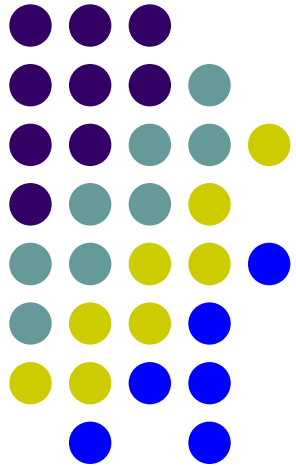
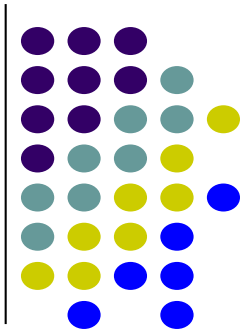
第9章

Linux进程

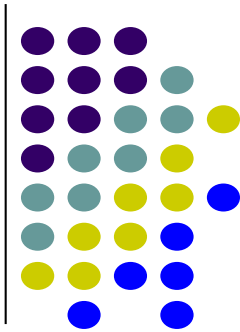


本章内容



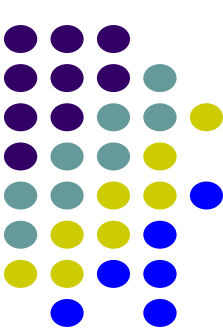
- 进程管理与系统监视
- 创建进程
- 进程调度
- 守护进程
- 进程通信

9.1 进程管理与系统监视



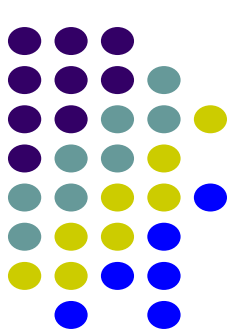
- 程序：是存储在磁盘上能完成一定功能的指令序列和数据的静态实体。
- 进程：是指具有独立功能的程序的一次运行过程，也是系统资源分配和调度的基本单位。
- 程序与进程的区别：进程不是程序，但由程序产生。程序只是一个静态的指令集合，不占系统的运行资源；而进程是一个随时都能发生变化、动态的、使用系统运行资源的程序。
- 可以这么认为：程序代表我们期望完成某工作的计划和步骤，它还浮现在纸面上，等待具体实现。而具体实现过程由进程来完成，它除了包含程序中的所有内容外，还可能包含一些额外的数据。
- 作业：正在执行的一个或多个相关进程可形成一个作业，一个作业可启动多个进程。

进程管理



- Linux是一个多用户、多任务的操作系统，在同一时间允许多个用户向操作系统发出各种操作命令。每当运行一个命令时，系统就会同时启动一个进程。Linux操作系统是多进程并发环境，进程之间呈现为多级结构。
- Linux系统在刚刚启动时，运行于内核方式，此时只有一个初始化进程在运行，该进程首先对系统进行初始化，然后执行初始化程序（即/sbin/init），初始化进程是系统的第一个进程，以后的所有进程都是初始化进程的子进程。
- 初始化进程通过函数调用fork()创建一系列的系统进程，如logger、update、cron等。

进程管理



- 为管理方便，Linux给每一个进程分配唯一的进程标识符PID，操作系统通过进程标识符管理进程。无论何时，当用户注册登录后，系统为用户创建一个shell进程（即用户的login shell）。输入一个命令或执行一个程序时，Shell进程会产生一个相应的子进程。
- 直接从终端读写的进程（作业）为前台进程(作业），正在运行而又无法直接从终端读写的进程为后台进程(作业）。在同一时刻，一个终端只有一个前台进程，但可以拥有多个后台进程。

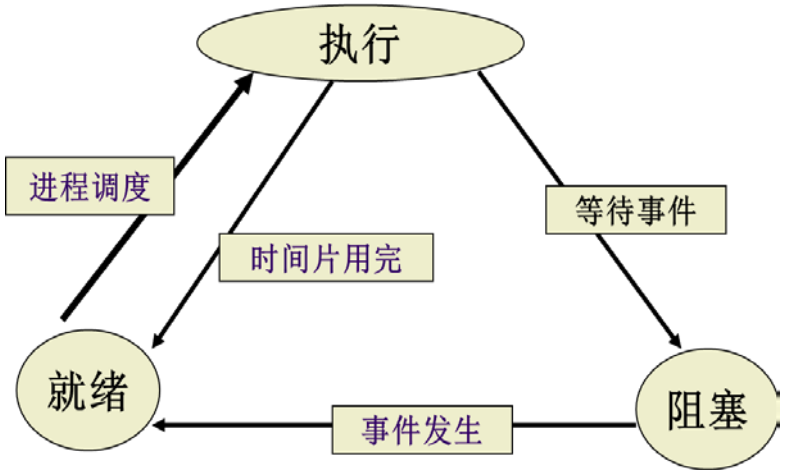
linux进程状态

- 通常在操作系统中，进程至少有三种基本的状态：

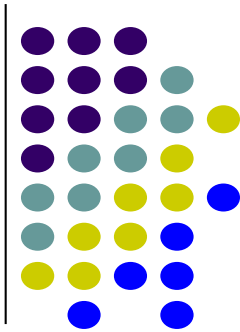
运行态：当前进程已分配到CPU，正在处理器上执行的状态。处于这种状态的进程个数不能大于CPU的数目。（分为核心态运行和用户运行态。）

就绪态：指进程已具备运行条件，但因为其他进程正占用CPU，所以暂时不能运行而等待分配CPU的状态。（分内存就绪和外存就绪。）

阻塞态：进程因等待某种事情发生（例如等待I/O操作，等待其他进程发来的信号等）而暂时不能运行的状态。即使CPU空闲，它也无法使用。



linux进程状态



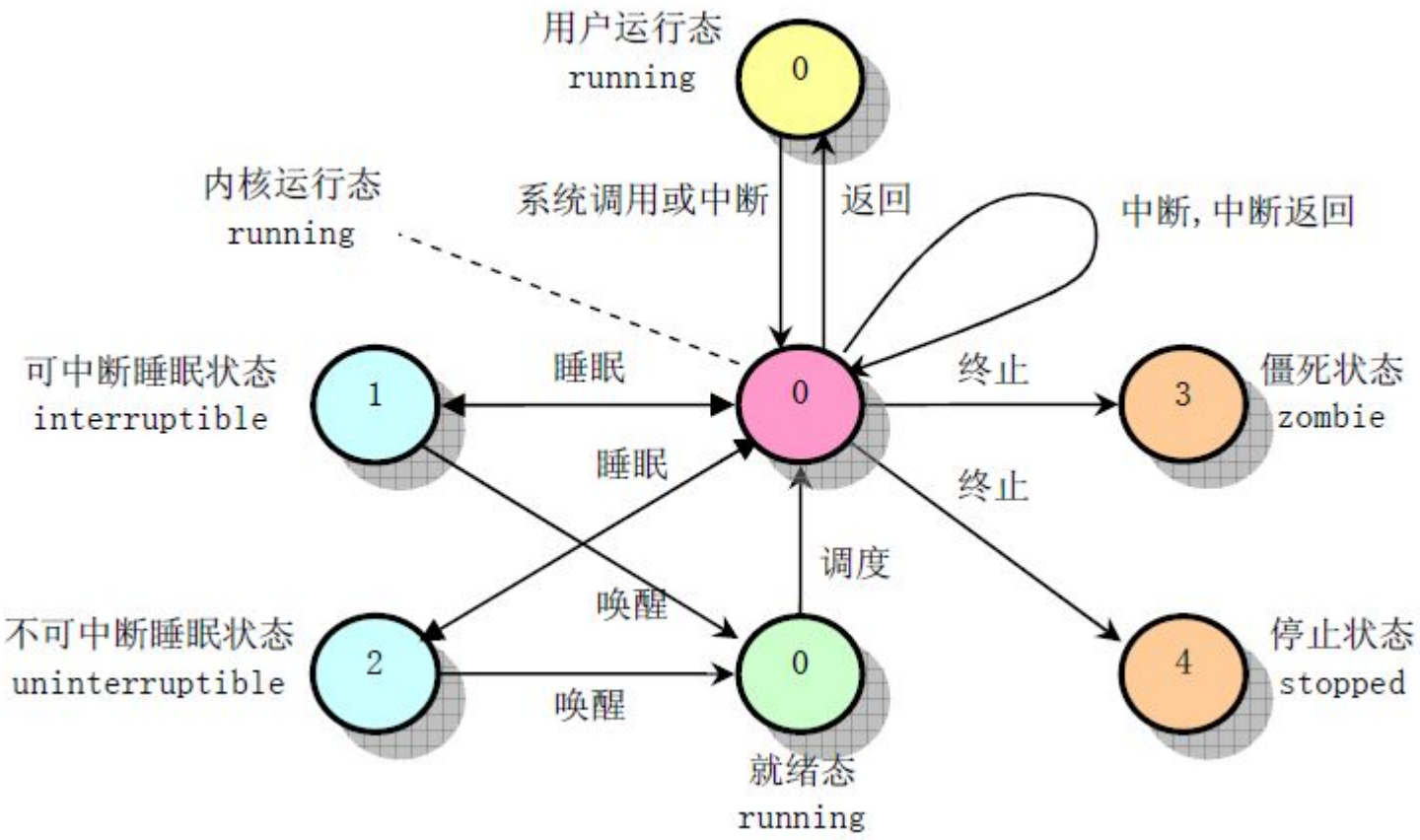
- 除了以上三种基本状态，Linux还描述进程的以下状态：

休眠状态：进程主动暂时停止运行（分可中断休眠和不可中断休眠）。

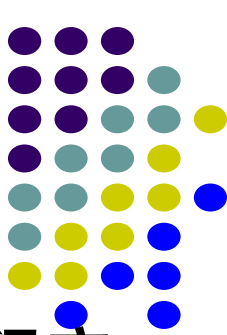
僵死状态：进程的运行已经结束，但它没有释放系统资源，如没有释放内存和task_struct结构等。

终止状态：进程已经结束，系统正在回收资源。

linux进程状态

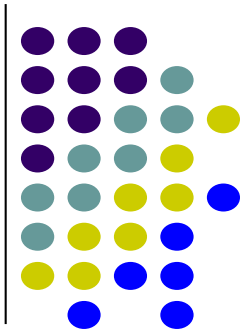

用户运行态
running
0
内核运行态
running
系统调用或中断
返回
中断, 中断返回
可中断睡眠状态
interruptible
1
睡眠
0
终止
3
僵死状态
zombie
睡眠
终止
调度
唤醒
不可中断睡眠状态
uninterruptible
2
0
4
停止状态
stopped
唤醒
就绪态
running

启动Linux进程



- 在系统中，键入需要运行的程序的名称，执行一个程序，其实就启动了一个进程。启动一个进程有两个主要途径：手工启动和系统调度启动。
- 手工启动是指由用户输入Shell命令后直接启动，可以分为前台启动和后台启动。
  - 用户输入Shell命令行后按回车键就启动了一个前台作业。这个作业可能同时启动多个前台进程。
  - 而在Shell命令行的末尾加上“&”符号，再按回车键，就启动一个后台作业。
- 调度启动是系统按照用户要求的时间或方式执行特定的进程，可分为at调度、batch调度、cron调度。

前后台切换



- bg命令

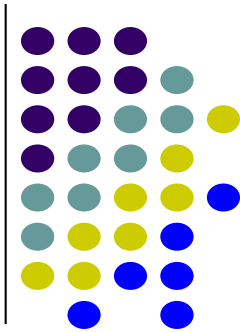
  格式：bg［作业号］

  功能：将前台作业切换到后台运行，功能上与在Shell命令行的末尾加上“&”符号类似。若未指定作业号，则将当前作业切换到后台。

例1：使用vi编辑f1文件，然后使用[Ctrl+Z]组合键挂起vi，在将其切换到后台。

```
[root@ Linux  root] # vi  f1
......
[1]+  Stopped                 vi  f1
[root@ Linux  root] # bg  1
[1]+  vi  f1  &
```

前后台切换



- fg命令

  格式：fg［作业号］

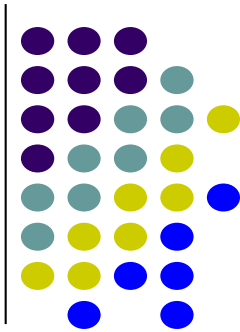
  功能：将后台作业切换到前台运行。若未指定作业号，则将后台作业序列中的第一个作业切换到前台。

例2：将例1中的作业号为1的作业切换到前台继续编辑。

［root@ Linux root］# fg 1

这时，系统进入vi编辑状态，用户可以继续编辑文件f1。

管理进程的Shell命令



显示作业号

- jobs命令

  格式：jobs［选项］

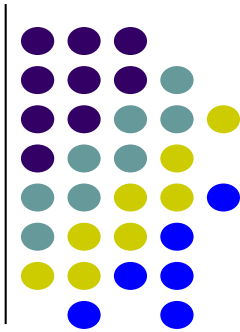
  功能：显示当前所有作业。

- 主要选项说明：

  -p　仅显示进程号

  -l　显示进程号和作业号

查看进程状态



ps命令

格式：ps ［选项］

功能：显示系统中当前的进程及其状态。

常用选项：

-a　　显示当前终端上所有用户的进程。

-e　　显示所有进程的信息。

-l　　显示进程的详细信息，包括父进程号、登录的终端号、进程优先级等。

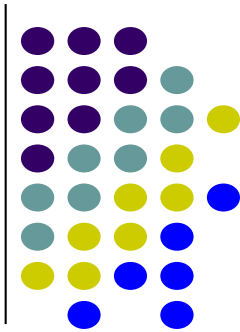
-u　　包括进程所有者在内的详细信息。

-x　　显示后台进程的信息。

-t终端号　　显示指定终端上的进程信息。

ps -ef |grep 列出需要进程

查看进程状态



例如，查看当前用户在当前控制台上启动的进程：

```
[root@ Linux  root] # ps
PID      TTY       TIME          CMD
2135     tty1      00:00:00      bash
3178     tty1      00:00:00      ps
```

显示信息分为4个字段，其中：

PID：进程号，系统根据这个编号处理相应的进程

TTY：表示登录的终端号，桌面环境或远程登录的终端号表示为pts/n(n为终端编号，从0开始依次编号)，字符界面的终端号表示为tty1～tty6，没有控制台的进程显示为“?”

TIME：表示该进程消耗的CPU时间

CMD：是command（命令）的缩写，往往表示正在执行的命令名

查看当前控制台上进程的详细信息。

［root@ Linux root］＃ps －l

该命令使用“-l”参数，它除了显示ps命令的4个基本字段外，另外还有10个附加信息可供查看。其主要输出项说明如下：

F：该进程状态的标记

S：进程状态代码。主要状态有以下几种：

- D：不可中断的休眠状态，常用于设备I/O
- R：运行状态
- S：休眠状态
- T：暂停或终止状态
- Z：僵死状态

UID：进程执行者的ID号

PPID：父进程的标识符

PRI：进程执行的动态优先级

NI： 进程执行的静态优先级

WCHAN：若该进程在睡眠，则显示睡眠中的系统函数名

SZ：进程占用内存空间的大小，以KB为单位

查看系统中每位用户的全部进程。

［root@ Linux root］# ps -aux

该命令显示系统中所有用户执行的进程，包括没有控制台的进程及后台进程。

主要输出项说明：

%CPU：CPU使用率百分比

%MEM：内存使用率百分比

VSZ：占用的虚拟内存大小

RSS：占用的物理内存大小

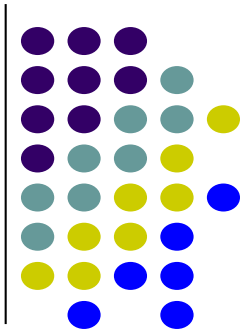
STAT：进程的状态

START：进程的开始时间

系统管理员常常配合grep命令的使用，以缩小查看的范围：

［root@ Linux root］# ps -aux | grep tom

pstree命令



- pstree命令

  格式：pstree［选项］

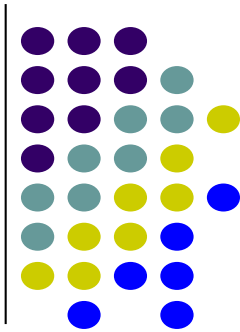
  功能：以树形图形显示进程之间的相互关系。

- 主要选项说明：

  -a　显示启动进程的命令行

  -n　按照进程号进行排序

终止进程



kill命令

格式：kill -9 进程号

功能：终止正在运行的进程或作业。超级用户可终止所有的进程，普通用户只能终止自己启动的进程。

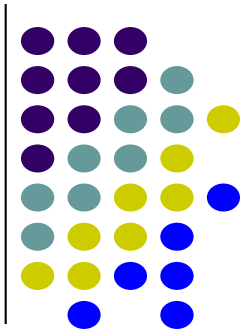
kill命令有许多选项，这里主要说明kill命令结束进程。

在终止进程前需要知道进程的pid，使用命令ps -e可以得到所要终止的进程的pid。

例3：系统管理员要终止进程httpd，首先使用ps命令查得进程httpd进程的pid为5198，然后使用kill命令终止该进程。

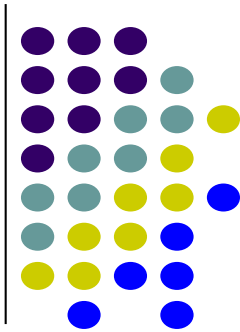
```
[root@ Linux root] # ps -ef |grep httpd
UID      PID    PPID   C    STIME    TTY    TIME    CMD
root     5198    1      0     08:45      1     09:10    httpd
[root@ Linux root] # kill -9 5198
```

终止进程



- 为什么要杀死进程
  - 该进程占用了过多的CPU时间
  - 该进程缩住了一个终端，使其他前台进程无法运行
  - 运行时间过长，但没有预期效果
  - 产生了过多到屏幕或磁盘文件的输出
  - 无法正常退出

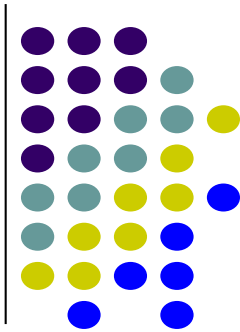
进程的挂起和恢复



- 进程的中止（挂起）和终止
  - 挂起（Ctrl+Z）
  - 终止（Ctrl+C）

- 进程的恢复
  - 恢复到前台继续运行（fg）
  - 恢复到后台继续运行（bg）

系统监视



who 命令

格式：who [选项]

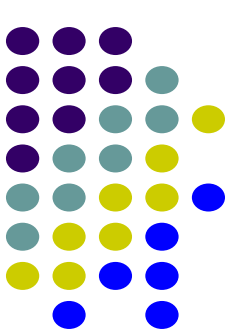
功能：查看当前已经登录的所有用户。

top 命令

格式：top [-d 秒数]

功能：动态显示CUP利用率、内存利用率和进程状态等相关信息。默认每5s更新显示信息，而“-d 秒数”选项可指定更新间隔。

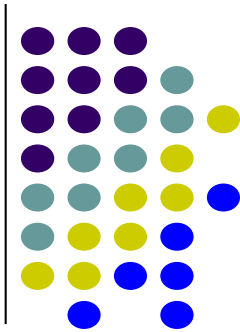
系统监视



Top命令显示的信息可分为上下两部分，上半部分显示当前时间、已运行时间，用户数；显示总进程及各种进程状态的进程数；还显示进程、内存和交换分区的使用情况。下半部分显示各进程的详细信息，默认按照进程的CPU使用率排列所有的进程。按[M]键将按照内存使用率排列所有进程，按[T]键将按照进程的执行时间排列所有进程，按[P]键将恢复按照CPU使用率排列所有进程，

按[Ctrl+C]组合键或[Q]键结束top命令。

指定进程优先级



- nice命令

  格式：nice -n command

  功能：指定程序的运行优先级

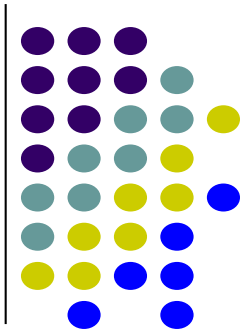
例如：

[root@ Linux root] #nice - -5 myprogram&

在后台以-5的优先级运行myprogram

其中n是为进程运行指定的优先级，n的数值越大，进程的优先级越低，n可以取负值让运行级提高。

指定进程优先级



- renice命令

  格式：renice -n pid

  功能：改变一个正在运行的进程的优先级

  是被指定的进程

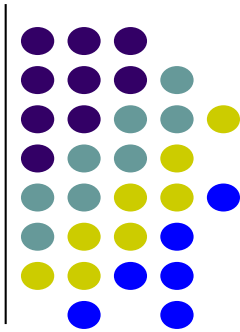
  例如：

  [root@ Linux root] # renice - -5 777

  将正在运行的PID为777的进程优先级改为-5

- Linux中进程优先级的取值范围是-20～19之间的整数，取值越高，优先级越低，默认优先级为0。启动进程的普通用户只能降低进程优先级，超级用户不但可以降低优先级，也可以提高优先级。

free命令



- free(选项)
- 格式：free　[选项]

  功能：显示内存和交换分区的使用情况。

  主要选项：

  -b：以Byte为单位显示内存使用情况；

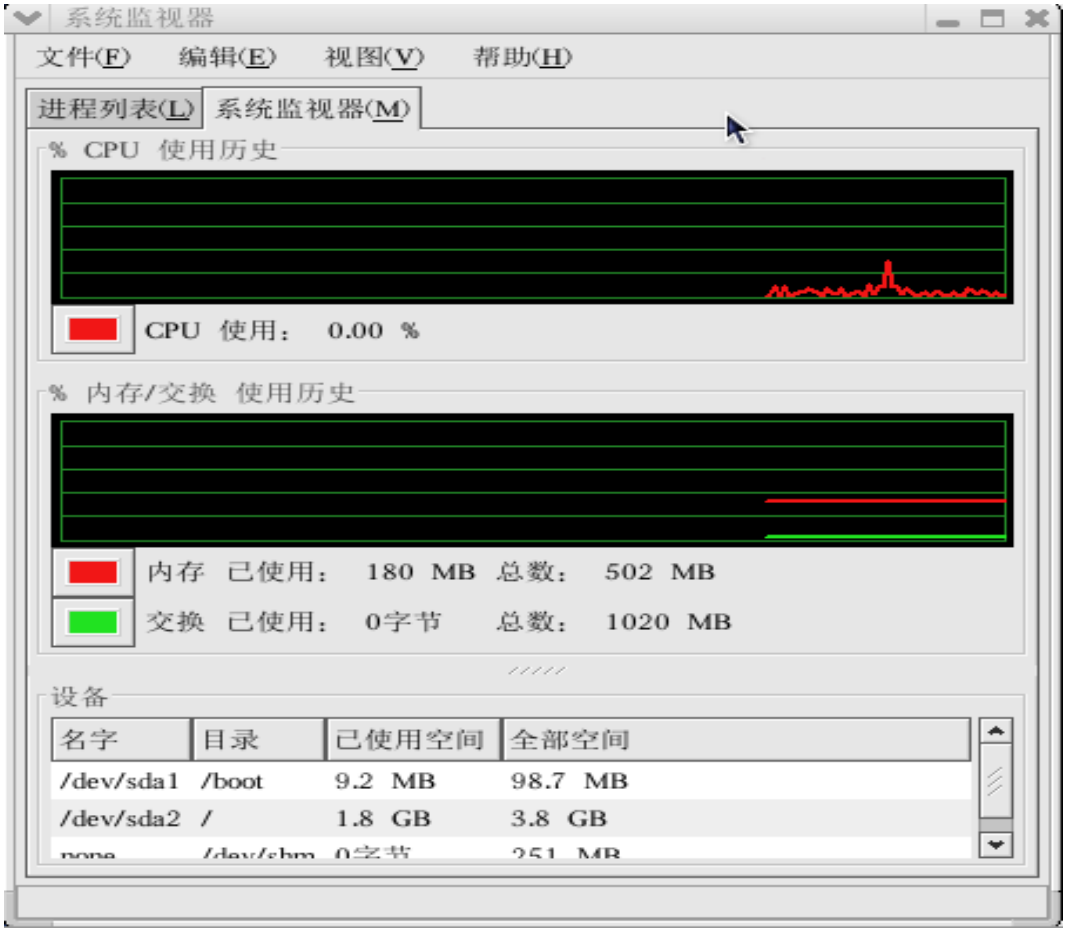
  -m：以MB为单位显示内存使用情况，默认以KB为单位；

  -s<间隔秒数>：持续观察内存使用状况；

  -t：显示内存总和列；

桌面环境下管理进程

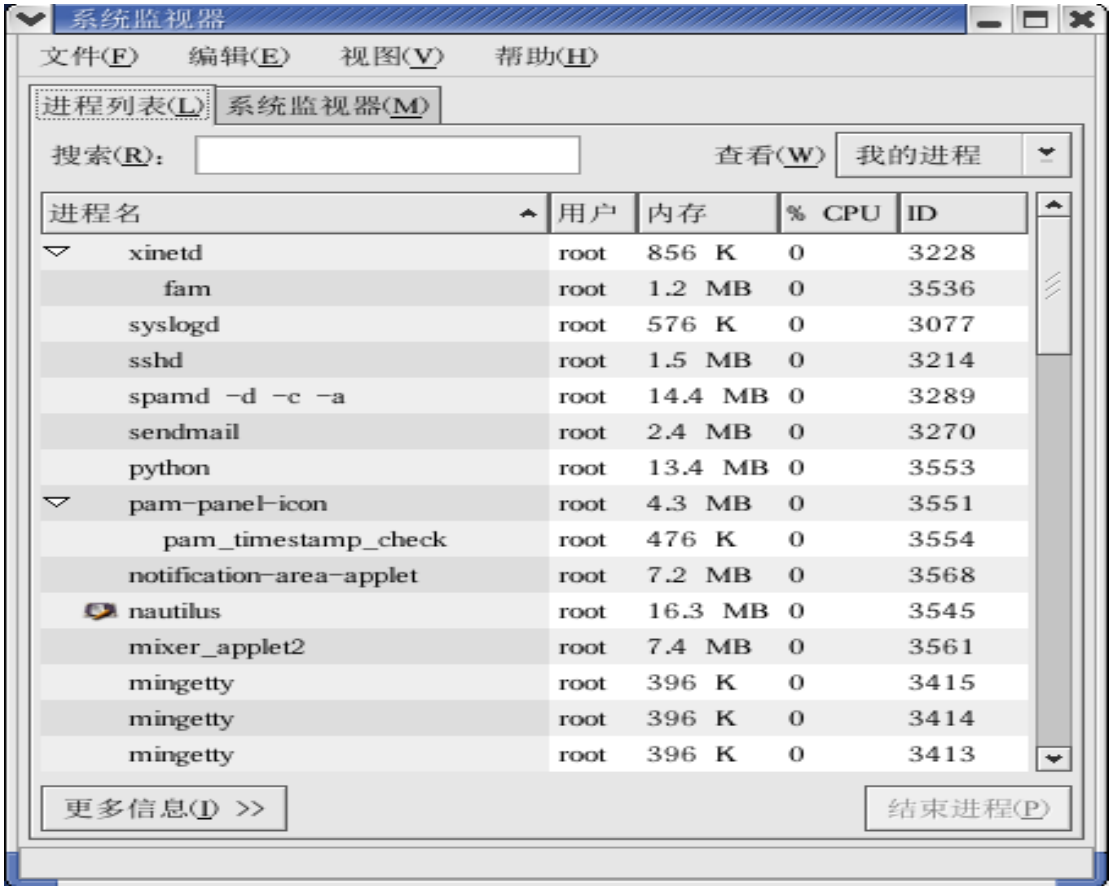
在桌面环境下依次选择“主菜单→系统工具→系统监视器”，系统监视器窗口显示了系统基本软硬件信息。



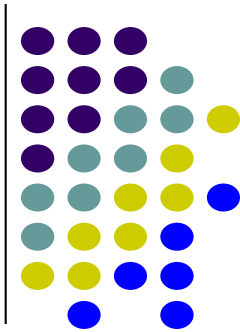
桌面环境下管理进程

查看进程：打开“进程列表”，显示进程属性信息。

用户可以通过“编辑”菜单改变进程的运行状态、进程优先级等。



启动进程



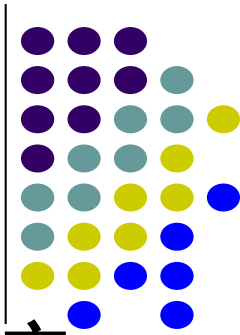
手工启动

由用户输入命令，直接启动一个进程便是手工启动进程。

例如：

- 前台启动：ls
- 后台启动：ls &

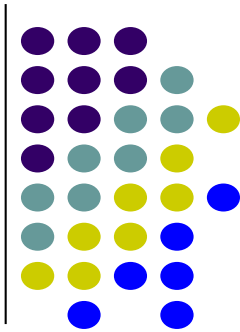
启动进程



调度启动是系统按用户要求的时间或方式执行特定的进程。Linux中可实现at调度、batch调度和cron调度：

- 在指定时刻执行命令序列—at命令
  - 有时候需要进行一些比较费时且占用资源的维护工作，例如网站数据库备份等，这些工作适合在深夜进行，这时用户就可以事先进行调度安排，指定任务运行的时间和场合，到时候系统会自动完成这一切工作。
  - 用户用at命令在指定时刻运行指定的命令序列。at命令可以只指定时间，也可以时间和日期一起指定。

at调度



at调度

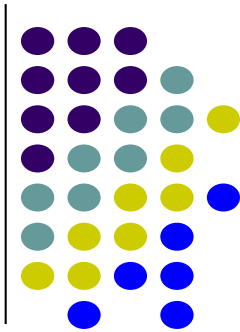
格式：at [选项] [时间]

功能：设置在指定时间执行指定的命令。

主要选项说明：

- **-l　显示等待执行的调度作业**
- **-d　删除指定的调度作业**
- **-v　显示作业执行的时间**
- **-m　作业结束后发送邮件给执行的at命令的用户**

at调度



进程的执行时间可采用以下方法表示：

（1）绝对计时法

HH:MM：指定具体的时间，默认采用24小时计时制。若采用12小时计时制，则时间后面需加上AM或PM.

MMDDYY、MM/DD/YY、DD.MM.YY：指定具体的日期，必须写在具体时间之后。年份可用两位数字表示，也可用四位数字表示。

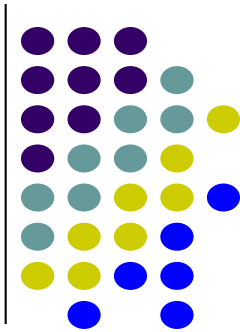
（2）相对计时法

now+时间间隔：时间单位为minutes(分钟）、hours(小时）、days(天）、weeks(星期）。

（3）直接计时法

today（今天）、tomorrow（明天）、midnight（深夜）、noon（中午）、teatime（下午四点）

at调度

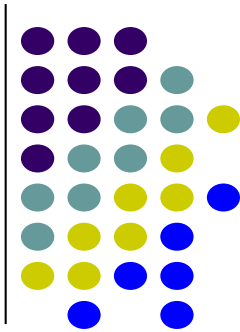


例4：设置at调度，要求在2018年12月31日23时59分向登录在系统上的所有用户发送Happy New Year信息。

```
［root@ Linux  root］#  at 23:59 12312018
at> wall Happy New Year!
at> <EOT>
job 1 at 2018-12.31 23:59
```

- 输入at命令后出现“at>”提示符，等待用户输入将执行的命令。输入完成后按“Ctrl+D”组合键结束，屏幕将显示该at调度的执行时间。

at调度



例5：查看at调度。

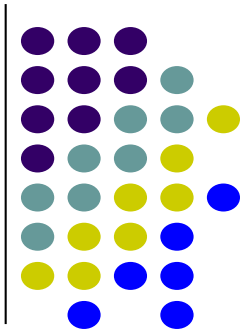
```
[root@ Linux  root]#  at  –l

  1     2018-12-31 23:59  a  root
```

例6：删除at调度。

```
[root@ Linux  root]#  at  –d 1
[root@ Linux  root]#  at  –l
[root@ Linux  root]#
```

batch调度



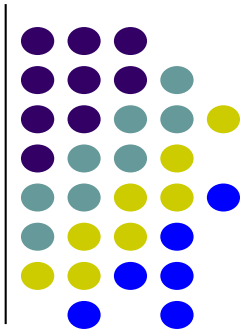
batch调度

格式：batch [选项] [时间]

功能：与at命令几乎一样，唯一的区别是at命令是在指定时间内很精确地执行指定命令，而batch命令，如果不指定运行时间，进程将在系统较空闲时运行。

- batch调度适合于时间上要求不高，但运行时占用系统资源较多的工作。batch命令的选项与at命令相同。通常不为batch命令指定时间参数，因为batch命令的特点就是由系统决定执行任务的时间。

batch调度



例7：使用batch命令执行根目录下查询文本文件的功能。

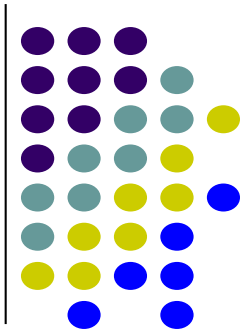
［root@ Linux root］# batch

at> find /-name *.txt

at> <EOT>

job 2 at 2018-5-10 22:00

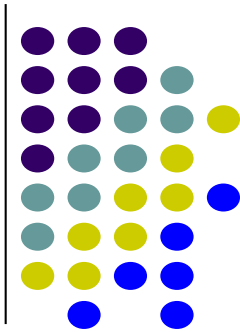
cron调度



cron调度

- at调度和batch调度中指定的命令只能执行一次。但在实际的系统管理中有些命令需要在指定的日期和时间重复执行，例如每天例行要做的数据备份。cron调度正可以满足这种需求
- corn命令通常是在系统启动时就由一个Shell脚本自动启动，并进入后台（所以不需要使用“&”符号）。
- cron命令运行时会搜索“/var/spool/cron”目录，寻找以用户名命名的crontab文件，找到就载入内存。

cron调度

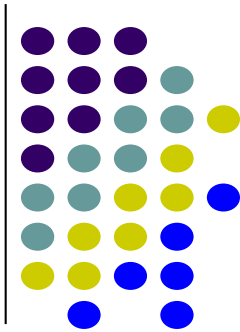


cron调度

- corn启动后，将先检查是否有用户设置了crontab文件，如果没找到，就转入“休眠”状态，释放系统资源，因此，该后台进程占用资源极少。它每分钟“醒”来一次，查看当前是否有需要运行的命令。命令执行结束后，任何输出都将作为邮件发送给crontab的所有者。
- 实际上，安排周期性任务的命令是crontab。该命令用于安装、删除或列出用于驱动cron后台进程的表格。crontab命令的基本格式如下：

格式：crontab [选项]

cron调度



格式：crontab [选项]

功能：管理crontab配置文件

主要选项说明：

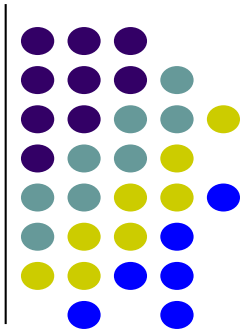
-e：创建并编辑crontab配置文件，如果不指定用户，则表示编辑当前用户的crontab文件。

-l：显示crontab文件内容，如果不指定用户，则表示显示当前用户的crontab文件内容。

-r：从/var/spool/cron目录中删除某个用户的crontab文件，如果不指定用户，则默认删除当前用户的crontab文件。

-i：在删除用户的crontab文件时给确认提示。

crontab配置文件

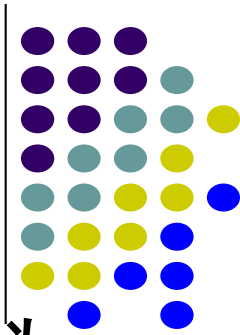


- 用户的crontab配置文件保存于/var/spool/cron目录中，其文件名与用户名相同(比如，helen用户的crontab配置文件为/var/spool/cron/helen)。
- crontab配置文件保留cron调度的内容，每一行表示一个调度任务。每个调度任务包括6个字段，所有字段不能为空，字段之间用空格分开：

f1	f2	f3	f4	f5	command
分	时	日	月	周	命令

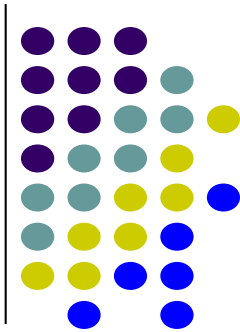
- 如果执行的命令未使用输出重定向，那么系统将会把执行结果以邮件的方式发送给crontab文件的所有者。

crontab配置文件



- 第一列f1代表分钟1~59：当f1为*表示每分钟都要执行；为*/n表示每n分钟执行一次；为a-b表示从第a分钟到第b分钟这段时间要执行；为a,b,c,...表示第a,b,c分钟要执行；
- 第二列f2代表小时0~23（0表示凌晨）：当f2为*表示每小时都要执行；为*/n表示每n小数执行一次；为a-b表示从第a小时到第b小时这段时间要执行；为a,b,c,...表示第a,b,c小时要执行
- 第三列f3代表日1~31：含义如上所示，以此类推；
- 第四列f4代表月1~12：含义如上所示，以此类推；
- 第五列f5代表星期0~6（0表示星期天）：含义如上所示，以此类推；
- 第六列command代表要运行的命令。

cron调度



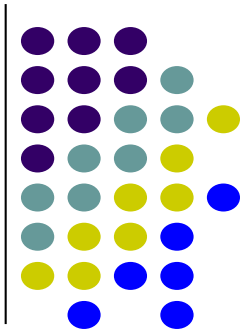
例8：helen用户设置cron调度，要求每周五的17时00分将/home/helen/data目录中的所有文件归档压缩为/backup目录中的helen-data.tar.gz文件。

［helen@ Linux helen］$ crontab –e

输入crontab –e 命令后，系统自动启动vi编辑器，输入以下配置文件的内容后保存退出

00 17 * * 5 tar –czf /backup/helen-data.tar.gz /home/helen/data

cron调度



例9：helen用户查看cron调度的内容。

［helen@ Linux helen］$ crontab –l

00 17 * * 5 tar –czf /backup/helen-data.tar.gz /home/helen/data

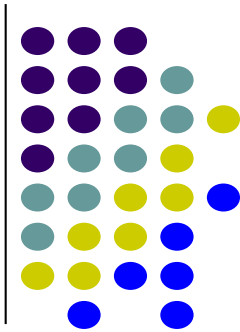
例10：helen用户删除cron调度。

［helen@ Linux helen］$ crontab –r

［helen@ Linux helen］$ crontab –l

no crontab for helen

练习

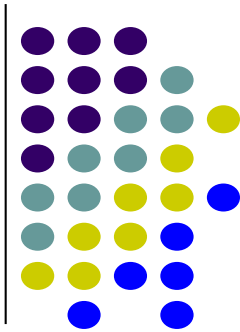


练习1：某系统管理员需每天需要重复下列工作，请按要求编制一个解决方案：

1. 在下午5:30删除/abc目录下的全部子目录和全部文件；

2. 从上午9:00～下午6:00每小时读取/xyz目录下test文件中后5行全部数据追加到/backup目录下的bake1.txt文件内；

3. 每周一下午5：50分将/data目录下的所有目录和文件归档并压缩为文件backup.tar.gz。



解答：

（1）用vi编辑器创建一个名为progx的crontab文件；

vi /root/grogx.cron

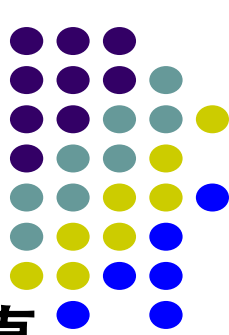
（2）progx文件的内容为：

30 17 * * * rm –rf /abc/*

00 9-18/1 * * * tail –5 /xyz/test>>/backup/bake1.txt

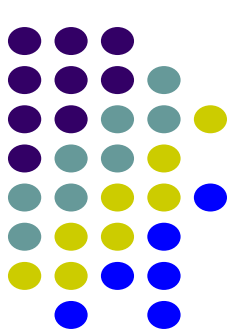
50 17 * * 1 tar –zcvf backup.tar.gz /data

内存管理

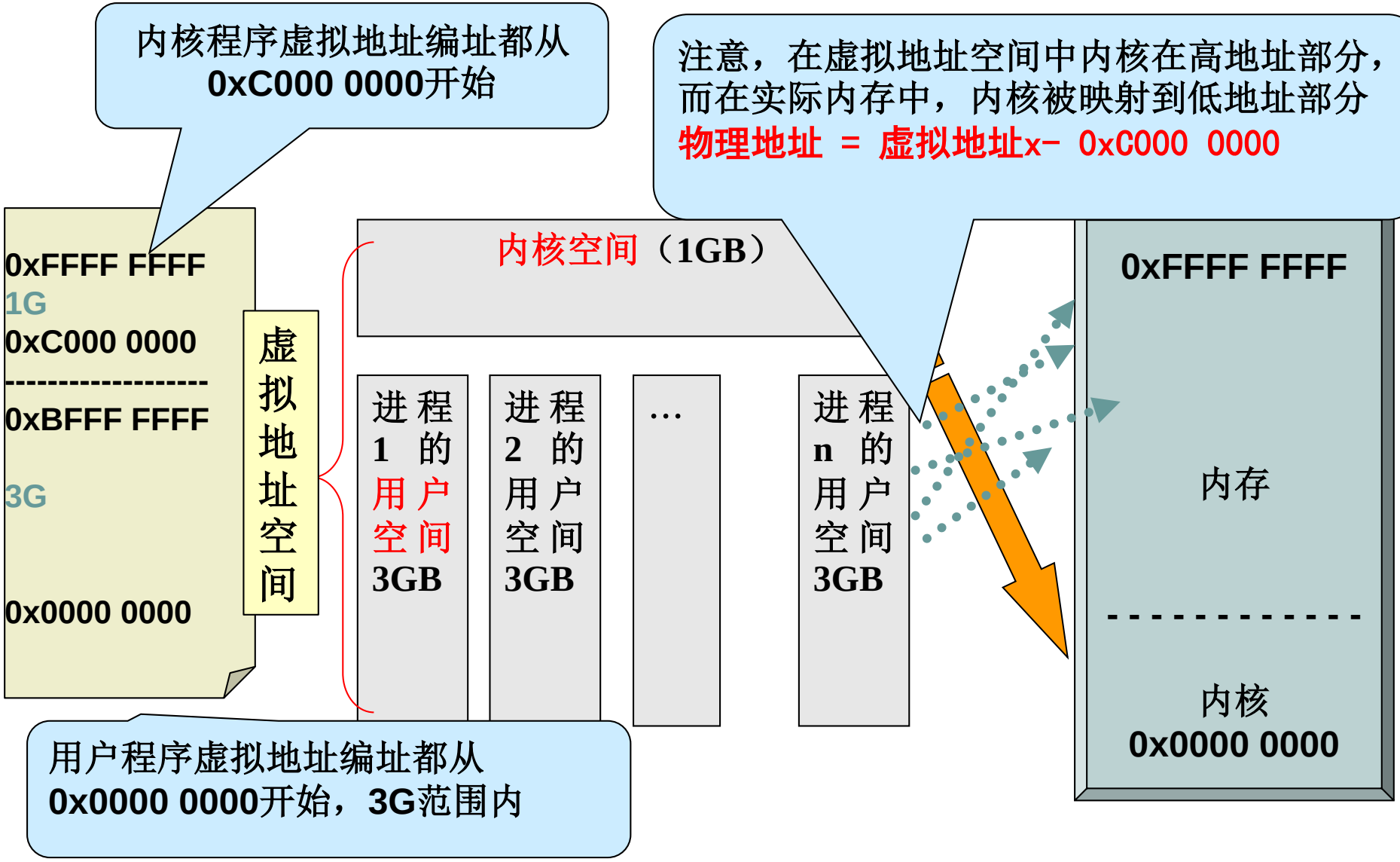


- 进程管理与内存管理密不可分，进程的创建过程就直接涉及内存空间的问题。
- 一个进程经过编译和链接之后，成为可执行文件。操作系统核心将可执行文件作为进程实体装入内存，进程实体分为正文段、数据段和堆栈段。
- 正文段由程序中的代码构成；数据段由程序运行所用到的数据构成；堆栈段由函数调用传递参数、保留现场、存放返回地址和变量等构成。
- 程序运行过程中，堆栈段由操作系统创建并不断更新。堆栈段分核心堆栈和用户堆栈。

内存管理

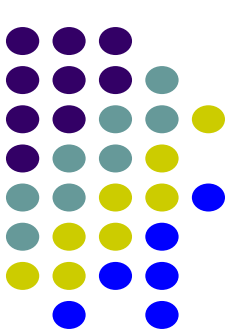


- 32位linux虚拟地址空间的大小是4GB。虚拟地址空间也就是程序员看到的内存空间。
- 虚拟地址空间：一个程序编译、连接后形成的地址空间就是一个虚拟地址空间。
- linux内核将4GB空间的高1GB供内核使用，称内核空间。
- 低3GB供各个进程使用，称用户空间（地址空间）
- 用户空间各自独立，每个进程通过系统调用进入内核，所以内核空间是进程共享的。



虚拟地址空间、内核空间和用户空间示意图

内核空间



- 我们把内核的代码和数据称为“内核映象”。内核被编译连接形成内核的虚拟地址，占用的是虚拟地址空间中从3G到4G的高1G的线性空间。
- 内核虚地址到物理地址的映射关系：

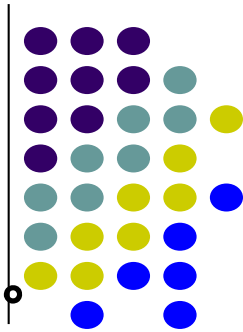
  物理地址 = 虚拟地址 - 0xC000 0000（ PAGE_OFFSET ）

  3GB(0xC0000000)就是物理地址与虚拟地址之间的位移量。
- 内核也有自己的内核页目录项和页表，可理解为页从内存0块开始正好一一对应，映射效果等价于上式。

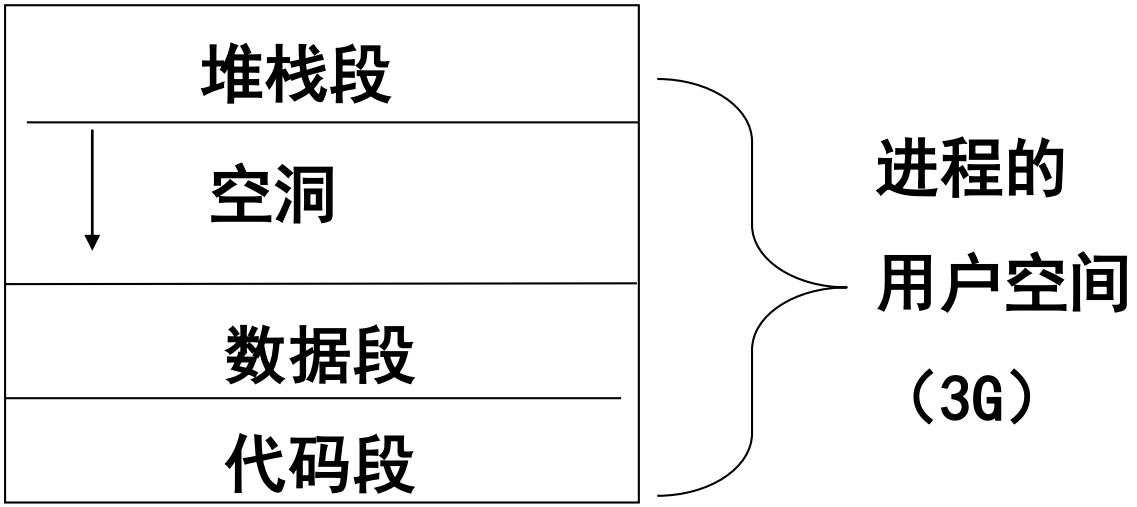
内核空间

- 从不同进程的角度看3G到4G的线性空间是有着同样的内核页目录项和页表的，从而对应同样的物理内存空间。
- 不同的进程通过系统调用可以访问内核空间，不同进程中访问该空间的虚拟地址对应的内容一样，所以可说3G到4G的内核空间是所有进程共享的。
- 进程之间的差异体现在0到3G用户空间所对应的页目录项和页表。

进程的用户空间

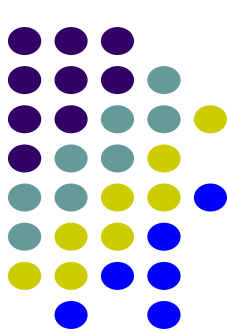


- 进程的用户空间中存放的是用户程序的代码和数据。每个进程最大拥有3G字节私有虚存空间。
- 利用分页机制，根据每个进程的页表进行用户空间的内存映射。
- 每个进程经编译、链接后形成的二进制映像文件有一个代码段和数据段。
- 进程运行时须有自己独占的堆栈。
- 空洞部分：进程运行时调用malloc（）动态分配的空间——堆。
- 进程的整个用户空间并不全部装入，而是通过请求页机制，部分映射装入。



- 数据段包含了所有静态分配的数据空间，即全局变量和所有声明为static的局部变量等，这些空间是进程所必需的基本要求。
- 堆栈段由函数调用传递参数、保留现场、存放返回地址和局部变量构成。
- 堆栈段安排在用户空间的顶部，运行时由顶向下延伸；代码段和数据段在底部，运行时并不向上延伸。

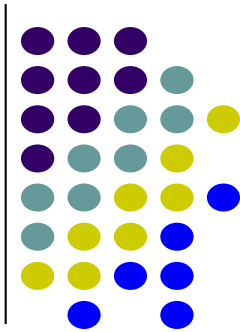
进程描述符



- 进程描述符是Linux系统用户描述进程的数据结构，进程描述符用结构task_struct表示：

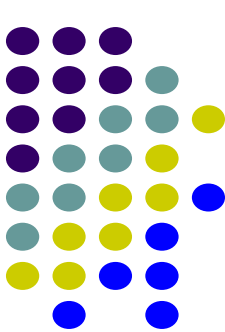
```
task_struct{
        long state;     /*进程状态*/
        int pid,uid,gid;   /*一些标识符*/
        struct task_struct  *parent, *child,
  *o_sibling, *y_sibling    /*一些亲属关系*/
   …
}
```

进程描述符包含的信息



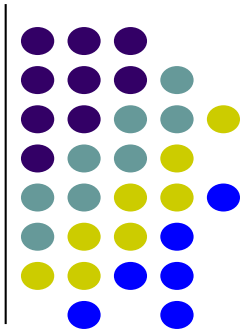
- 状态信息－描述进程动态的变化。
- 链接信息－描述进程的父／子关系。
- 各种标识符－用简单数字对进程进行标识。
- 进程间通信信息－描述多个进程在同一任务上协作工作。
- 时间和定时器信息－描述进程在生存周期内使用CPU时间的统计等信息。
- 调度信息－描述进程优先级、调度策略等信息。
- 文件系统信息－对进程使用文件情况进行记录。
- 虚拟内存信息－描述每个进程拥有的地址空间。
- 处理器环境信息－描述进程的执行环境(处理器的寄存器及堆栈等)

9.2 进程的创建



- Linux系统中创建进程的方法是用系统调用fork（）。调用系统调用 fork（）的进程是父进程，所创建的新进程为子进程。除进程0 之外，系统中所有的进程都是系统调用fork（）创建的。
- 系统调用 fork（）的实现过程首先是由内核为新进程在进程表中分配一个表项并给新进程一个唯一的进程标识符PID；
- 然后通过拷贝父进程的进程映象来创建一新进程，新进程将获得其父进程地址空间的一份拷贝；最后增加与该进程相关联的文件表和索引节点表的引用数，并将子进程的进程标志返回给父进程，收到父进程返回信号则标志新进程创建完毕。

Linux进程的创建步骤



fork（）系统调用生成子进程的步骤如下：

（1）如果系统内存够，则能创建进程，否则调用退出；

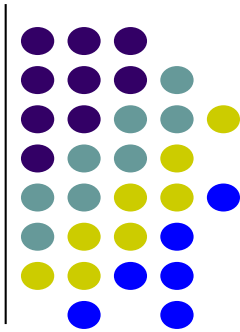
（2）为要创建的进程分配一个空闲的进程表项和唯一的进程标识符PID;

（3）如果进程所在的用户其进程数没超过系统限制，则能创建进程；否则调用退出；

（4）设置进程状态为创建，拷贝父进程表项中数据到进程表项中

（5）修改新进程描述符，将新进程标识符写入新进程描述符创建到父进程的链接；初始化进程创建的时间、需要运行的时间等新进程私有信息。

Linux进程的创建步骤

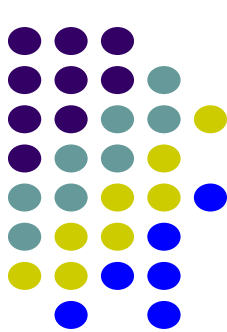


（6） 将当前目录的索引节点引用数和文件 表中打开文件的引用数加1；

（7） 拷贝父进程上下文到进程的上下文；

（8） 如果正在执行的是其父进程则设置进程状态为就绪，否则初始化u 区的计时域。

Linux进程的创建



- 在系统允许的条件下，用户可以按需要创建多个进程。

  系统函数调用格式如下：

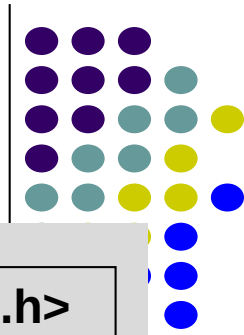
  pid=fork()

  pid是新创建进程的标识符。

- 如果该函数调用成功，返回值有两个：pid=0的返回值为执行子进程的代码；pid>0的返回值为执行父进程的代码；如果系统调用失败，返回值为-1。通常情况如下：

```
if(pid==0)
call subprocess_program;
else if(pid>0)
complete parent_process_program;
```

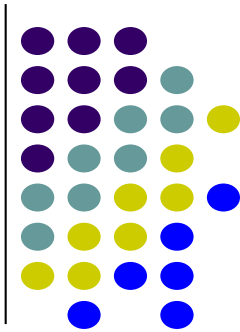
进程示例



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

```
 main()
{ int pid;
 printf(Now this is process1.\n”);
 printf(System calling fork() will start.\n”);
 pid = fork();
 if (pid ==0)
 printf(“This is child process.\n”);
 else  if(pid>0)
 printf(“This is process1(parent process).\n”);
  else
 printf(“fork faild.\n”);
 printf(“The program end.\n”);
 exit(0);
}
```

进程示例



编译并运行这个程序：

```
$gcc fork_test1.c -o fork_test1
```

执行：

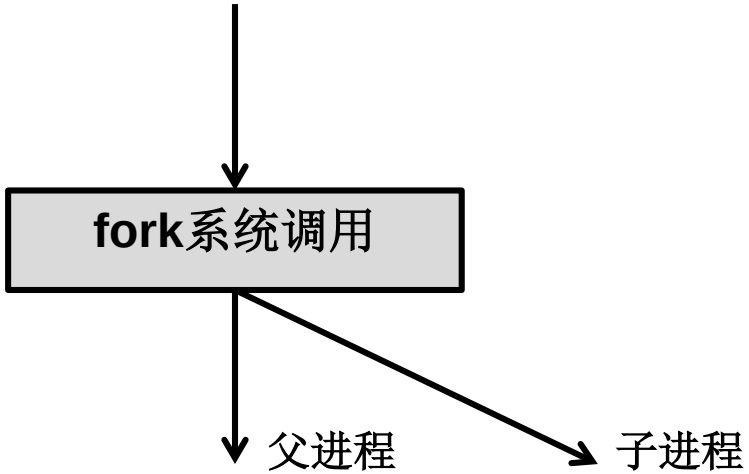
```
$./fork_test1
```

输出结果

```
  Now this is process1.
  System calling fork() will start.
  This is child process.
  The program end.
  This is process1(parent process).
  The program end.
```

创建进程

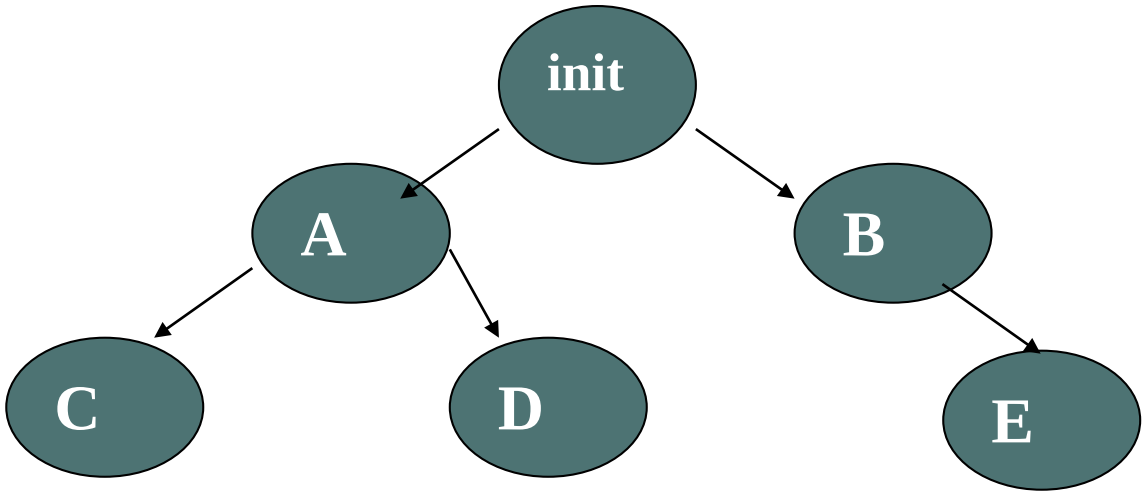
- 进程创建时，新进程共享了父进程的代码段，复制了父进程的数据段。在新进程创建成功后，父子进程并发执行，各自独立的被调度运行。 如果子进程结束，父进程希望能收回控制权，所以子进程不能覆盖父进程给予的控制区。



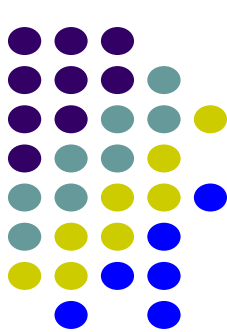
Fork系统函数调用图

进程层次

- linux启动时创建一个特殊进程init，以后诞生的进程都是其子进程、孙进程。
- 该进程为每个终端进程创建管理进程，等待用户登陆；用户登陆后再为每个用户启动一个shell进程接收用户命令。

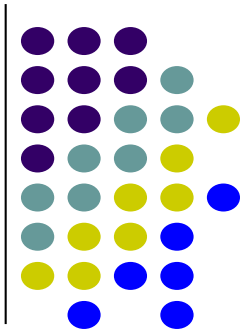


创建进程



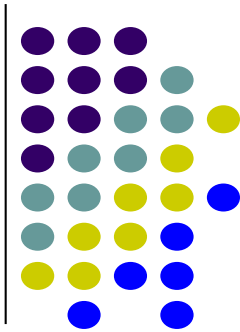
- 从概念上讲，fork()就像细胞的裂变，调用fork()的进程是父进程，而新裂变出的进程就子进程。调用fork()后，子进程被创建，此时父进程和子进程都从这个系统调用内部继续运行。
- fork函数的特点就是“调用一次，返回两次”，在父进程中调用一次，在父进程和子进程中各返回一次。
- 在Linux运行的每个进程进程标识符PID，它就是进程的身份证号码，每个人的身份证号码不同，每个进程的进程ID也不相同。系统调用getpid()就是获得进程标识符的。

进程示例



```
#i nclude<stdio.h>
# include<sys/types.h>
# include<unistd.h>    /* 系统调用的定义所在头文件 */
main()
{
      pid_t pid;          /*此时仅有一个进程*/
         printf("PID before fork():%d\n", (int)getpid() );
      pid=fork();             /*此时已经有两个进程在同时运行*/
    if(pid<0)        printf("error in fork!");
     else if(pid==0)
          printf("I am the child process, my process ID is %d\n",getpid());
      else
          printf("I am the parent process, my process ID is %d\n",getpid());
}
```

进程示例



编译并运行这个程序：

```
$gcc fork_test2.c -o fork_test2
```

执行：

```
$./fork_test2
```

输出结果

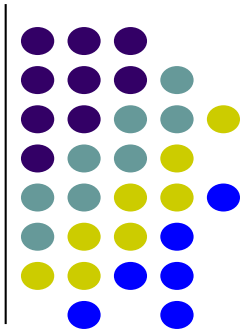
```
  PID before fork()：3857
I am the child process, my process ID is 3858
I am the parent process, my process ID is 3857
```

练习2：实现下面的程序，并分析结果。

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

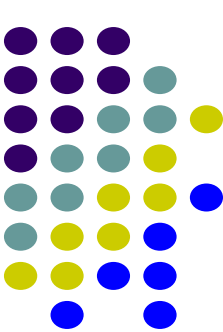
```
int main()
{ pid_t pid;
 int  count =0;
 pid = fork();
 if (pid < 0)
      print("fork failed !");
  else if (pid == 0)
     {
       printf("I am the child process, my id is %d\n",getpid());
       printf("我是父亲的孩儿\n");
       count++;
      }
   else
     { printf("I am the parent process,my id is %d\n",getgid());
        printf("我是孩儿他爹/n");
        count++; }
  printf("统计结果是：%d\n",count);
 return 0;
}
```

结束进程



- 代码运行结束或进程中有系统调用exit（）都会结束进程。结束进程需要释放进程所占用的一切资源并通知父进程。
- 在Linux核心中，结束进程的源代码程序为Linux/Kernel/exit.c。该程序包含进程释放、会话终止和程序退出处理函数，以及杀死进程、进程终止、挂起进程以及进程信号发送函数send_sig()和通知父进程子进程终止函数tell_father()等。

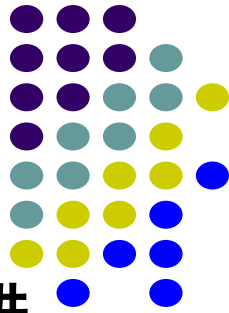
9.3 进程调度



进程相关参数

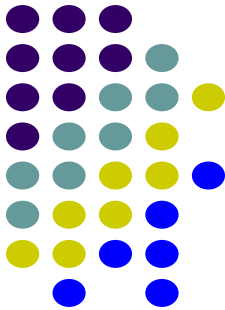
- Linux进程调度首先将进程分为实时进程和普通进程，并分别采用不同的调度策略。
- 普通进程的实时优先级（rt_priority)为0；如果为实时进程，则其实时优先级大于0。实时进程调度总是优先于普通进程调度。
- 进程调度准则以CPU的时间片为单位，并根据进程相关参数policy、priority、counter、rt_priority的值进行调度。这些参数在结构task_struct中定义，其中：
  - policy：进程的调度策略，用来区分实时进程和普通进程；
  - priority：进程(包括实时和普通)的静态优先级；
  - counter：进程剩余的时间片，在应用中，它的起始值就是priority的值；
  - rt_priority：实时进程特有的实时优先级，用于实时进程间的选择。

进程调度



- Linux用函数goodness()来衡量一个处于可运行状态的进程值得运行的程度。该函数综合了以上提到的四项，还结合一些其他的因素，给每个处于可运行状态的进程赋予一个权值，调度程序以这个权值作为选择进程的唯一依据。
- 对于普通进程，Linux采用动态优先调度（基于动态优先级的时间片多级队列进程调度）。进程创建时，优先级priority被赋一个初值，这个数字同时也是计数器counter的初值。进程运行过程中，counter值不断减少，而priority值不变。
- 动态优先级高的进程优先获得CPU的运行权，被CPU执行一个时间片后，被更高优先级的进程或处于普通队列队首的进程抢占，被抢占后的进程进入下一个就绪队列。

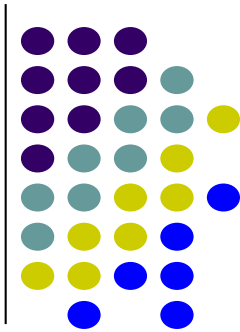
动态优先级进程调度



- 一个普通进程的时间片用完之后，并不马上用priority对counter进行赋值，直到所有处于就绪状态普通进程的时间片都用完了以后，才令counter等于priority，并对counter重新赋值，这时普通进程才有了再次被调度的机会。
- 普通进程运行过程中，counter的减小给了其它进程得以运行的机会，直至counter减为0时才完全放弃对CPU的使用。
- 时间片：Linux的时间单位是“时钟滴答”，不同操作系统对一个时钟滴答的定义不同而已（Linux为10ms）。进程的时间片就是指多少个时钟滴答，比如，若priority为20，则分配给该进程的时间片就为20个时钟滴答，也就是：

  20*10ms=200ms。
- Linux中进程的调度策略(policy)、优先级(priority)等可以作为参数由用户自定，具有相当的灵活性。

进程调度

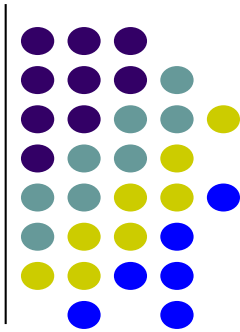


- 对于实时进程，Linux采用了两种调度策略：

  即FIFO(先来先服务调度)和RR（时间片轮转调度）。对应这两种调度策略，实时就绪进程被组织成HED_FIFO和SCHED_RR队列。

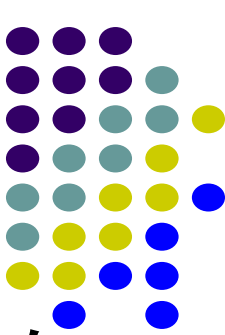
- 实时进程具有一定程度的紧迫性，用rt_priority来表现进程的紧迫程度。实时进程的counter只用来表示该进程的剩余时间片，并不作为衡量它是否值得运行的标准，这和普通进程是有区别的。

进程调度



- 对于先来先服务调度策略，实时优先级高的进程更紧迫，可优先分配CUP。在运行过程中，如果没有优先级更高的进程，则一直运行，直到完成；如果有优先级更高的进程，则会被抢占，进程放弃CUP，被放到SCHED_FIFO队列的末尾排队。
- 对于时间片轮转调度策略，在实时进程运行过程中，counter为进程的剩余时间片。只要时间片没有用完，进程会一直运行，直到有一个进程因I/O阻塞，或主动释放CPU,或者是CPU被另一个具有更高rt_priority的实时进程抢先，进程放弃CPU,被放到SCHED_RR队列的末尾排队。

9.4 Linux守护进程



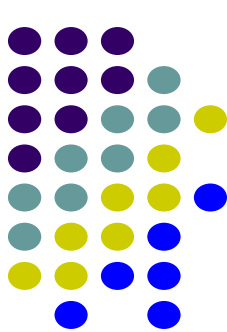
Linux操作系统包括3中不同类型的进程，每种进程都有自己的特点和属性。

（1）交互进程：由shell启动,可以工作在前/后台。

（2）批处理进程：这种进程与终端没有联系，是一个进程序列。

（3）守护进程：Linux系统自动启动，工作在后台，用于监视特定服务。

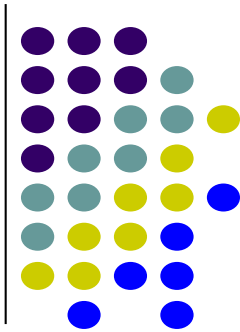
Linux守护进程



一、守护进程简介

- 守护进程（Daemon，也称为精灵进程）是Linux系统3种进程之一，也是非常重要的一种。守护进程可以完成很多重要的工作，包括系统管理和网络服务等。
- 守护进程生存期长。它们独立于控制终端并且周期性地运行来执行某种特定的任务或连续运行，等待处理系统中某些发生的事件（例如xinetd和lpd）。守护进程通常在系统引导装入时启动，在系统关闭时终止。这些进程没有控制终端，在后台运行。
- 守护进程与后台进程的区别:后台运行的程序拥有控制终端，守护进程没有。

守护进程的启动



启动守护进程有如下方式：

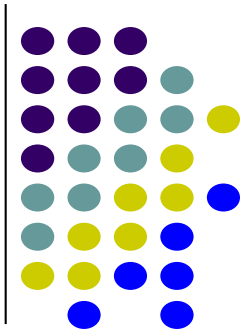
1. 在系统启动时由系统初始化脚本启动，这些脚本一般在/etc或／etc/rc开头的目录。如ineted超级服务器，web服务器等；

2. 人工手动从Shell提示符启动，任何具有相应的执行权限的用户都可以使用这种方法启动守护进程。

3. 使用cron守护进程启动，corn守护进程按一定的规则执行一些程序，由它启动的程序也以守护进程的方式运行。

4. 执行at命令启动，在规定的日期执行某个任务。

守护进程工作方式

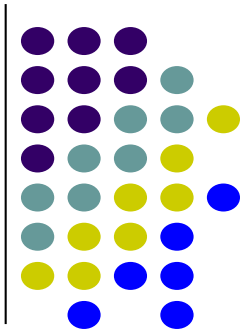


守护进程的工作方式

（1）运行独立的守护进程

- 独立运行的守护进程由init脚本负责管理，所有独立运行的守护进程的脚本在/etc/rc.d/init.d/目录下。系统服务都是独立运行的守护进程包括：syslogd和cron等。
- 运行独立的守护进程工作方式称作：stand－alone。工作在stand－alone模式下的网络服务有route、gated、Web服务器Apache和邮件服务器Sendmail、域名服务器Bind等。

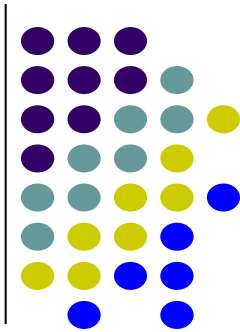
守护进程工作方式



（2）xinetd模式

- 从守护进程的概念可以看出，对于系统所要通过的每种服务，都必须运行一个监听某个端口连接所发生的守护进程，这意味着资源浪费。为解决这个问题，Linux引进了“网络守护进程服务程序”的概念。
- 与独立运行模式相比，xinetd模式也称超级服务器。xinetd能够同时监听多个指定的端口，在接受用户请求时，他能根据用户请求的端口不同，启动不同的网络服务进程来处理这些用户请求。可以把xinetd看做一个管理启动服务的管理服务器，它决定把一个客户请求交给哪个程序处理，然后启动相应的守护进程。

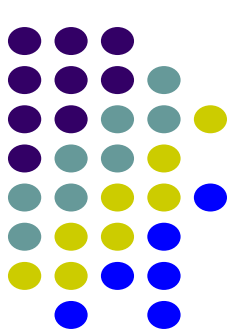
守护进程工作方式



守护进程的工作方式

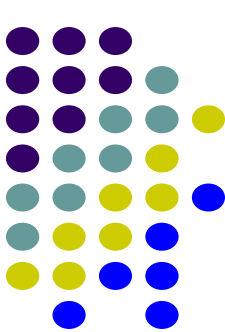
和stand－alone工作模式相比，xinetd模式下系统不需要每一个网络服务进程都监听其服务端口。运行单个xinetd就可同时监听所有服务端口，这样就降低了系统开销。但是对于访问量大、经常出现并发访问时，xinetd想要频繁启动对应的网络服务进程，反而会导致系统性能下降。

9.5 进程通信



- Linux系统是多进程并发，进程间为实现相互制约和合作需要彼此传递消息。然而每个进程都只在自己独立的地址空间中运行，无法直接访问其他进程的空间，因此，当进程需要交换数据时，必须采用某种特定的手段，这就是进程通信。进程通信（Inter-Process Communication,IPC）是进程间采用某种方式相互传递消息，少则是一个数值，多则是一大批字节数据。

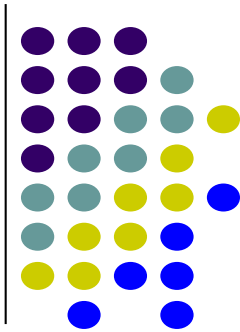
进程通信



进程间通信有以下目的：

- 数据传输：一个进程需要将它的数据发送给另一个进程，发送的数据量在一个字节到几兆字节之间。
- 共享数据：多个进程想要操作共享数据，一个进程对共享数据的修改，别的进程应该立刻看到。
- 通知事件：一个进程需要向另一个或一组进程发消息，通知它（它们）发生了某种事件（如进程终止时要通知父进程）。
- 资源共享：多个进程之间共享同样的资源。为了作到这一点，需要内核提供锁和同步机制。
- 进程控制：有些进程希望完全控制另一个进程的执行（如Debug进程），此时控制进程希望能够拦截另一个进程的所有陷入和异常，并能够及时知道它的状态改变。

进程通信方式



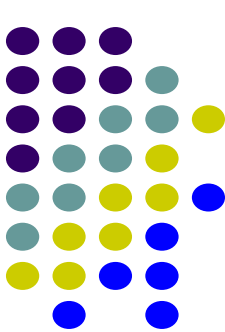
Linux中使用的几种进程通信方式：

（1）信号(Signal)：信号是进程间可相互发送控制信息，一般只有几个字节的数据，传递的信量小。信号是在软件层次上对中断机制的一种模拟，是比较复杂的通信方式，用于通知接受进程有某事件发生，一个进程收到一个信号与处理器收到一个中断请求效果上是一样的。

（2）管道（pipes)：管道是连接两个进程的一个数据传输通路，一个进程向管道写数据，另一个进程从管道读数据，实现两进程之间同步传递字节流。管道的信息传输量大、速度快。内置同步机制，使用简单。

管道可用于具有亲缘关系进程间的通信。有名管道，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。

进程通信方式

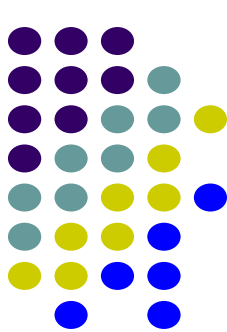


（3）消息队列：消息是结构化的数据，消息队列是由链接而成的链式队列。具有写权限的进程可以向消息队列中按照一定的规则添加新消息，有读权限的进程则可以从消息队列中读取消息。

与管道不同的是，这是一种异步的通信方式，发送方把消息送入消息队列中，然后继续运行；接收进程在合适的时机去取消息。相比信号来说，消息队列传递的信号量更大，能够传递格式化的数据。

消息通信是异步的，适合于在异步运行的进程间交换信息。

进程通信方式

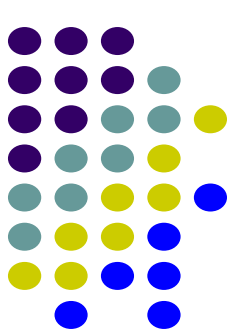


（4）共享内存：最有用的进程间通信方式。在内存中开辟一段存储区，将这个区映射到多个进程的地址空间，使得多个进程可以访问同一块内存空间。

通信双方直接读写这个存储区，即可达到数据共享的目的。由于共享内存区就在进程自己的地址空间内，因此访问额速度最快，只要发送进程将数据写入共享内存，接收进程就可立即得到数据。

适合传递大量的、实时的数据。但它没有内置同步机制，需要配合信号量或互斥锁等实现进程的同步。较之管道，共享内存的使用较复杂。

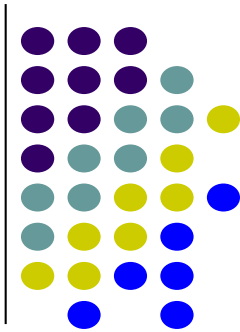
进程通信方式



（5）信号量：主要作为进程间以及同一进程不同程之间的同步手段。信号量是一组原子操作，分别用P,V表示。P操作表示减少一个信号量的值，V操作表示增加一个信号量的值。

信号量可以用来保证两个或多个关键代码段不被并发调用。在进入一个关键代码段之前，进程或线程必须获取一个信号量；一旦该关键代码段完成了，那么该进程或线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个进程或线程释放信号量。

进程通信方式



（6）套接字（Socket）： 这是一种更为一般的进程间通信机制，它可用于不同机器之间的进程间通信，应用非常广泛。TCP用主机的IP地址加上主机上的端口号就叫做套接字或插口。

套接字是网络通信过程中端点的抽象表示，包含进行网络通信必需的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。