Technology Tarena 达内科技

Unix系统高级编程

PART 2

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	进程
	10:30 ~ 11:20	
	11:30 ~ 12:20	进程的各种ID
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	创建子进程
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

进程

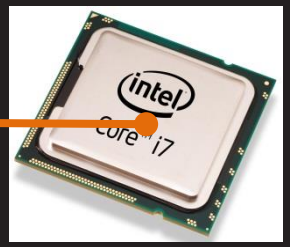
Technology Tarena 达内科技

进程的基本概念

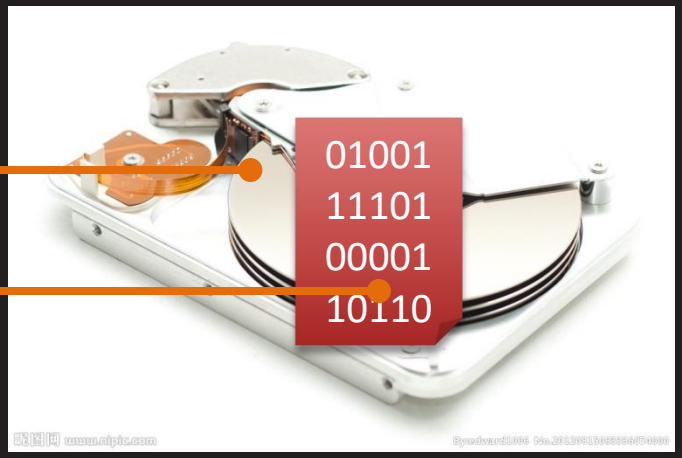
进程与程序

- 程序是被存储在磁盘上，包含机器指令和数据的文件
- 进程是被装载到内存中，被处理器操作的代码和数据
- 一个程序可被同时运行为多个进程
- 进程在操作系统中执行特定的任务

知识讲解

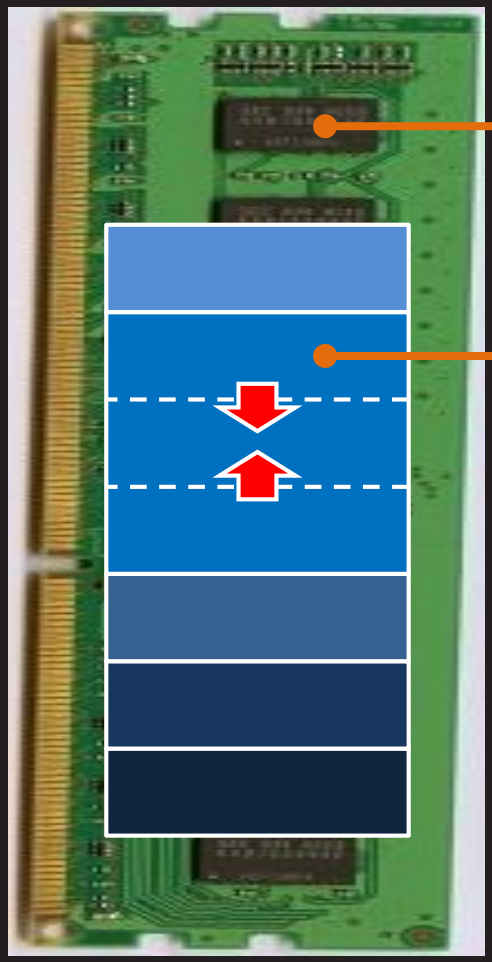


处理器



磁盘

程序



内存

进程

进程的分类

- Unix/Linux系统中的进程一般被分为以下三类
  - 交互式进程
    - 由Shell启动，既可在前台运行，也可在后台运行，通过终端接收用户的输入，并为用户提供输出
    - 如：vi、ps等
  - 批处理进程
    - 与终端没有联系，以进程序列的方式，在无需人工干预的条件下完成一组批量任务
    - 如：各种Shell脚本程序
  - 守护进程
    - 又名精灵进程，是系统的后台服务进程
    - 独立于控制终端，周期性地执行某种任务或等待某些事件
    - 在系统引导时启动，在系统关闭时终止，生命周期很长
    - 如：crond、lpd等

知识讲解

Technology Tarena 达内科技

进程快照

进程快照

- 使用ps命令可以查看当前在系统中运行的进程快照
  - 简单形式

```
$ ps
```

以简略方式显示当前用户拥有控制终端的进程信息

```
  PID TTY          TIME CMD
 2668 pts/0    00:00:00 su
 2697 pts/0    00:00:02 bash
16070 pts/0    00:00:00 ps
```

其中各列含义如下：

- PID - 进程标识
- TTY - 控制终端次设备号
- TIME - 进程运行时间
- CMD - 进程启动命令

进程快照（续1）

- 使用ps命令可以查看当前在系统中运行的进程快照
  - BSD风格常用选项
    - a - 显示所有用户拥有控制终端的进程信息
    - x - 也包括没有控制终端的进程
    - u - 以详尽方式显示
    - w - 以更大列宽显示

知识讲解

```
$ ps axu
```

```
USER       PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
avahi      868  0.0  0.0   3464   348 ?      S    01:51   0:00 avahi-daemon: c
colord     917  0.0  1.0  53572 10428 ?      Sl   01:51   0:00 /usr/lib/i386-l
root       953  0.0  0.0   4636   860 tty2   Ss+  01:51   0:00 /sbin/getty -8
root       967  0.0  0.0   4636   856 tty6   Ss+  01:51   0:00 /sbin/getty -8
root      1001  0.0  0.0   2180   712 ?      Ss   01:51   0:00 acpid -c /etc/a
root      1002  0.0  0.0   2624   932 ?      Ss   01:51   0:00 cron
daemon    1003  0.0  0.0   2476   348 ?      Ss   01:51   0:00 atd
```

进程快照（续2）

Technology Tarena 达内科技

- 使用ps命令可以查看当前在系统中运行的进程快照
  - SVR4风格常用选项
    - `-e` - 显示所有用户的进程信息
    - `-f` - 按完整格式显示
    - `-F` - 按更完整格式显示
    - `-l` - 按长格式显示

知识讲解

```
$ ps -efl
```

```
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
1 S avahi      868   857  0  80   0 -   866 unix_s 01:51 ?        00:00:00 avahi
4 S colord     917     1  0  80   0 - 13393 poll_s 01:51 ?        00:00:00 /usr/
4 S root       953     1  0  80   0 -  1159 n_tty_ 01:51 tty2     00:00:00 /sbin
4 S root       967     1  0  80   0 -  1159 n_tty_ 01:51 tty6     00:00:00 /sbin
1 S root      1001     1  0  80   0 -   545 poll_s 01:51 ?        00:00:00 acpid
1 S root      1002     1  0  80   0 -   656 hrtime 01:51 ?        00:00:00 cron
1 S daemon    1003     1  0  80   0 -   619 hrtime 01:51 ?        00:00:00 atd
```

进程快照（续3）

- 进程信息列表
  - **USER/UID** ：进程的用户ID
  - **PID** ：进程ID
  - **%CPU/C** ：CPU使用率
  - **%MEM** ：内存使用率
  - **VSZ** ：占用虚拟内存大小(KB)
  - **RSS** ：占用物理内存大小(KB)
  - **TTY** ：终端次设备号
    - **ttyn** - 实终端/物理终端
    - **pts/n** - 伪终端/虚拟终端
    - **?** - 无控制终端，如后台进程

知识讲解

进程快照（续4）

- 进程信息列表
  - **STAT/S** ：进程状态
    - **O** - 就绪，等待被调度
    - **R** - 运行，Linux下没有O状态，就绪状态也用R表示
    - **S** - 可唤醒睡眠，系统中断，获得资源，收到信号，都可被唤醒，转入运行状态
    - **D** - 不可唤醒睡眠，只能被wake_up系统调用唤醒
    - **T** - 暂停，收到SIGSTOP信号转入暂停状态，收到SIGCONT信号转入运行状态
    - **W** - 等待内存分页(2.6内核以后被废弃)

知识讲解

进程快照（续5）

- 进程信息列表
  - STAT/S ：进程状态
    - X - 死亡，不可见
    - Z - 僵尸，已停止运行，但其父进程尚未获取其状态
    - < - 高优先级
    - N - 低优先级
    - L - 有被锁到内存中的分页，实时进程和定制IO
    - s - 会话首进程
    - l - 多线程化的进程
    - + - 在前台进程组中

知识讲解

进程快照（续6）

- 进程信息列表
  - **START/STIME** ：进程开始时间
  - **TIME** ：进程运行时间
  - **COMMAND/CMD** ：进程启动命令
  - **F** ：进程标志，由下列值位或
    - **1** - 通过fork产生但是没有exec
    - **4** - 拥有超级用户特权
  - **PPID** ：父进程ID
  - **NI** ：进程nice值，-20到19
  - **PRI** ：进程动态优先级
    - 静态优先级=80+nice，60到99，值越小优先级越高，内核在静态优先级的基础上，根据进程的交互性计算出它的动态优先级，体现对IO消耗型进程的奖励和对处理机消耗型进程的惩罚

知识讲解

进程快照（续7）

- 进程信息列表
  - **ADDR** ：内核进程的内存地址，普通进程显示“-”
  - **SZ** ：占用虚拟内存页数
  - **WCHAN** ：进程正在等待的内核函数或事件
  - **PSR** ：进程当前被指派给哪个处理器运行

知识讲解

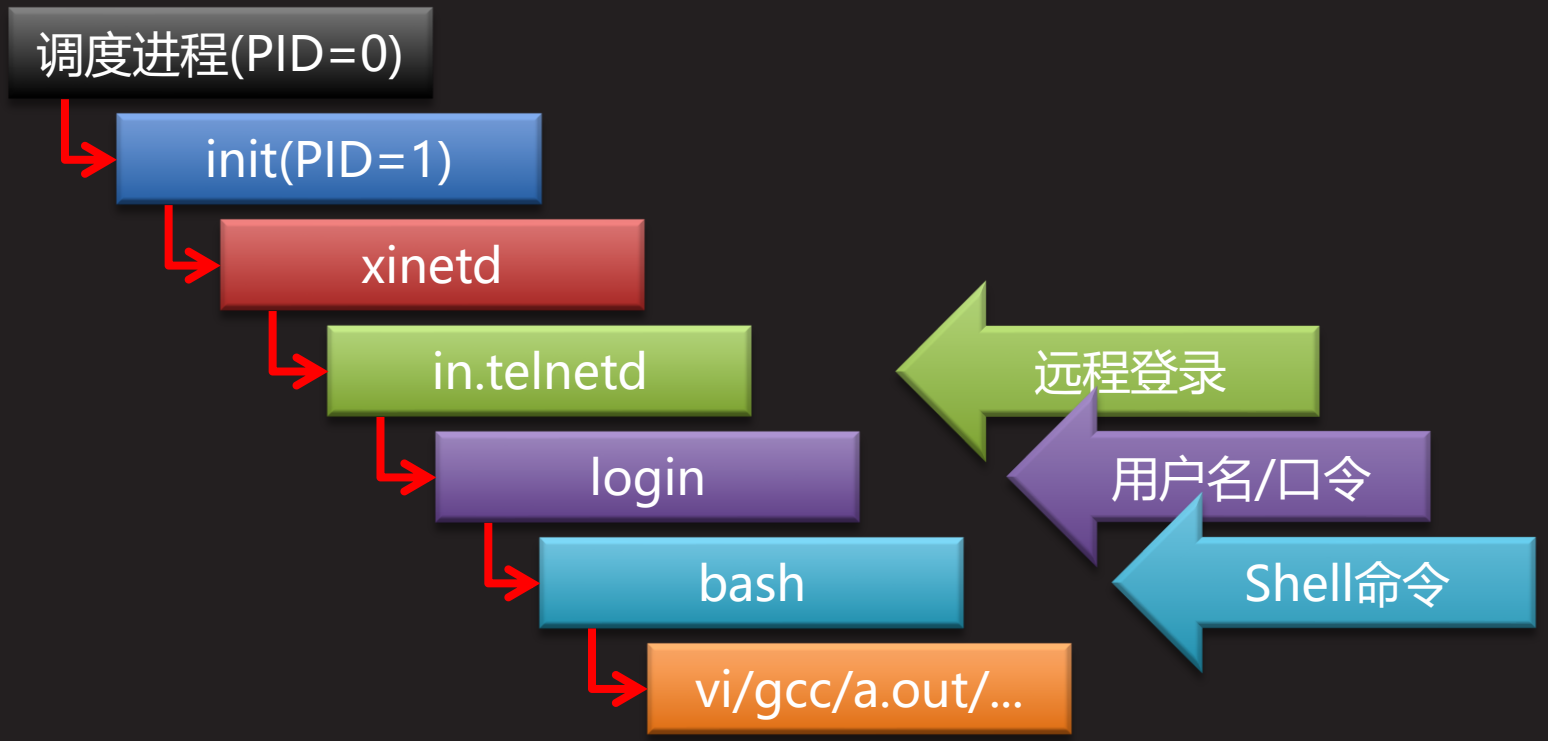
Technology Tarena 达内科技

父子孤尸

父子进程

- Unix/Linux系统的中进程存在父子关系。一个父进程可以创建多个子进程，但每个子进程最多只能有一个父进程。整个系统中只有一个根进程，即PID为0的调度进程。系统中的所有进程构成了一棵以调度进程为根的进程树

知识讲解



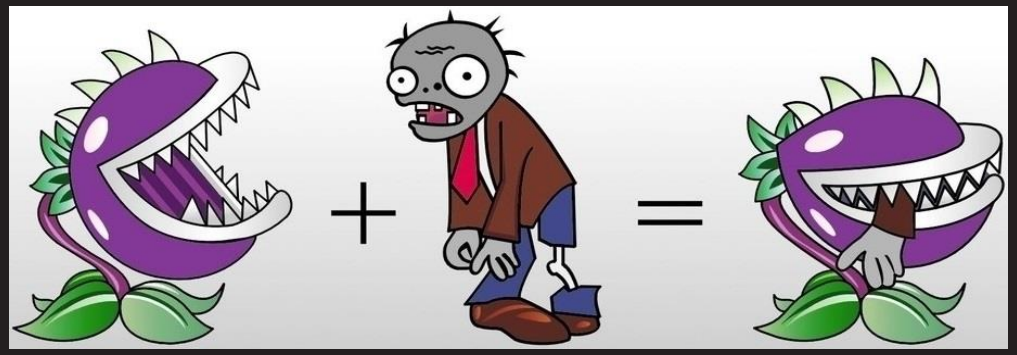
孤儿进程

- 父进程创建子进程以后，子进程在操作系统的调度下与其父进程同时运行
- 如果父进程先于子进程终止，子进程即成为孤儿进程，同时被init进程收养，即成为init进程的子进程，因此init进程又被称为孤儿院进程
- 一个进程成为孤儿进程是正常的，系统中的很多守护进程都是孤儿进程

知识讲解

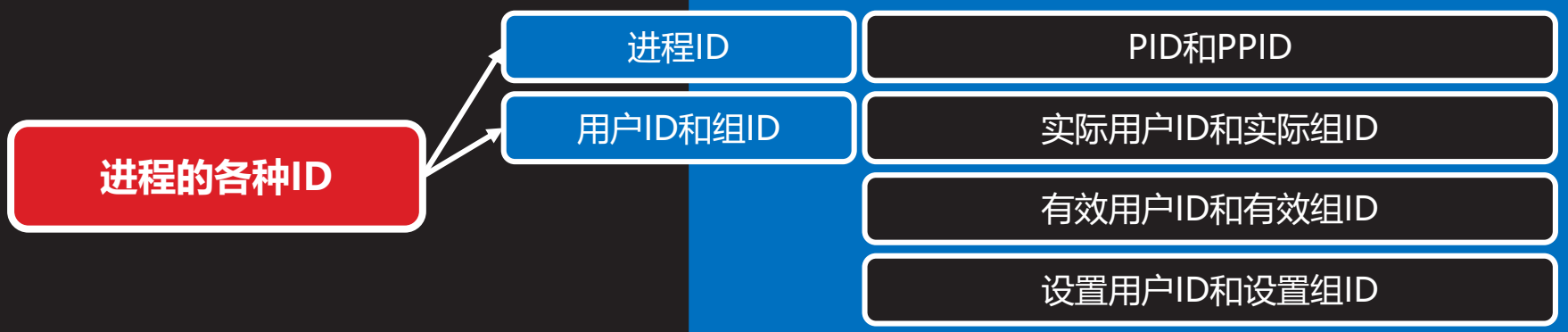
僵尸进程

- 如果子进程先于父进程终止，但父进程由于某种原因，没有回收子进程的退出状态，子进程即成为僵尸进程
- 僵尸进程虽然已经不再活动，但其终止状态仍然保留，也会占用系统资源，直到被其父进程回收才得以释放
- 如果父进程直到终止都未回收它的已成僵尸的子进程，init进程会立即收养并回收这些处于僵尸状态的子进程，因此一个进程不可能既是孤儿进程同时又是僵尸进程
- 一个进程成为僵尸进程需要引起注意，如果它的父进程长期运行而不终止，僵尸进程所占用的资源将长期得不到释放



知识讲解

进程的各种ID


进程的各种ID
进程ID
用户ID和组ID
PID和PPID
实际用户ID和实际组ID
有效用户ID和有效组ID
设置用户ID和设置组ID

Technology Tarena 达内科技

进程ID

PID和PPID

- 每个进程都有一个非负整数形式的唯一编号，即PID (Process Identification，进程标识)
- PID在任何时刻都是唯一的，但是可以重用，当进程终止并被回收以后，其PID就可以为其它进程所用
- 进程的PID由系统内核根据延迟重用算法生成，以确保新进程的PID不同于最近终止进程的PID

知识讲解

PID和PPID（续1）

- 系统中有些PID是专用的，比如
  - 0号进程，调度进程
    - 亦称交换进程(swapper)，系统内核的一部分，所有进程的根进程，磁盘上没有它的可执行程序文件
  - 1号进程，init进程
    - 在系统自举过程结束时由调度进程创建
    - 读写与系统有关的初始化文件，引导系统至一个特定状态，收养系统中的孤儿进程
    - 以超级用户特权运行的普通进程，永不终止
  - 2号进程，页守护进程
    - 负责虚拟内存系统的分页操作
- 除调度进程以外，系统中的每个进程都有唯一的父进程，对任何一个子进程而言，其父进程的PID即是它的PPID

PID和PPID（续2）

- 获取调用进程的PID

```
#include <unistd.h>

pid_t getpid (void);
```

返回调用进程的PID

- 获取调用进程的PPID

```
#include <unistd.h>

pid_t getppid (void);
```

返回调用进程的父进程的PID

Technology Tarena 达内科技

用户ID和组ID

实际用户ID和实际组ID

- 当一个用户通过合法的用户名和口令登录系统以后，系统就会为他启动一个Shell进程，Shell进程的实际用户ID和实际组ID就是该登录用户的用户ID和组ID。该用户在Shell下启动的任何进程都是Shell进程的子进程，自然也就继承了Shell进程的实际用户ID和实际组ID
- 获取调用进程的实际用户ID和实际组ID

知识讲解

```
#include <unistd.h>

uid_t getuid (void);
uid_t getgid (void);
```

分别返回调用进程的实际用户ID和实际组ID

有效用户ID和有效组ID

- 一个进程的用户和组身份决定了它可以访问哪些资源，比如读、写或者执行某个文件。但真正被用于权限验证的并不是进程的实际用户ID和实际组ID，而是其有效用户ID和有效组ID。一般情况下，进程的有效用户ID和有效组ID就取自其实际用户ID和实际组ID，二者是等价的
- 获取调用进程的有效用户ID和有效组ID

知识讲解

```
#include <unistd.h>

uid_t geteuid (void);
uid_t getegid (void);
```

分别返回调用进程的有效用户ID和有效组ID

设置用户ID和设置组ID

- 如果用于启动进程的可执行文件带有设置用户ID位和(或)设置组ID位，那么该进程的有效用户ID和(或)有效组ID就不再取自其实际用户ID和(或)实际组ID，而是取自可执行文件的拥有者用户ID和(或)组ID

知识讲解

mode_t st_mode; // unsigned int

1			S_ISUID	设置用户ID
	1		S_ISGID	设置组ID
		1	S_ISVTX	粘滞
4	2	1		

设置用户ID和设置组ID（续1）

- 例如
  - 假设ids文件的拥有者用户和组都是root，且其它用户对该文件有可执行权限

```
-rwxr-xr-x 1 root root ... ids
```

  - 隶属于tarena组(GID=1000)的tarena用户(UID=1000)登录系统，运行ids程序

```
    进程ID：...
  父进程ID：...
实际用户ID：1000
  实际组ID：1000
有效用户ID：1000
  有效组ID：1000
```

知识讲解

设置用户ID和设置组ID（续2）

- 例如
  - 可以看到进程的有效用户ID和实际用户ID一样都是1000，而有效组ID也和实际组ID一样都是1000
  - 现在root用户为ids文件添加设置用户ID和设置组ID权限位

```
# chmod u+s ids
# chmod g+s ids
```

```
-rwsr-sr-x 1 root root ... ids
```

知识讲解

设置用户ID和设置组ID（续3）

Technology Tarena 达内科技

- 例如
  - 隶属于tarena组的tarena用户再次运行ids程序

```
    进程ID：...
  父进程ID：...
实际用户ID：1000
  实际组ID：1000
有效用户ID：0
  有效组ID：0
```

  - 不难发现，进程的实际用户ID和实际组ID并没有发生变化，仍然是1000，但它的有效用户ID和有效组ID却变成了0，显然这是ids文件的拥有者root用户的用户ID的组ID，而参与权限判断，决定该进程能做什么不能做什么的恰恰是它的有效用户ID和有效组ID，tarena用户扮演root用户行权

知识讲解

进程的各种ID

【参见：TTS COOKBOOK 】

课堂练习

- 进程的各种ID

创建子进程

fork

fork

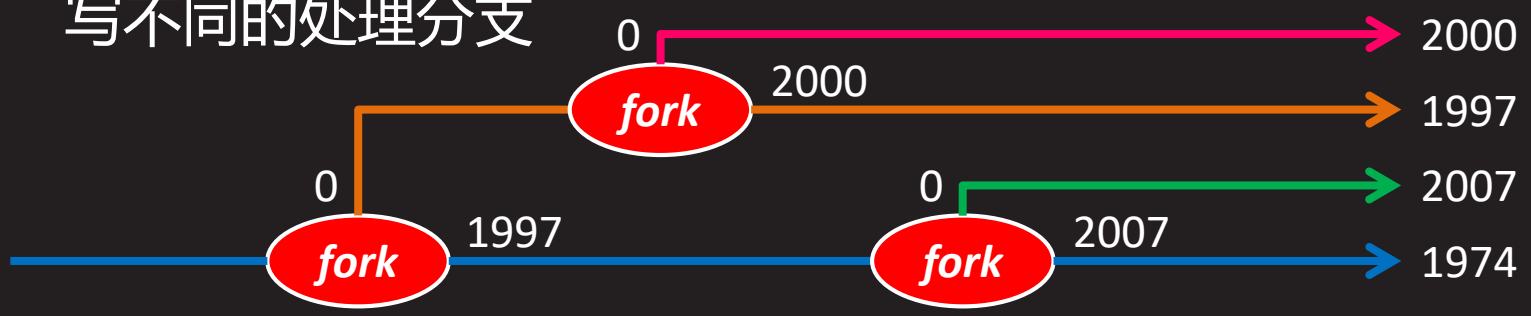
- 创建子进程

```
#include <unistd.h>

pid_t fork (void);
```

成功分别在父子进程中返回子进程的PID和0，失败返回-1

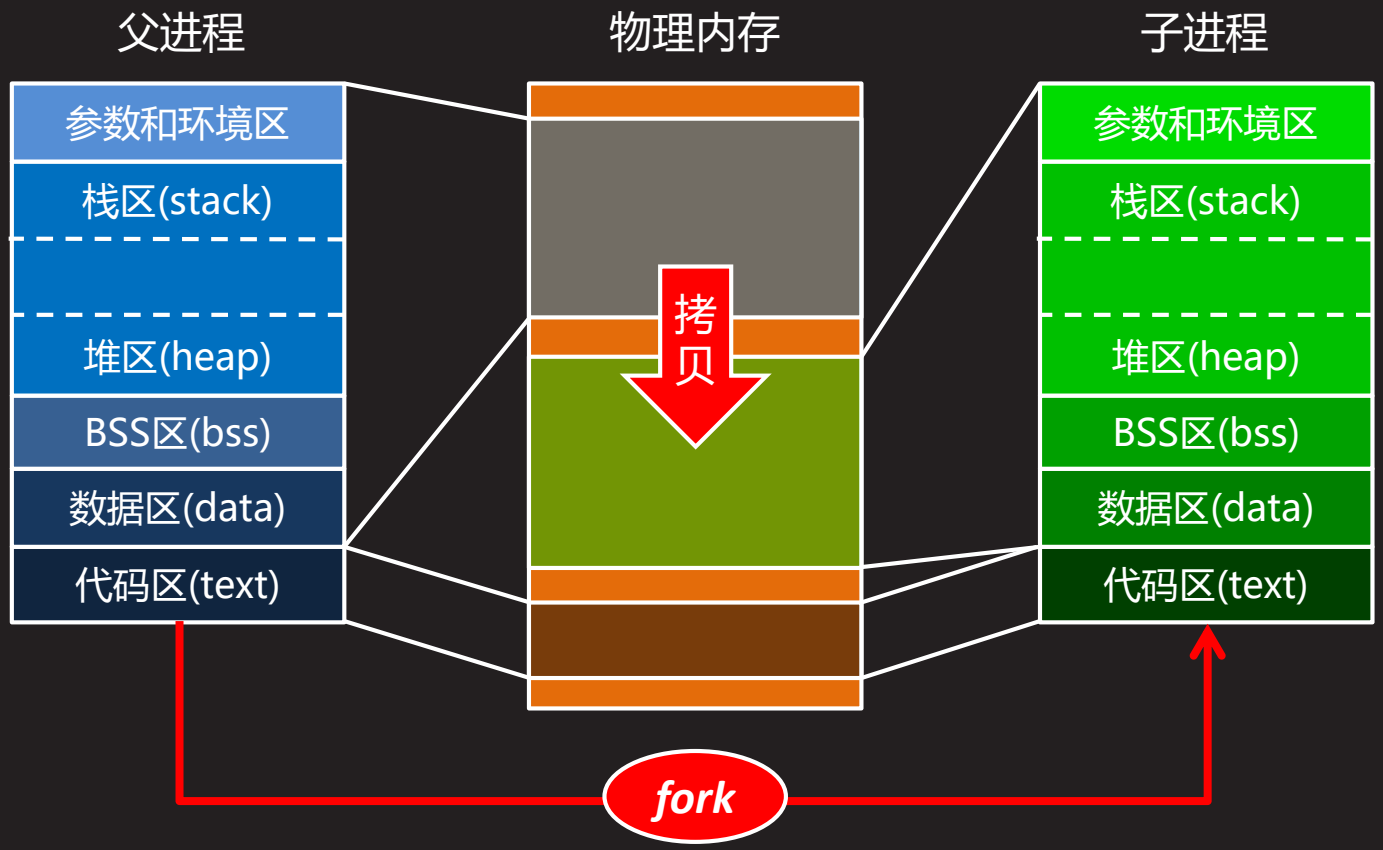
- 调用一次返回两次
  - 在父进程中返回所创建子进程的PID，而在子进程中返回0
  - 函数的调用者可以根据返回值的不同，分别为父子进程编写不同的处理分支



知识讲解

fork（续1）

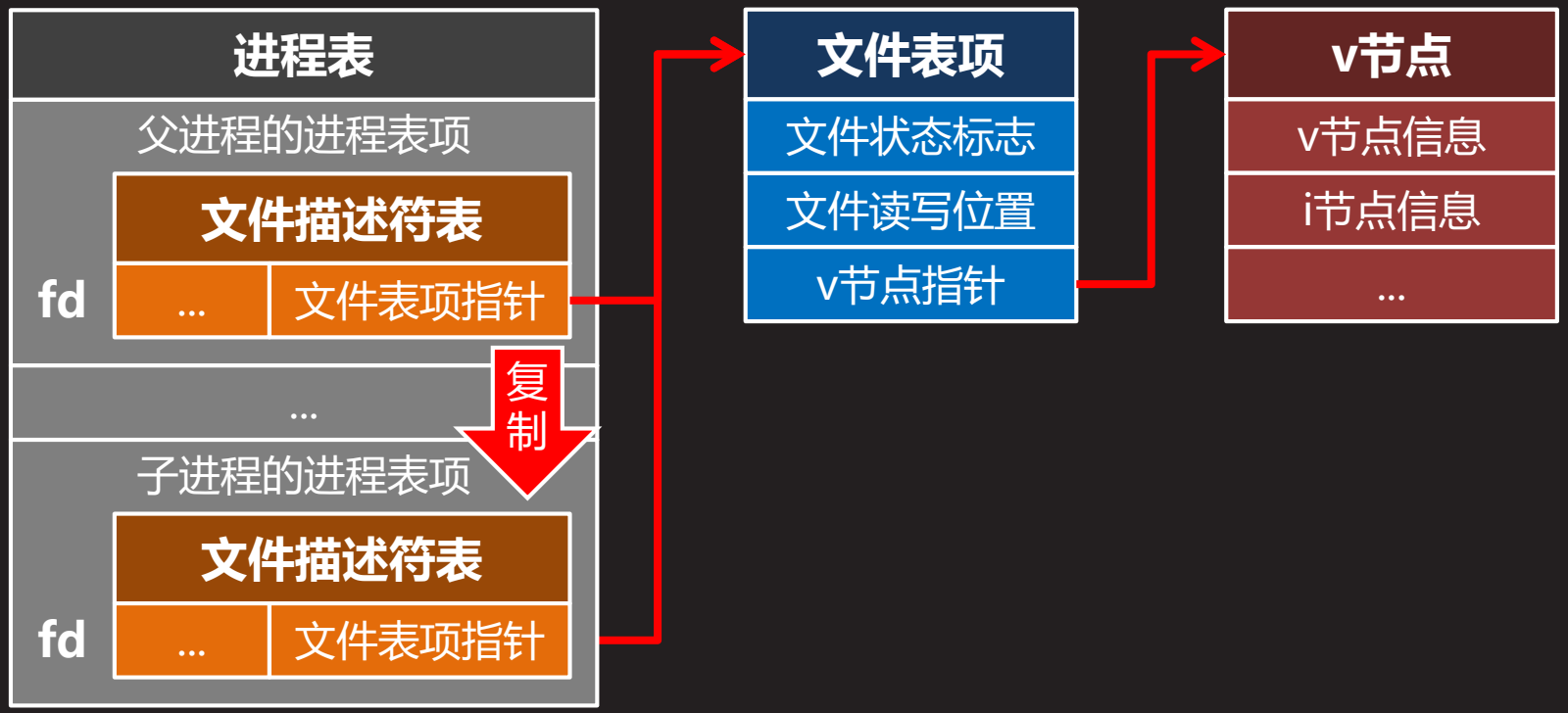
- 子进程是父进程的不完全副本，子进程的数据区、BSS区、堆栈区(包括I/O流缓冲区)，甚至参数和环境区都从父进程拷贝，唯有代码区与父进程共享



知识讲解

fork（续2）

- fork函数成功返回以后，父子进程各自独立运行，其被调度的先后顺序并不确定，某些实现可以保证子进程先被调度
- fork函数成功返回以后，系统内核为父进程维护的文件描述符表也被复制到子进程的进程表项中，文件表项并不复制



知识讲解

fork（续3）

- 系统总线程数达到上限(/proc/sys/kernel/threads-max)，或用户总进程数达到上限(ulimit -u)，fork函数将返回失败
- 一个进程如果希望创建自己的副本并执行同一份代码，或希望与另一个进程并发地运行，都可以使用fork函数
- 调用fork函数前的代码只有父进程执行，fork函数成功返回后的代码父子进程都会执行，受逻辑控制进入不同分支
  - pid_t pid = fork ();
    if (pid == -1) {
    perror ("fork"); exit (EXIT_FAILURE); }
    if (pid == 0) { 子进程执行的代码; }
    else { 父进程执行的代码; }
    父子进程都执行的代码;

知识讲解

创建子进程

【参见：TTS COOKBOOK 】

课堂练习

- 创建子进程

子进程是父进程的副本

【参见：TTS COOKBOOK 】

课堂练习

- 子进程是父进程的副本

Technology Tarena 达内科技

父子进程共享文件表

【参见：TTS COOKBOOK 】

课堂练习

- 父子进程共享文件表

Technology Tarena 达内科技

孤儿与僵尸

【参见：TTS COOKBOOK 】

课堂练习

- 孤儿与僵尸

vfork

vfork

- 创建轻量级子进程

```
#include <unistd.h>

pid_t vfork (void);

        成功分别在父子进程中返回子进程的PID和0，失败返回-1
```

- vfork与fork函数的功能基本相同，只有以下两点区别
  - vfork函数创建的子进程不复制父进程的物理内存，也不拥有自己独立的内存映射，而是与父进程共享全部地址空间
  - vfork函数会在创建子进程的同时挂起其父进程，直到子进程终止，或通过exec函数启动了另一个可执行程序

知识讲解

vfork（续1）

- 终止vfork函数创建的子进程，不要使用return语句，也不要调用exit函数，而要调用_exit函数，以避免对其父进程造成不利影响
- vfork函数的典型用法就是在所创建的子进程里直接调用exec函数启动另外一个进程取代其自身，这比调用fork函数完成同样的工作要快得多
  - pid_t pid = vfork ();
    if (pid == -1) {
        perror ("vfork"); exit (EXIT_FAILURE); }
    if (pid == 0)
        if (execl ("ls", "ls", "-l", NULL) == -1) {
            perror ("execl"); _exit (EXIT_FAILURE); }

知识讲解

vfork（续2）

- 写时复制(copy-on-write)
  - 传统意义上的fork系统调用，必须把父进程地址空间中的内容一页一页地复制到子进程的地址空间中(代码区除外)。这无疑是个十分漫长的过程(在系统内核看来)
  - 而多数情况下的子进程其实只是想读一读父进程的数据，并不想改变什么。更有甚者，可能连读一读都觉得多余，比如直接通过exec函数启动另一个进程的情况。漫长的内存复制在这里显得笨拙且毫无意义
  - 写时复制以惰性优化的方式避免了内存复制所带来的系统开销。在子进程创建伊始，并不复制父进程的物理内存，只复制它的内存映射表即可，父子进程共享同一个地址空间，直到子进程需要写这些数据时，再复制内存亦不为迟

vfork（续3）

- 写时复制(copy-on-write)
  - 写时复制带来的好处是，子进程什么时候写就什么时候复制，写几页就复制几页，没有写的就不复制。惰性优化算法的核心思想就是尽一切可能将代价高昂的操作，推迟到非做不可的时候再做，而且最好局限在尽可能小的范围里
  - 现代版本的fork函数已经广泛采用了写时复制技术，从这个意义上讲，vfork函数的存在纯粹只是一个历史遗留的产物，尽管它的速度还是比fork要快一点(连内存映射表都不复制)，但它的地位已远不如写时复制技术被应用到fork函数的实现中以前那么重要了

创建轻量级子进程

【参见：TTS COOKBOOK 】

课堂练习

- 创建轻量级子进程

Technology Tarena 达内科技

总结和答疑