变量一开始是共享的，一旦修改，就不共享。即读时共享，写时拷贝）

因此，无论是父进程还是子进程，它们都会从main函数开始执行，并按照代码的顺序逐行执行。这是因为操作系统将创建的子进程看作是一个全新的进程，并没有意识到该进程是通过复制父进程而来的，因此它会从main函数开始执行。需要注意的是，虽然父进程和子进程在执行相同的代码，但它们是独立的执行体，各自有自己的进程上下文和资源。子进程的执行不会影响父进程的执行，它们是并发执行的。这样的设计使得多进程编程更加灵活，可以实现并行处理和任务分配等功能。

四、父子进程虚拟地址空间

Linux的fork函数使用写时拷贝。内核此时并不复制整个进程的地址空间，而是父子进程共享一个地址空间。在需要写入的时候复制地址空间。资源的复制只有在写入的时候进行，其他时间通过只读方式共享地址空间。Fork之后父子进程共享文件。文件描述符指向相同的文件列表，引用计数增加。

Q：堆空间和栈空间的区别？

堆和栈是计算机内存中两种不同的存储方式，它们在内存管理和数据存储上有着一些重要的区别。

1. 分配方式：

- 栈（Stack）：栈的内存分配是自动的，由编译器进行管理。每当函数被调用时，函数的局部变量和参数会被分配到栈上，并在函数执行完毕后自动释放。栈采用先进后出（FILO）的方式进行内存分配和释放。

- 堆（Heap）：堆的内存分配是手动控制的，程序员需要显式地申请和释放内存。堆上的内存分配不会自动释放，需要程序员负责管理。如果不适当地使用或忘记释放堆上的内存，可能会导致内存泄漏。

2. 空间大小：

- 栈：栈的大小是固定的，通常较小。栈的大小在程序编译时就确定，并且在运行时不可改变。栈的大小受限于操作系统和编译器的限制，一般几个MB到几十MB不等。

- 堆：堆的大小通常比栈大得多。堆的大小受限于系统内存的总量，可以动态地分配和释放内存。

3. 存储内容：

- 栈：栈用于存储局部变量、函数参数、函数调用和返回值等短期存储的数据。它的生命周期与函数的执行周期相关，随着函数的调用和返回而动态变化。

- 堆：堆用于存储动态分配的数据，例如通过malloc()或new操作符在堆上申请的内存。堆上的数据可以在程序的不同部分共享和访问。

4. 内存管理：

- 栈：栈的内存管理由编译器自动完成，无需程序员干预。栈上的内存分配和释放是快速且高效的，但是其生命周期受到限制。

- 堆：堆的内存管理需要手动进行，程序员需要负责显式地申请和释放内存。堆上的内存管理相对复杂，容易出现内存泄漏和内存访问错误的问题。

栈适合存储短期的局部变量和函数调用相关的数据，内存管理由编译器自动完成；而堆适合存储动态分配的数据，需要程序员手动管理内存。

五、GDB多进程调试



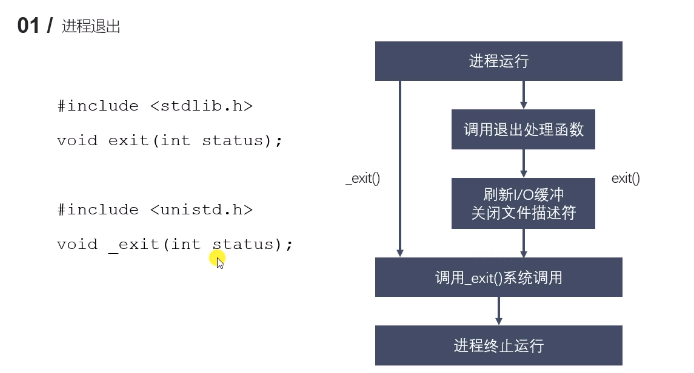
六、exec函数族：通过调用exec，在一个进程中去执行另一个程序。

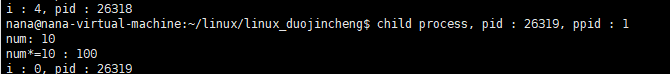
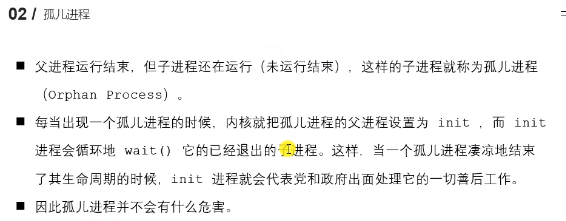


上图是孤儿进程。

。

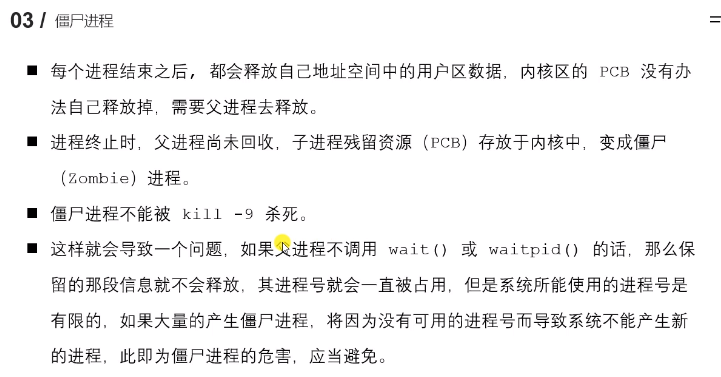
七、进程退出、孤儿进程、僵尸进程？





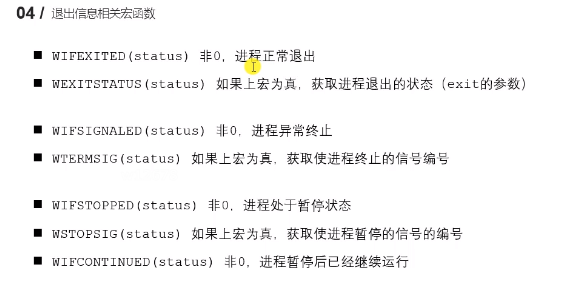
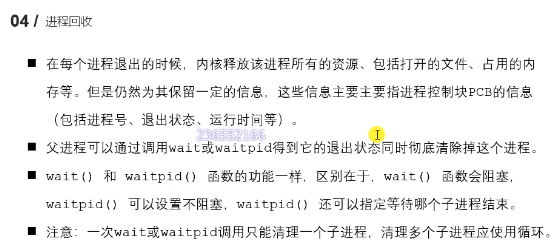
孤儿进程的父进程被设置为init，init的pid是1。资源回收。

父进程结束后，会切换到当前终端（父进程的父进程）。因为子进程和父进程共享内核区的文件描述符等一些资源，所以子进程的输出也是在当前终端输出。

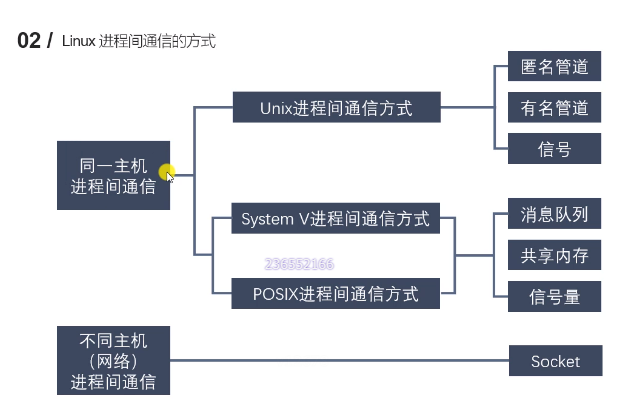
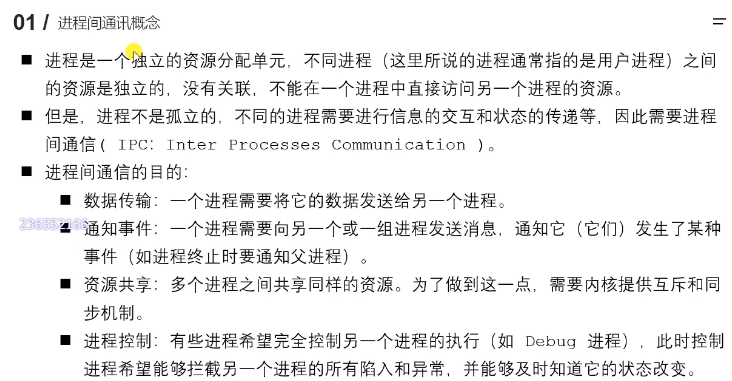


僵尸进程不可以通过kill -9 命令杀死；可以通过ctrl+C杀死进程（信号）。但是可以用kill -9 命令通过杀死父进程来杀死子进程。僵尸进程的清理工作需要由其父进程负责。父进程可以通过调用 wait() 或 waitpid() 等函数来获取僵尸进程的退出状态，并释放僵尸进程所占用的资源。

八、进程回收：wait()、waitpid()。



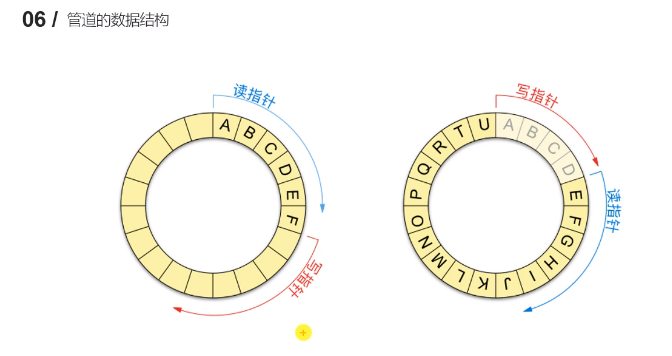
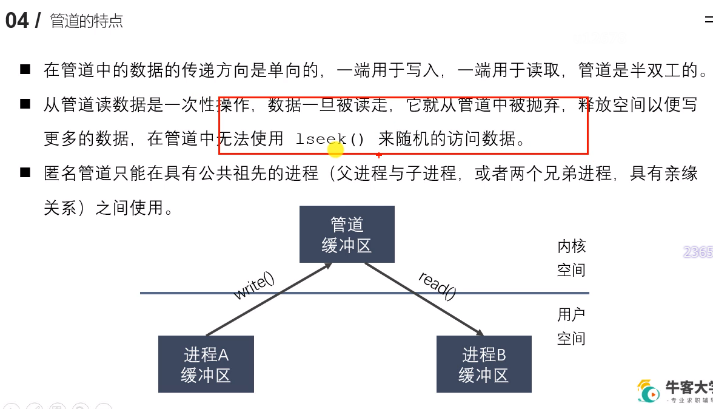
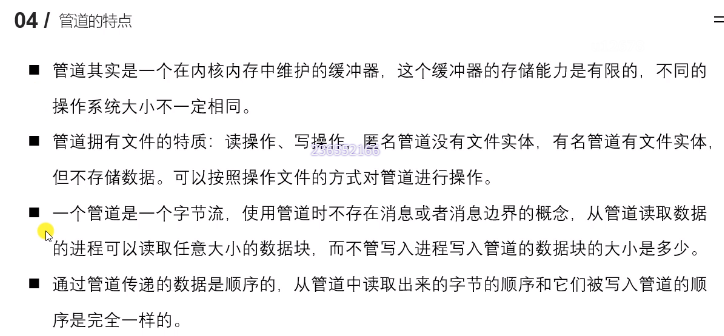
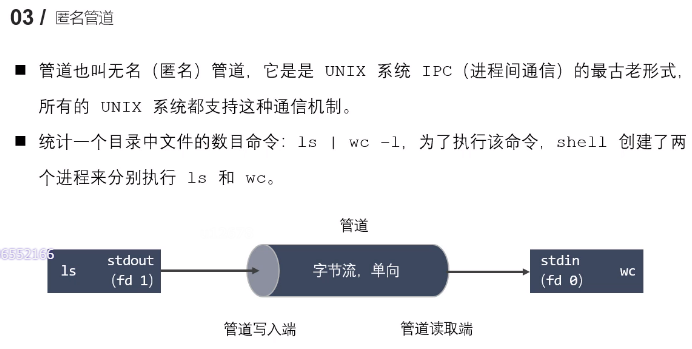
十、进程间通信



除了共享内存，内存映射也可以完成进程间通信。

十一、匿名管道

在fork函数之前创建管道：对应同一个文件描述符。



管道的数据结构：逻辑上环形的队列。（循环队列）。

十二、父子进程通过匿名管道通信



十四、管道的读写特点

管道的读写特点：

使用管道时，需要注意以下几种特殊的情况（假设都是阻塞I/O操作）

1.所有的指向管道写端的文件描述符都关闭了（管道写端引用计数为0），有进程从管道的读端读数据，那么管道中剩余的数据被读取以后，再次read会返回0，就像读到文件末尾一样。

2.如果有指向管道写端的文件描述符没有关闭（管道的写端引用计数大于0），而持有管道写端的进程也没有往管道中写数据，这个时候有进程从管道中读取数据，那么管道中剩余的数据被读取后，再次read会阻塞，直到管道中有数据可以读了才读取数据并返回。

3.如果所有指向管道读端的文件描述符都关闭了（管道的读端引用计数为0），这个时候有进程向管道中写数据，那么该进程会收到一个信号SIGPIPE, 通常会导致进程异常终止。

4.如果有指向管道读端的文件描述符没有关闭（管道的读端引用计数大于0），而持有管道读端的进程也没有从管道中读数据，这时有进程向管道中写数据，那么在管道被写满的时候再次write会阻塞，直到管道中有空位置才能再次写入数据并返回。

总结：

读管道：

管道中有数据，read返回实际读到的字节数。

管道中无数据：

写端被全部关闭，read返回0（相当于读到文件的末尾）

写端没有完全关闭，read阻塞等待

写管道：

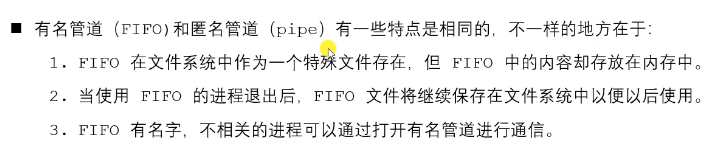
管道读端全部被关闭，进程异常终止（进程收到SIGPIPE信号）

管道读端没有全部关闭：

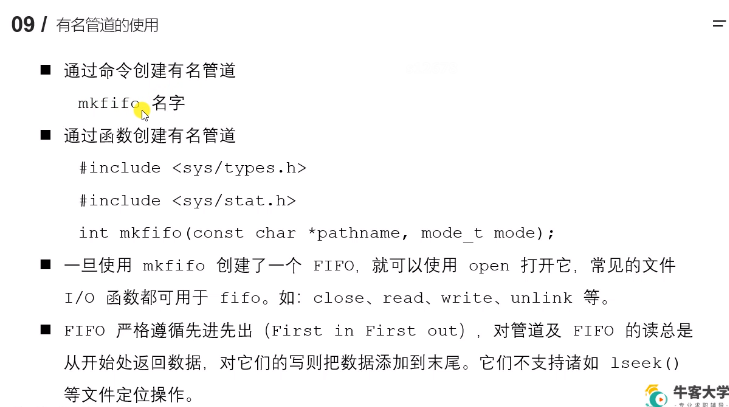
管道已满，write阻塞

管道没有满，write将数据写入，并返回实际写入的字节数。

十五、有名管道







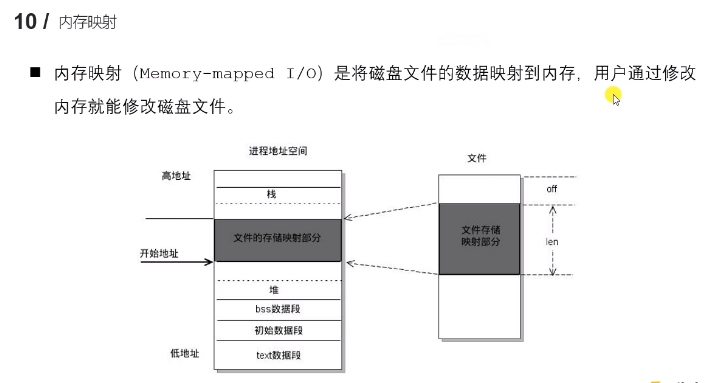
十七、内存映射：效率较高

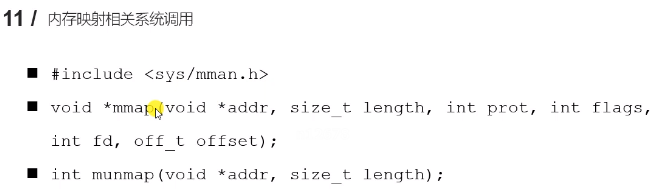
内存映射通过使用mmap()系统调用来创建映射，并将文件或其他设备的内容映射到一块内存区域。（虚拟地址空间的共享库）

多个进程可以独立的访问同一个文件的映射，但每个进程都有自己的虚拟地址空间和缓存副本。

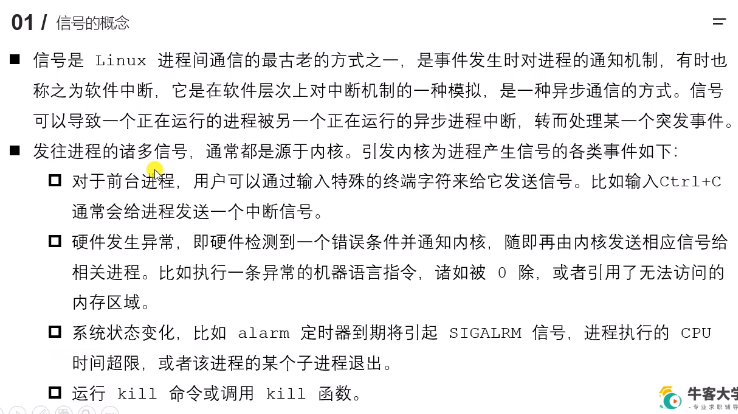
内存映射适用于频繁访问大文件或需要在多个进程之间共享数据的场景。

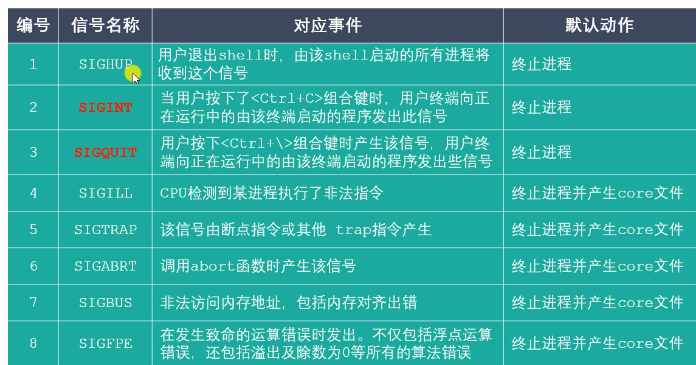
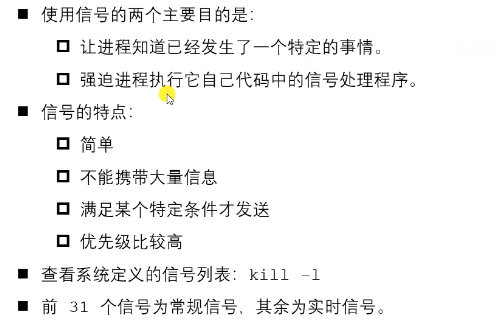
内存映射需要进行文件的读写操作，可以持久化的保存数据。

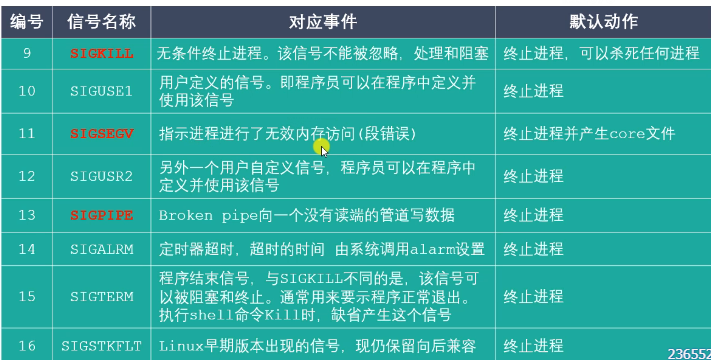


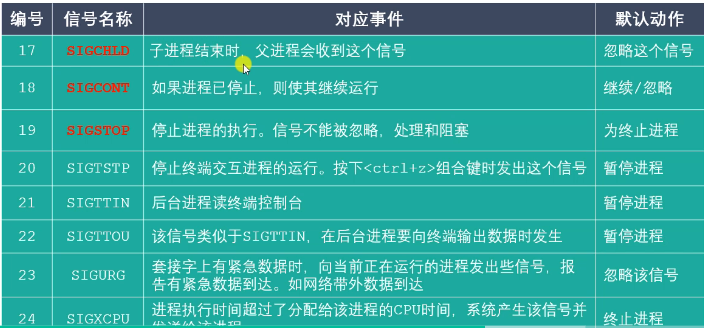


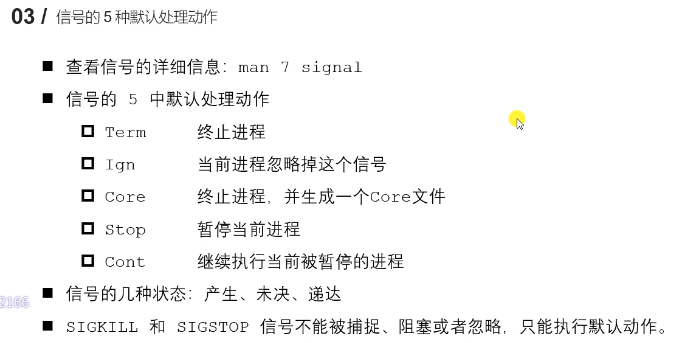
十九、信号



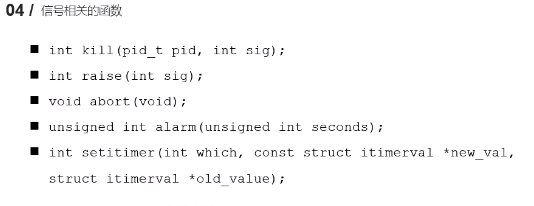




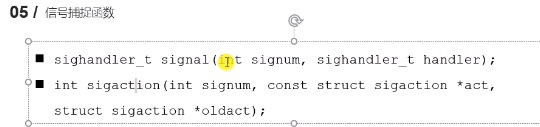




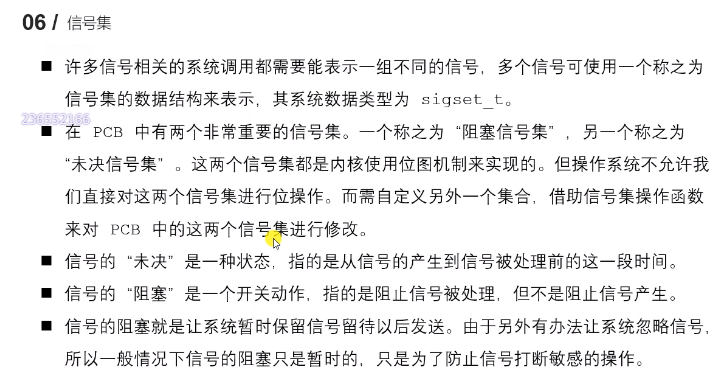
二十、kill、raise、abort、alarm、setitimer

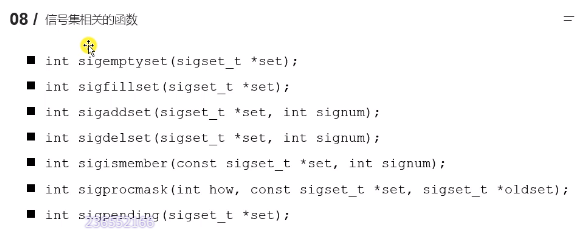


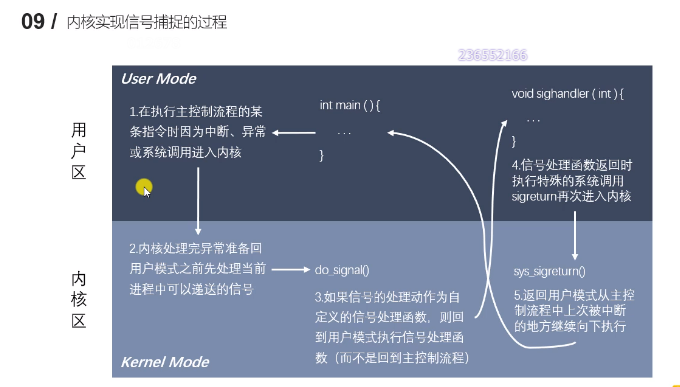
二十三、signal信号捕捉函数



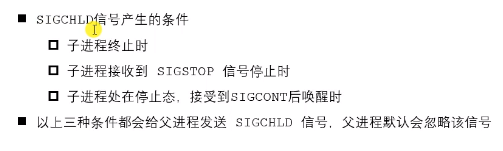
二十四、信号集







二十七、SIGCHLD信号



可用于解决僵尸进程的问题。

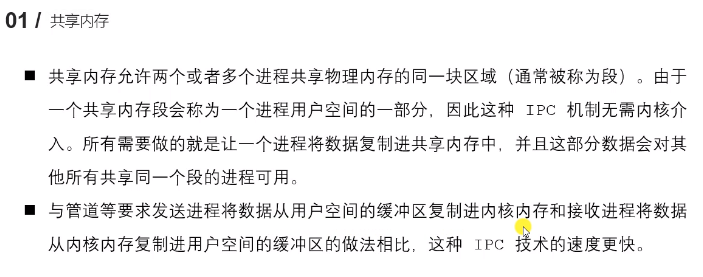
二十八、共享内存：需要进程同步

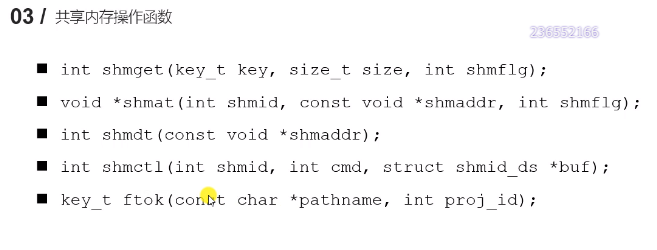
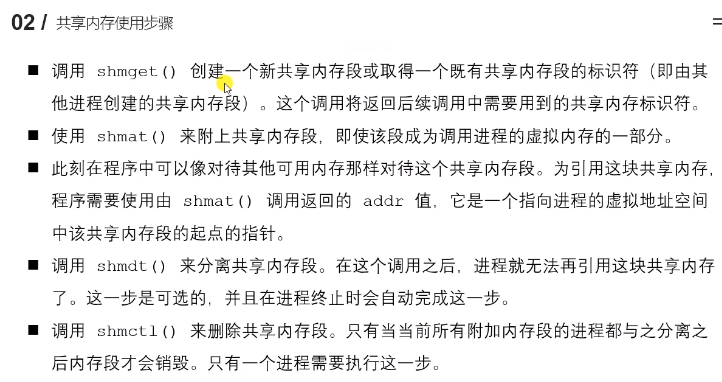
共享内存是一种通过将一块内存区域映射到多个进程的地址空间，使它们可以直接访问共享数据的机制。直接共享一块内存区域。

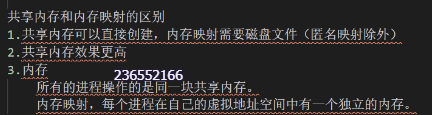
共享内存通过使用 shmget()、shmat() 等系统调用来创建共享内存区域，并将其附加到进程的地址空间。

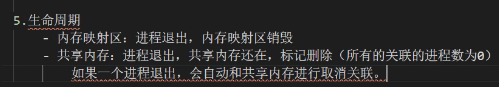
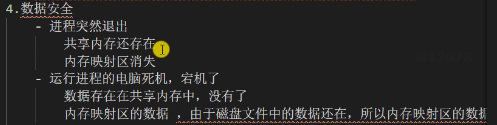
多个进程可以同时访问同一块共享内存，它们可以直接读写共享内存中的数据，而无需进行复制或数据传输。

共享内存适用于需要高性能且频繁交换数据的进程间通信场景。共享内存在断电或重新启动后会丢失数据。

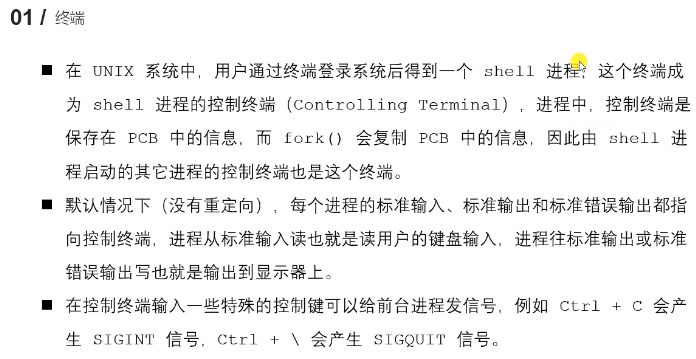




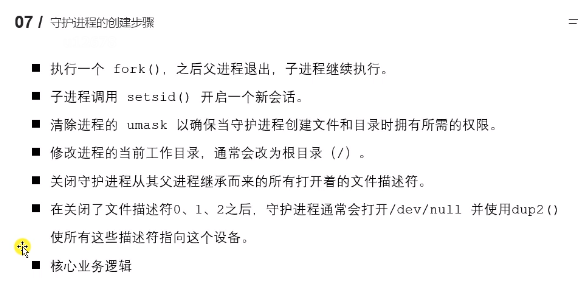
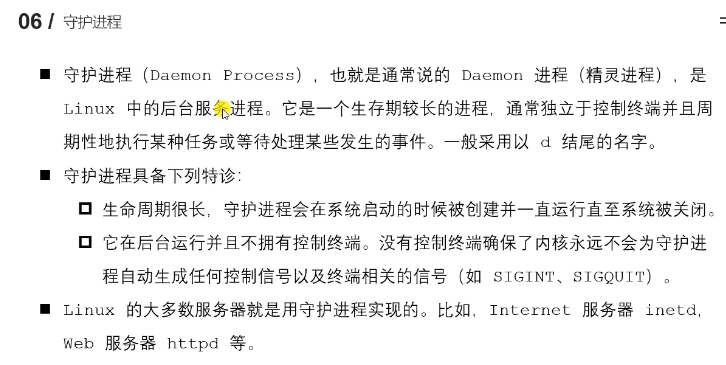
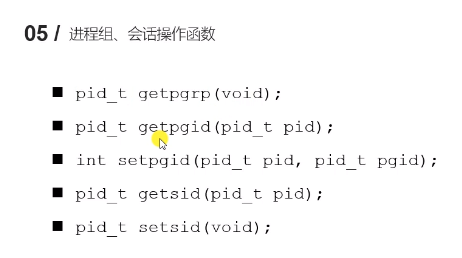
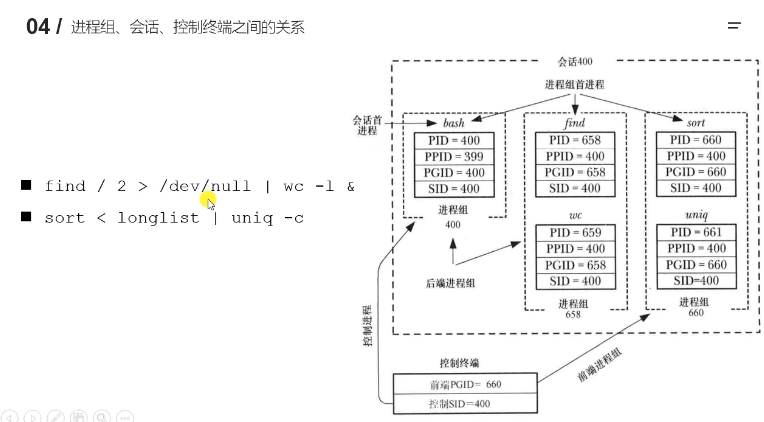
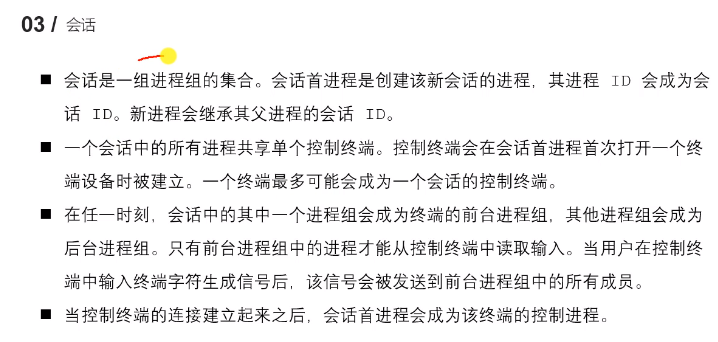
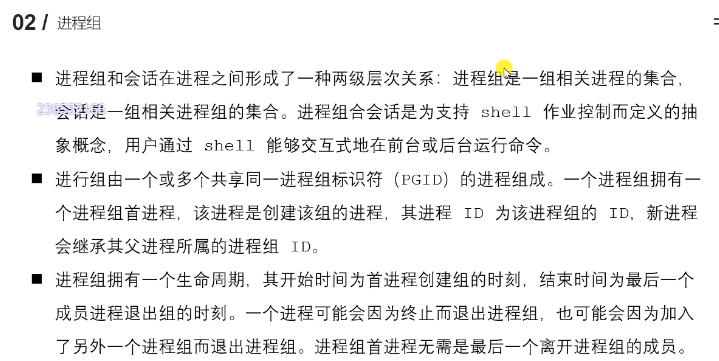




三十、守护进程



echo $$：查看当前进程的id。 tty ：查看当前进程的号



1. 父进程退出是因为不会在运行结束后出现shell提示符；子进程会从父进程处继承进程组的ID。
2. 子进程调用setsid()不会有控制终端；没有冲突。