进程概念

进程和程序

- 程序：(program)是存放在磁盘文件中的可执行文件；
- 进程：(process)是正在执行的(动态的)程序，占用系统资源，在内存中执行。**进程是分配系统资源的基本单位。**
- 区别：
  - 程序是静态的，进程是动态的；
  - 程序一般保存在磁盘中，不占用系统资源，进程会占用系统资源
  - 一个程序可以对应多个进程，一个进程可以执行一个或多个程序
  - 进程具有并发性，而程序没有
  - 程序没有生命周期，进程有生命周期

cpu密集型程序：程序大量的时间都在用CPU进行计算；

IO密集型程序：程序大量的时间都在和磁盘打交道；

并行：在同一时间，多个进程，每一个进程都拥有一个cpu进行计算；

并发：多个进程，只有少量的cpu，每个进程只能独占cpu一小会，就会让出cpu供其他进程运算；

进程运行特性

- 多任务：内存中可以存在多个进程
- 并发：多个进程可以并发执行
- 独立：进程之间互不影响

进程状态转换

- 运行状态

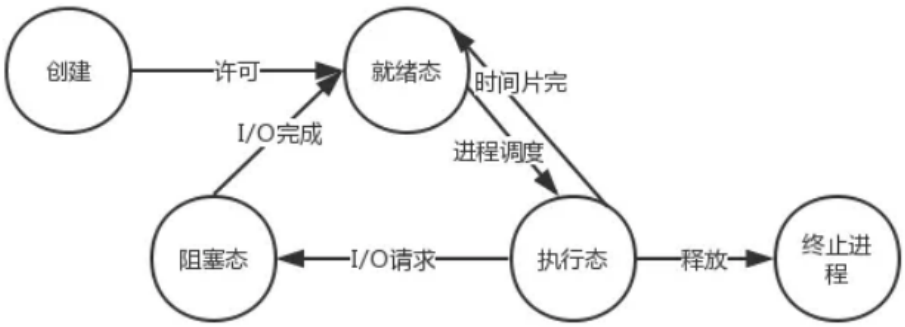
  表示当前进程占用着CPU等资源

- 就绪状态

  除CPU之外一切运行资源都已经准备就绪，等待CPU分配资源，只要分配资源后，即可立即运行

- 阻塞状态

  指进程在运行过程中，由于需要请求外部资源，造成当前进程无法继续执行，从而主动放弃当前占用的CPU资源，转而等待所请求的资源



进程控制块PCB

每个进程在内核中都有一个**进程控制块(PCB，process control block)**来维护进程的相关信息，Linux内核的进程控制块是**task_struct结构体。**

task_struct中主要内容：

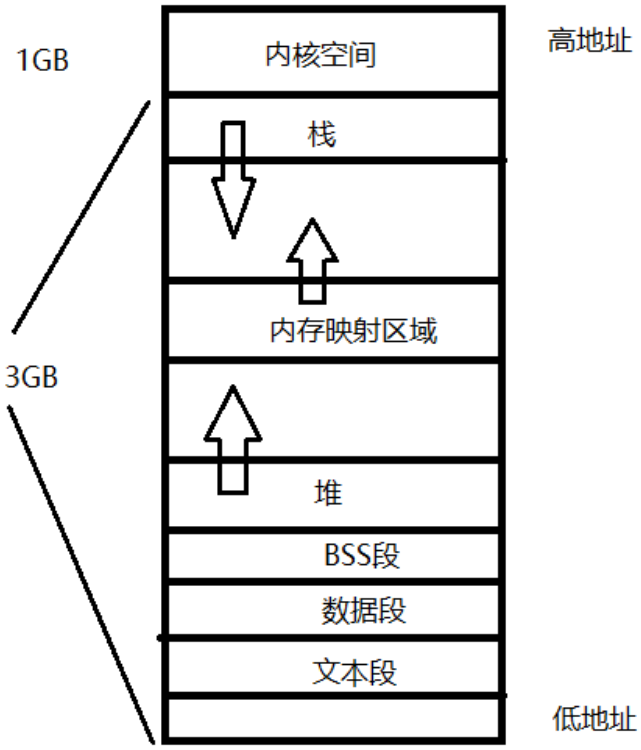
- 标识符：进程id，系统中每一个程序都有唯一的一个id(非负整数)
- 状态：运行、就绪、阻塞，表示进程的运行状态
- 优先级：相对于其他进程的优先级
- 程序计数器：程序中即将被执行的下一条指令
- 内存指针：程序代码和进程想关数据的指针，与其他进程共享的内存块的指针

进程地址空间

操作系统创建一个虚拟地址空间，作为进程可以引用的地址集合，32位系统为例是2^32=4GB，这个地址空间被分割成一个个的页面(page)，常见大小4KB。

这些页面被映射到物理内存，但并不是所有的页面都在内存中才可以运行程序，而是只加载需要的部分：

- 当进程引用到的内容在物理内存中，就由硬件映射到进程地址空间中
- 当进程引用到的内存不在物理内存中，就由操作系统负责将缺失的部分装入物理内存并重新执行失败的指令，这个过程也叫做「缺页中断」



一个进程地址空间主要包括：

- 代码段：只读
- 数据段：初始化过的变量
- BSS段：未初始化的变量
- 堆空间：低地址向高地址增长，调用malloc分配堆内存
- 栈空间：高地址向低地址增长，自动增长和释放，用于存储临时变量和函数调用
- 内存映射空间：比如用于进程间通信的共享内存文件

进程常用命令

kill

- 给进程发送一个信号
- SIGKILL 9号信号
- kill -9 pid ——杀死进程

ps

进程状态查看：**ps -aux / ps -ajxf**

```
[run@localhost progress]$ ps -ajxf
 PPID    PID   PGID    SID TTY       TPGID STAT   UID   TIME COMMAND
    0      2      0      0 ?           -1 S        0   0:00 [kthreadd]
    2      4      0      0 ?           -1 S<       0   0:00  \_ [kworker/0:0H]
    2      5      0      0 ?           -1 S        0   0:00  \_ [kworker/u256:0]
    2      6      0      0 ?           -1 S        0   0:00  \_ [ksoftirqd/0]
    2      7      0      0 ?           -1 S        0   0:00  \_ [migration/0]
    2      8      0      0 ?           -1 S        0   0:00  \_ [rcu_bh]
    2      9      0      0 ?           -1 R        0   0:00  \_ [rcu_sched]
    2     10      0      0 ?           -1 S<       0   0:00  \_ [lru-add-drain]
    2     11      0      0 ?           -1 S        0   0:00  \_ [watchdog/0]
```

- PPID ——父进程id
- PID ——当前进程id

- PGID ——组进程id
- SID ——会话id
- TTY ——登录者终端机的位置（与中断无关用？显示）
  - 一般liunx允许有7个终端，可以用ctrl0+alt+fn来切换，n是要切换的终端号
- TPGID ——进程连接到的TTY(终端)所在的前台进程组的id
- STAT ——进程状态字段
  - D(Disk sleep)磁盘休眠状态(不可中断休眠)：进程通常会等待IO的结束
  - R(running)运行状态：正在运行中或者在运行队列里的进程
  - S(sleeping)睡眠状态(可中断睡眠)：进程在等待事件完成
  - T(stopped)停止状态：可以通过发送SIGSTOP信号给进程来停止进程，这个暂停的进程可以通过发送 SIGCONT 信号让进程继续运行。
  - X死亡状态（dead）：这个状态只是一个返回状态，你不会在列表里看到这个状态
  - Z(zombie)僵尸进程
  - < 高优先级
  - N 低优先级
  - L 有页面在内存中被锁存
  - s 进程领导者，表示其有子进程
  - l 多线程
  - +位于前台进程组
- UID ——用户id

孤儿进程

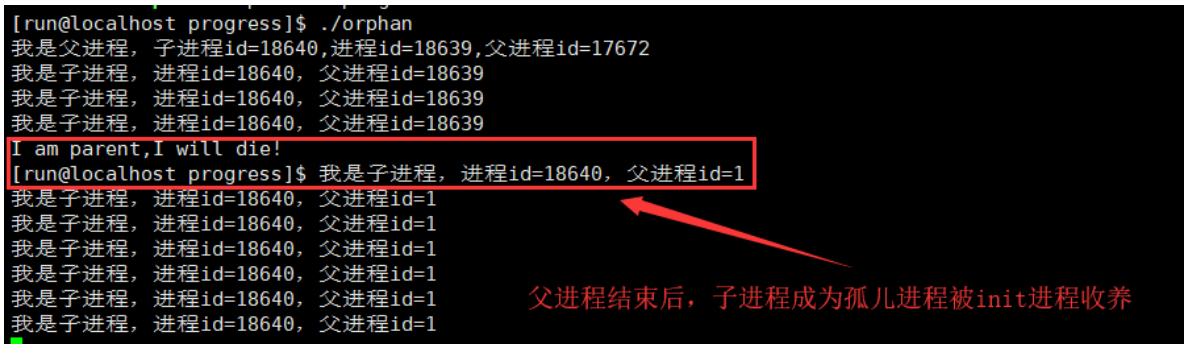
一个父进程退出或者被杀死，而它的一个或者多个子进程还在运行，那么这些子进程将成为**孤儿进程。**孤儿进程将被init进程(即pid为1的进程)所收养，并由init进程对它们完成状态收集工作。

举个栗子：

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

int main()
{
        //调用fork函数创建进程
        pid_t pid = fork();
        if(pid < 0)
        {
                perror("fork error");
                exit(1);
        }
        else if(pid == 0)
        {
                while(1)
                {
                        printf("我是子进程，进程id=%d，父进程id=%d\n",getpid(),getppid());
                        sleep(1);
                }
        }
        else
        {
```

```
                printf("我是父进程，子进程id=%d,进程id=%d,父进程id=%d\n",pid,getpid(),getppid());
                sleep(3);
                printf("I am parent,I will die!\n");
        }
        return 0;
}
```



子进程一直在运行，通过ctrl+z 和ctrl+c 无法杀死

僵尸进程

子进程已经终止了，但是它的父进程还没有回收其资源(PCB)，那么这个进程就称为**僵尸进程。**

举个栗子：

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

int main()
{
    pid_t pid = fork();
    if (pid < 0)
    {
        perror("fork err");
        exit(1);
     }
     else if (pid == 0)
     {
        printf("I am child process, pid = %d, ppid = %d\n", getpid(), getppid());
        sleep(2);
        printf("I am child, and i will die");
     }
     else
     {
        // 防止父进程先结束，产生孤儿进程。
        while (1)
        {
            printf("I am father process, pid = %d\n", getpid());
            sleep(1);
        }
  }
  // 杀死僵尸进程方法，把其父进程kill即可
  return 0;
```

```
}
```

环境变量

环境变量是指在操作系统中用来指定操作系统运行环境的一些参数。

常见环境变量

- PATH:可执行文件的搜索路径
- HOME:当前用户主目录的路径
- SHELL:当前Shell,它的值通常是/bin/bash

查看环境变量

命令查看

```
echo $name  //name是环境变量名称
```

getenv函数

```
#include <stdlib.h>
char *getenv(const char *name);
```

getenv()用来取得参数name环境变量的内容

- name：环境变量的名称
- 返回值：执行成功则返回指向该内容的指针，找不到符合的环境变量名称则返回NULL