1.冯诺依曼体系结构

2. 进程

2.1 基本概念

课本概念：程序的一个执行实例，正在执行的程序等

内核观点：担当分配系统资源（CPU时间，内存）的实体。

2.2描述进程-PCB

进程信息被放在一个叫做进程控制块的数据结构中，可以理解为进程属性的集合。

课本上称之为PCB（process control block），Linux操作系统下的PCB是: task\_struct.

2.3 task\_ struct内容分类

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

2.4进程状态(进程状态：就绪，运行，阻塞)

运行状态(R)：并不意味着程序就正在运行，他可能正在运行或在运行队列里面。

睡眠状态(S)：意味着进程正在等待事件完成。可中断睡眠态。

磁盘休眠状态(D)（Disk sleep）：有时候也叫不可中断睡眠状态（uninterruptible sleep），在这个状态的进程通常会等待IO的结束。

停止状态(T)（stopped）： 可以通过发送 SIGSTOP 信号给进程来停止（T）进程。这个被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，你不会在任务列表里看到这个状态

####僵死进程(Z): 如何产生：子进程先于父进程退出，将自己的

退出原因保存在pcb中，操作系统检测到子进程退出，因为父进程 有可能关注退出原因，所以不敢随意释放所有资源，通知父进程的退出，但是这时候父进程可能正在打麻将，没有关注到这个通知，导致子进程退出了。但是资源一直没有被释放，处于僵死状态，成为僵尸进程. 处理：干掉父进程 如何避免：进程等待

2.5进程优先级(越小优先级越高)

2.5.1基本概念

①cpu资源分配的先后顺序，就是指进程的优先权（priority）。

②优先权高的进程有优先执行权利。配置进程优先权对多任务环境的linux很有用，可以改善系统性能。

③，还可以把进程运行到指定cup，把不重要的进程安排到某个CPU，可以大大改善系统整体性能。

UID : 代表执行者的身份

PID : 代表这个进程的代号

PPID ：代表这个进程是由哪个进程发展衍生而来的，亦即父进程的代号

PRI ：代表这个进程可被执行的优先级，其值越小越早被执行

NI ：代表这个进程的nice值

Nice命令和renice命令

④用top命令更改已存在进程的ni

进入top后按“r”–>输入进程PID–>输入nice值

其他概念

竞争性: 系统进程数目众多，而CPU资源只有少量，甚至1个，所以进程之间是具有竞争属性的。为了高效完成任务，更合理竞争相关资源，便具有了优先级

独立性: 多进程运行，需要独享各种资源，多进程运行期间互不干扰

并行: 多个进程在多个CPU下分别，同时进行运行，这称之为并行

并发: 多个进程在一个CPU下采用进程切换的方式，在一段时间之内，让多个进程都得以推进，称之为并发。

3.环境变量

3.1常见环境变量：

PATH : 指定命令的搜索路径

HOME : 指定用户的主工作目录(即用户登陆到Linux系统中时,默认的目录)

SHELL : 当前Shell,它的值通常是/bin/bash。

3.2 和环境变量相关的命令

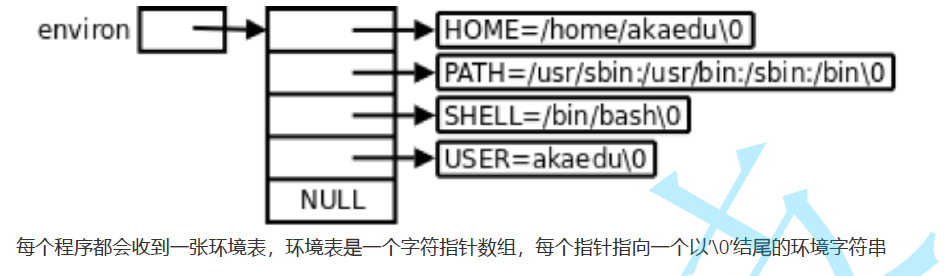
echo:查看环境变量的值

export：设置环境变量的值

env：查看整个系统环境变量

unset：清除环境变量的值

set：显示本地定义的shell和环境变量的值



3.3环境变量通常是具有全局属性的

环境变量通常具有全局属性，可以被子进程继承下去

4.程序地址空间

Task\_struct--->mm\_struct(程序地址空间)--->页表-- >物理内存

5. 进程创建

fork函数初识

在linux中fork函数时非常重要的函数，它从已存在进程中创建一个新进程。新进程为子进程，而原进程为父进程。

pid\_t fork(void); 返回值:子进程中返回0，父进程返回子进程id，出错返回-1

①进程调用fork，当控制转移到内核中的fork代码后，内核做：

分配新的内存块和内核数据结构给子进程

将父进程部分数据结构内容拷贝至子进程

添加子进程到系统进程列表当中

fork返回，开始调度器调度

5.2 进程退出

Return、\_exit、exit,, ctrl+c

①\_exit函数: #include <unistd.h> void \_exit(int status);

参数：status 定义了进程的终止状态，父进程通过wait来获取该值

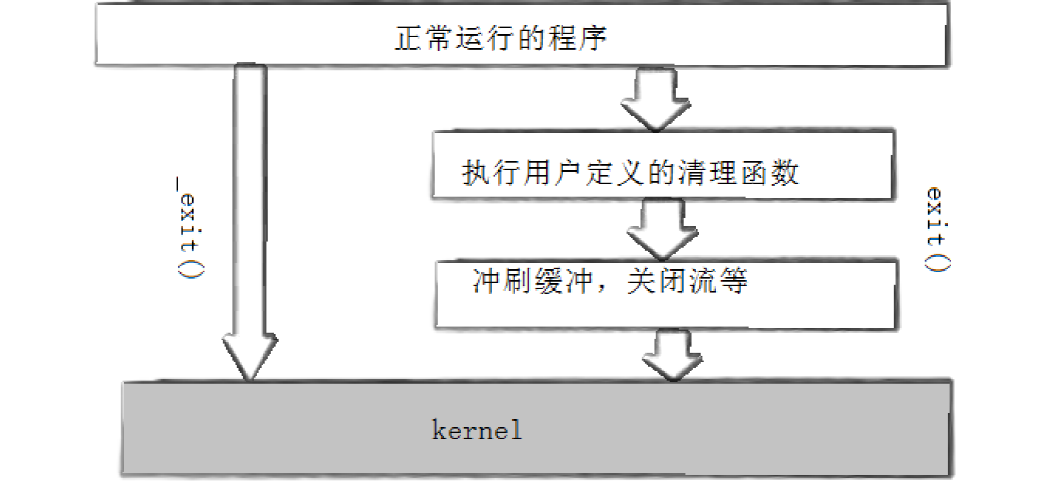
说明：虽然status是int，但是仅有低8位可以被父进程所用。所以\_exit(-1)时，在终端执行$?发现返回值是255。

②exit函数：#include <unistd.h> void exit(int status);

1. 执行用户通过 atexit或on\_exit定义的清理函数。

2. 关闭所有打开的流，所有的缓存数据均被写入

3. 调用\_exit



4.进程等待

4.1进程等待必要性

①子进程退出，父进程如果不管不顾，就可能造成‘僵尸进程’的问题，进而造成内存泄漏。

②另外，进程一旦变成僵尸状态，那就刀枪不入，“杀人不眨眼”的kill -9 也无能为力，因为谁也没有办法杀死一个已经死去的进程。

③最后，父进程派给子进程的任务完成的如何，我们需要知道。如，子进程运行完成，结果对还是不对，或者是否正常退出。

④父进程通过进程等待的方式，回收子进程资源，获取子进程退出信息。

4.2wait方法

#include<sys/types.h>

#include<sys/wait.h>

pid\_t wait(int\*status);

返回值：成功返回被等待进程pid，失败返回-1。

参数：输出型参数，获取子进程退出状态,不关心则可以设置成为NULL

4.3 waitpid方法

pid\_ t waitpid(pid\_t pid, int \*status, int options);

返回值：

当正常返回的时候waitpid返回收集到的子进程的进程ID；

如果设置了选项WNOHANG,而调用中waitpid发现没有已退出的子进程可收集,则返回0；

如果调用中出错,则返回-1,这时errno会被设置成相应的值以指示错误所在；

参数：

pid：

Pid=-1,等待任一个子进程。与wait等效。

Pid>0.等待其进程ID与pid相等的子进程。

status:

WIFEXITED(status): 若为正常终止子进程返回的状态，则为真。（查看进程是否是正常退出）

WEXITSTATUS(status): 若WIFEXITED非零，提取子进程退出码。（查看进程的退出码）

options:

WNOHANG: 若pid指定的子进程没有结束，则waitpid()函数返回0，不予以等待。若正常结束，则返回该子进程的ID。