西安理工大学
XI'AN UNIVERSITY OF TECHNOLOGY

Linux基础及应用
课程要点复习

王一川
chuan@xaut.edu.cn
https://jsj.xaut.edu.cn/info/1049/1049.htm
金花校区教六1114-2
计算机学院，网络工程系

考试形式

上机实验考试：

老师分组进行检查，检查老师从实验指导书中任意抽取5道题，检查执行结果，并询问学生对代码的理解情况，每题8分，共计40分。

闭卷卷面考试：

判断题：　1分*8=8分

填空题：　1分*8=8分

名词解释：5分*4=20分

简答题：　5分*4=20分

应用题：7分*2=14分，可能涉及:加注释、写结果、解释函数含义、画流程图等。

编程题：3道题，7+8+15=30分。

总成绩=平时成绩（10分）+实验考试（40分）+卷面成绩*50%

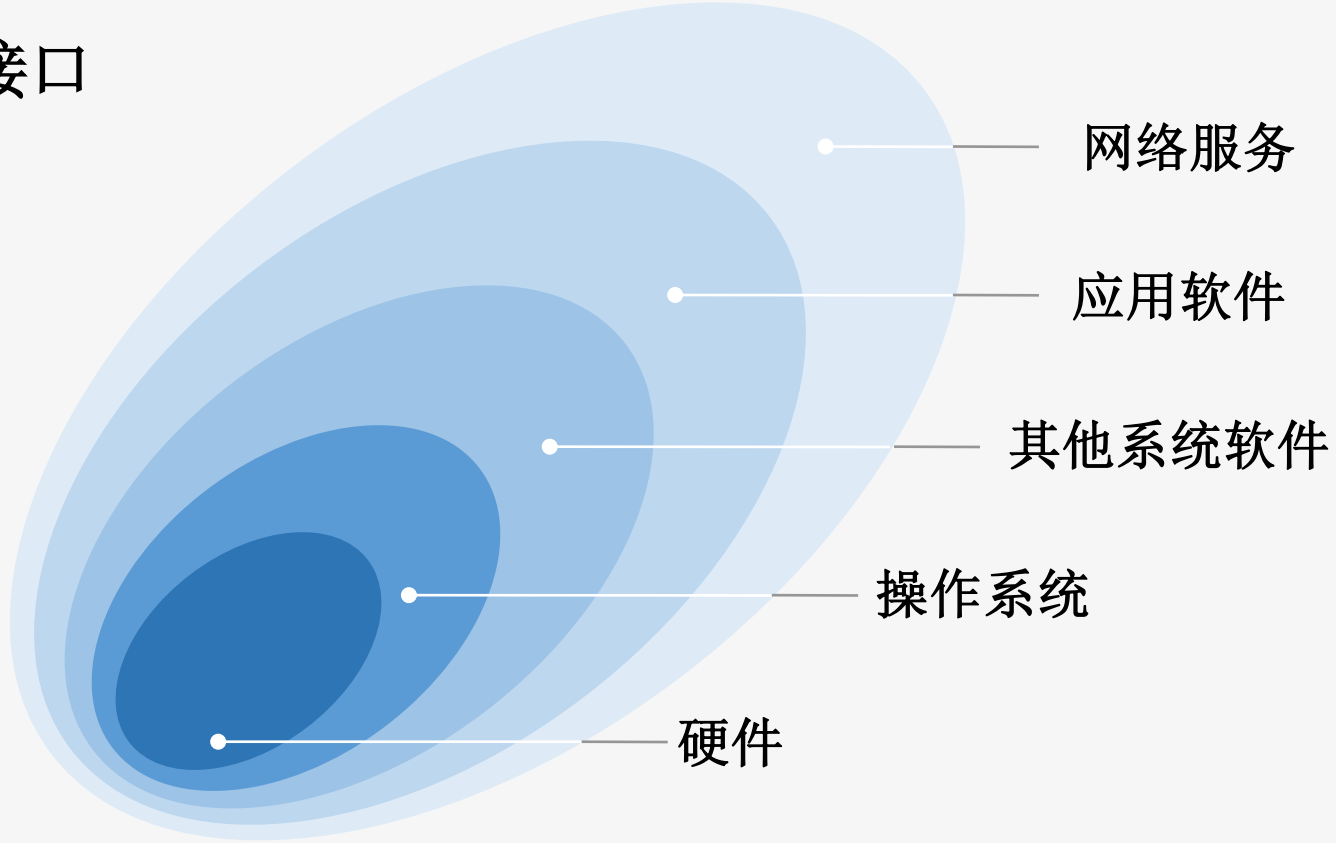
什么是计算机操作系统?

操作系统是用于管理和控制计算机所有软、硬件资源的一组程序

- ✓计算机硬件与其它软件的接口
- ✓用户和计算机的接口

操作系统的主要功能:

- 处理机管理
- 存储管理
- 设备管理
- 信息(文件)管理



Linux的现状

- LINUX已能被移植到目前已知的各种公开的体系结构上，覆盖的领域小到穿戴设备，大到超级计算机集群。
- 若干计算机巨头公司都纷纷参与提供相关的解决方案。
- LINUX克隆了Unix但并不是Unix！
- LINUX借鉴了若干Unix的设计理念，实现了Unix的API，但并未直接使用Unix代码，这才是LINUX能提供GPL版权的重要原因。
- LINUX是非商业化的产品，这是它被广泛使用的原因。
- 目前LINUX上已经包纳了众多的软件。

Linux的特点——遵循POSIX标准

- POSIX 表示可移植操作系统接口
  (Portable Operating System Interface)
- POSIX是在Unix标准化过程中出现的产物。
- POSIX 1003.1标准定义了一个最小的Unix操作系统接口
- 任何操作系统只有符合这一标准，才有可能运行Unix程序

Linux的特点——GNU

- GNU 是 GNU Is Not Unix 的递归缩写，是自由软件基金会的一个项目 。
- GNU 项目产品包括 emacs 编辑器、著名的 GNU C 和 Gcc编译器等，这些软件叫做GNU软件。
- GNU 软件和派生工作均适用 GNU 通用公共许可证，即 GPL（General Public License ）
- LINUX的开发使用了众多的GUN工具
- GPL规定授予其他任何人可以合法复制、发行和修改GNU项目软件的权利。但是经过修改过的软件必须公开源代码，允许其他人免费下载。
- GPL 允许软件作者拥有软件版权

Linux的特点总结

1. 开放性
2. 多用户
3. 多任务
4. 良好的用户界面
5. 设备独立性
6. 丰富的网络功能
7. 可靠的系统安全
8. 良好的可移植性

Linux的结构

LINUX系统一般有4个主要部分：内核、Shell、文件系统和应用程序。

1. 内核

内核是操作系统的核心，具有很多最基本的功能，如虚拟内存、多任务、共享库、需求加载、可执行程序和TCP/IP网络功能。LINUX内核的主要模块分为存储管理、CPU和进程管理、文件系统、设备管理和驱动、网络通信、系统的初始化和系统调用等几个部分。

Linux的结构

2. Shell

Shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。实际上，Shell是一个命令解释器，它解释由用户输入的命令并且将它们送到内核。另外，Shell编程语言具有普通编程语言的很多特点，用这种编程语言编写的Shell程序与其他应用程序具有同样的效果。

Linux的结构

3. 文件系统

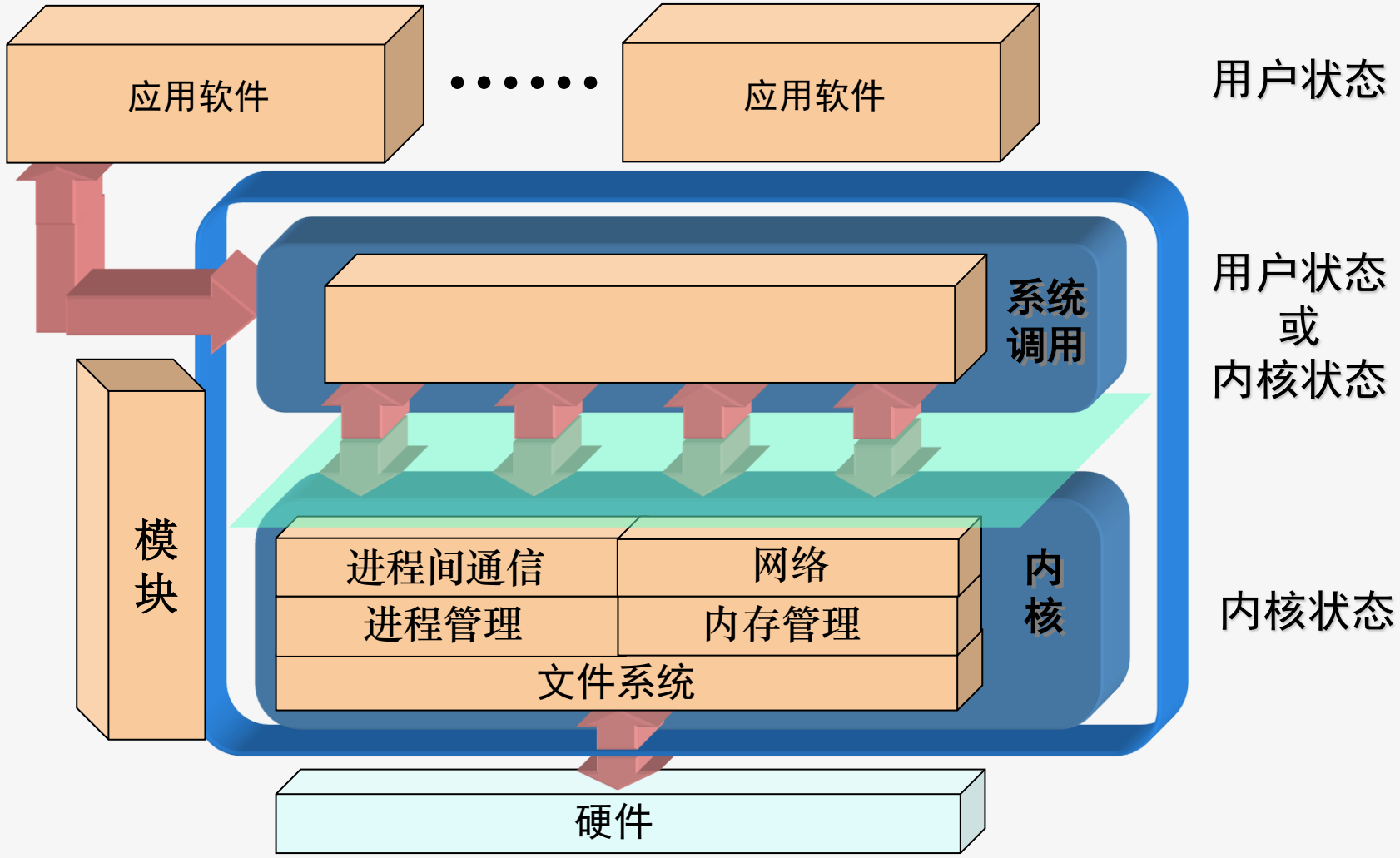
文件系统是文件存放在磁盘等存储设备上的组织方法。LINUX系统能支持多种目前流行的文件系统，如ext2，ext3，FAT，FAT32，VFAT和ISO9660等。

Linux的结构

4. 应用程序

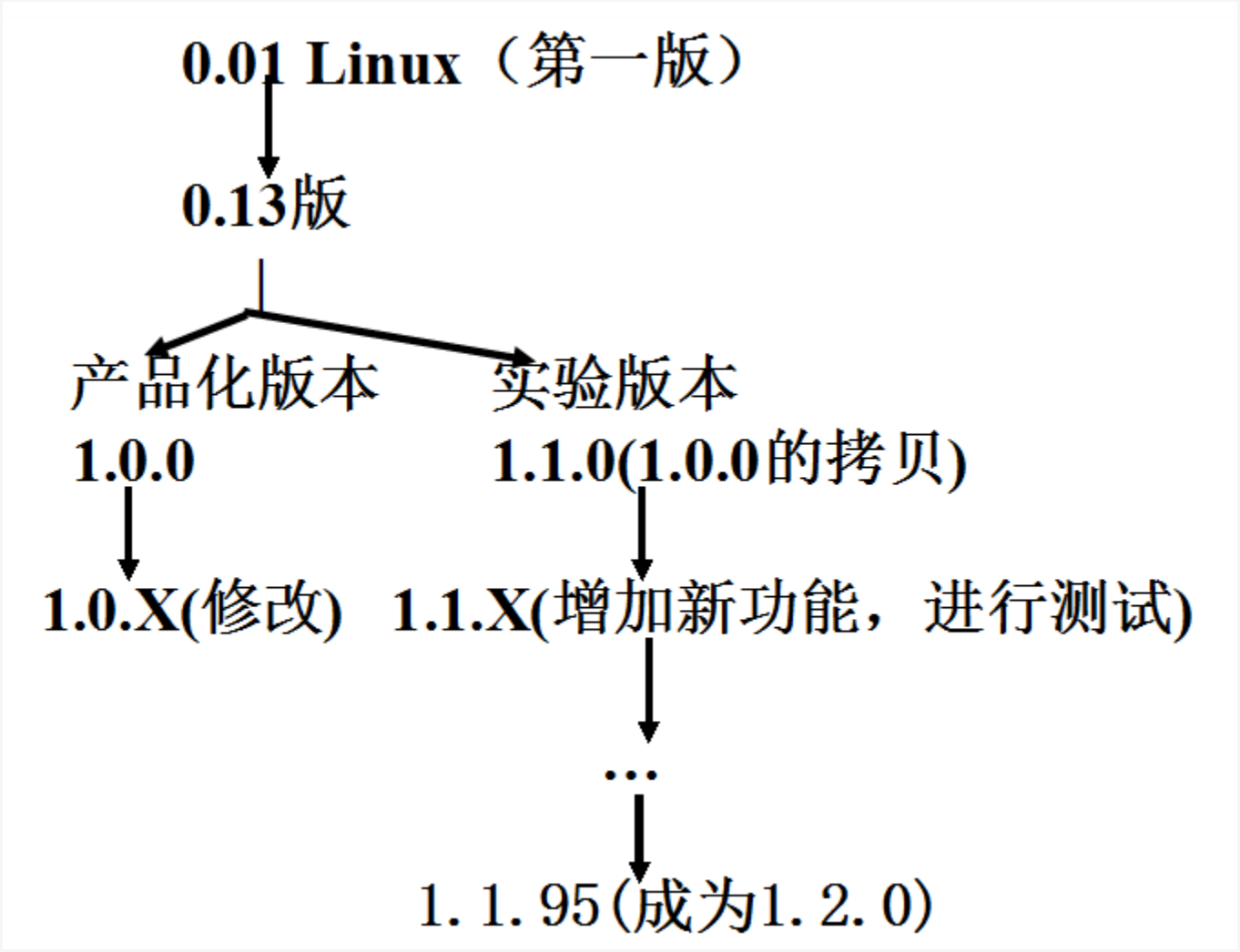
标准的LINUX系统都有一套称为应用程序的程序集，它包括文本编辑器、编程语言、X Window、办公软件、Internet工具和数据库等。

Linux的结构


应用软件
应用软件
用户状态
系统调用
用户状态
或
内核状态
模块
进程间通信
网络
进程管理
内存管理
文件系统
内核
内核状态
硬件

Linux的结构

LINUX内核版本树



Linux的结构

- 用户进程—运行在LINUX内核之上的一个庞大软件集合。
- 系统调用—内核的出口，用户程序通过它使用内核提供的功能。
- LINUX内核—操作系统的灵魂，负责管理磁盘上的文件、内存，负责启动并运行程序，负责从网络上接收和发送数据包等等。
- 硬件—包括了LINUX安装时需要的所有可能的物理设备。例如，CPU、 内存、硬盘、网络硬件等等。

Linux的结构

- 进程调度－控制着进程对CPU的访问。
- 内存管理－允许多个进程安全地共享主内存区域
- 虚拟文件系统－隐藏各种不同硬件的具体细节，为所有设备提供统一的接口。
- 网络－提供了对各种网络标准协议的存取和各种网络硬件的支持。
- 进程间通信(IPC)－ 支持进程间各种通信机制，包括共享内存、消息队列及管道等。

Linux目录结构

- /bin: bin就是二进制(binary)的英文缩写。在这里存放前面Linux常用操作命令的执行文件，如mv、ls、mkdir等。有时，这个目录的内容和/usr/bin里面的内容一样，它们都是放置一般用户使用的执行文件。

- /boot：这个目录下存放操作系统启动时所要用到的程序。如启动grub就会用到其下的/boot/grub子目录。

- /dev：该目录中包含了所有Linux系统中使用的外部设备。要注意的是，这里并不是存放的外部设备的驱动程序，它实际上是一个访问这些外部设备的端口。由于在Linux中，所有的设备都当作文件一样进行操作，比如：/dev/cdrom代表光驱，用户可以非常方便地像访问文件、目录一样对其进行访问。

Linux目录结构

- /etc：该目录下存放了系统管理时要用到的各种配置文件和子目录。如网络配置文件、文件系统、x系统配置文件、设备配置信息、设置用户信息等都在这个目录下。系统在启动过程中需要读取其参数进行相应的配置。
- /etc/rc.d：该目录主要存放Linux启动和关闭时要用到的脚本，在后面的章节中还会进一步地介绍。
- /etc/rc.d/init：该目录存放所有Linux服务默认的启动脚本(在新版本的Linux中还用到的是/etc/xinetd.d目录下的内容)。

Linux目录结构

- /home：该目录是Linux系统中默认的用户工具根目录。执行adduser命令后系统会在/home目录下为对应账号建立一个名为同名的主目录。
- /lib：该目录是用来存放系统动态链接共享库的。几乎所有的应用程序都会用到这个目录下的共享库。因此，千万不要轻易对这个目录进行什么操作。
- /lost+found：该目录在大多数情况下都是空的。只有当系统产生异常时，会将一些遗失的片段放在此目录下。
- /media：该目录下是光驱和软驱的挂载点。
- /misc：该目录下存放从DOS下进行安装的实用工具，一般为空。
- /mnt：该目录是软驱、光驱、硬盘的挂载点，也可以临时将别的文件系统挂载到此目录下。

Linux目录结构

- /proc：该目录是用于放置系统核心与执行程序所需的一些信息。而这些信息是在内存中由系统产生的，故不占用硬盘空间。
- /root：该目录是超级用户登录时的主目录。
- /sbin：该目录是用来存放系统管理员的常用的系统管理程序。
- /tmp：该目录用来存放不同程序执行时产生的临时文件。一般Linux安装软件的默认安装路径就是这里。
- /usr：这是一个非常重要的目录，用户的很多应用程序和文件都存放在这个目录下，类似与Windows下的Program Files的目录。

Linux目录结构

- /usr/bin：系统用户使用的应用程序。
- /usr/sbin：超级用户使用的比较高级的管理程序和系统守护程序。
- /usr/src：内核源代码默认的放置目录。
- /srv：该目录存放一些服务启动之后需要提取的数据。
- /sys：这是Linux 2.6内核的一个很大的变化。该目录下安装了2.6内核中新出现的一个文件系统sysfs。

Linux目录结构

- sysfs文件系统集成了下面3种文件系统的信息：针对进程信息的proc文件系统、针对设备的devfs文件系统以及针对伪终端的devpts文件系统。该文件系统是内核设备树的一个直观反映。当一个内核对象被创建的时候，对应的文件和目录也在内核对象子系统中被创建。
- /var：这也是一个非常重要的目录，很多服务的日志信息都存放在这里。

基本命令与使用--文件管理与传输

- 1.ls命令
- 2.cd命令
- 3.pwd命令
- 4. mkdir命令
- 5.rmdir命令
- 6.rm命令
- 7.cp命令
- 8.mv命令
- 9.find命令
- 10.ln命令
- 11.cat命令
- 12.chmod命令

基本命令与使用--磁盘管理与维护

- 1.fdisk命令
- 2.mount命令

12.chmod命令

例如:

```
444 -r--r--r-
600 -rw-------
644 -rw-r--r-
666 -rw-rw-rw-
700 -rwx------
744 -rwxr--r-
755 -rwxr-xr-x
777 -rwxrwxrwx
```

注:使用ll命令查看文件/文件夹属性时候,一共有10列,第一个小格表示是文件夹或者连接等等 d表示文件夹,l表示连接文件,-表示文件

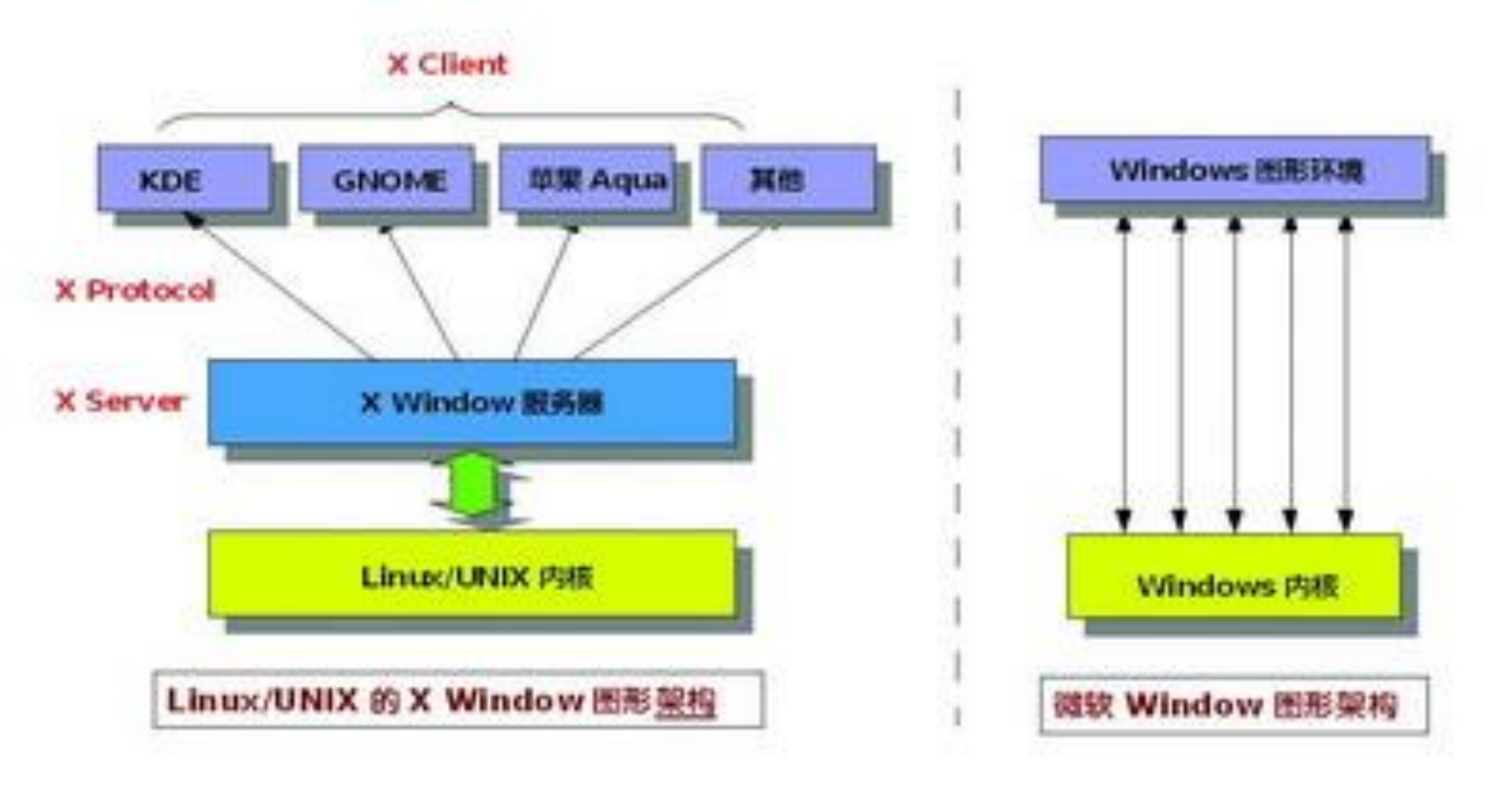
基本命令与使用--系统管理与设置

- 1.shutdown命令
- 2.ps命令
- 3.kill命令

基本命令与使用--网络相关

- 1.ifconfig命令
- 2.ping命令
- 3.netstat命令

Linux人机交互——图形界面

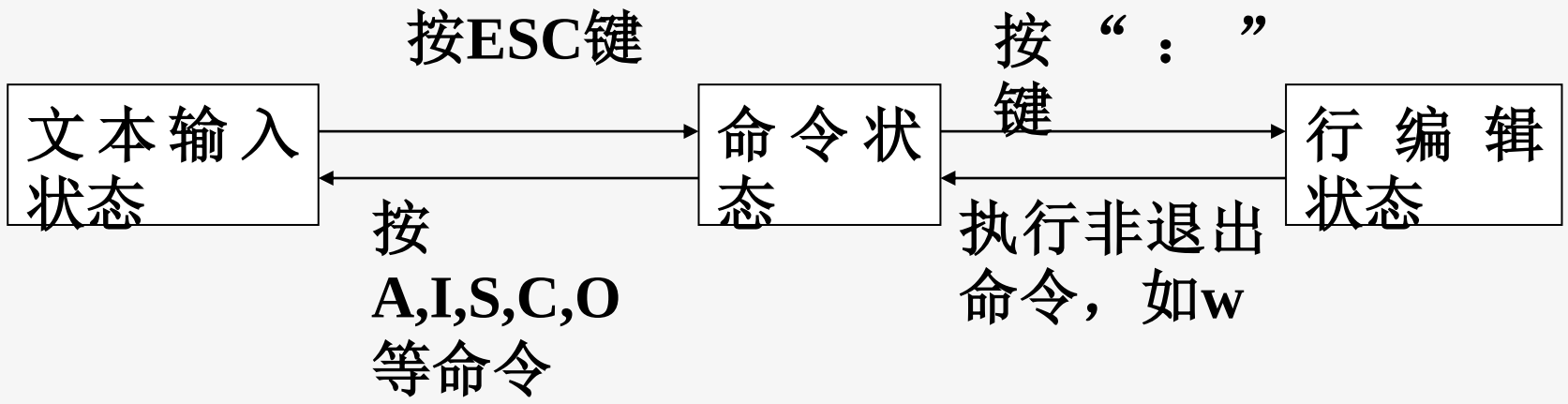


VI的三种工作模式

- 命令模式
  - 启动VI默认进入命令模式。此时界面不能编辑，只能接受命令(键入的命令看不到)
  - 文件的保存，退出，文本的删除、复制、搜索等操作
- 输入模式
  - 编辑模式
  - 命令模式下输入a（附加命令）、c（修改命令）、i（插入命令）、o（另起新行）、r（替换命令）以及s（替换命令）进入该模式。按esc返回命令模式
- 行编辑模式
  - 实际上也是命令模式的一种，在命令模式下输入冒号进入一个命令行，可显示地输入命令（所以也有些人认为是两种工作模式）。

三种模式可自由切换，一般切换命令就是操作的英文单词的首字母

三种模式间的转换关系



实验指导书中shell编程例子

命令结果重定向

- 1　　stdout标准输出
- 2　　stderr标准错误

- 输出重定向到文件file，终端上只能看到标准错误：
  #命令 >file
- 错误重定向到文件file，终端上只能看到标准输出：
  #命令 2>file
- 标准输出和标准错误都重定向到file，终端上看不到任何信息：
  #命令 >file 2>&1
  （等于#命令 1>file 2>&1）

屏蔽命令任何输出的：>/dev/null 2>&1

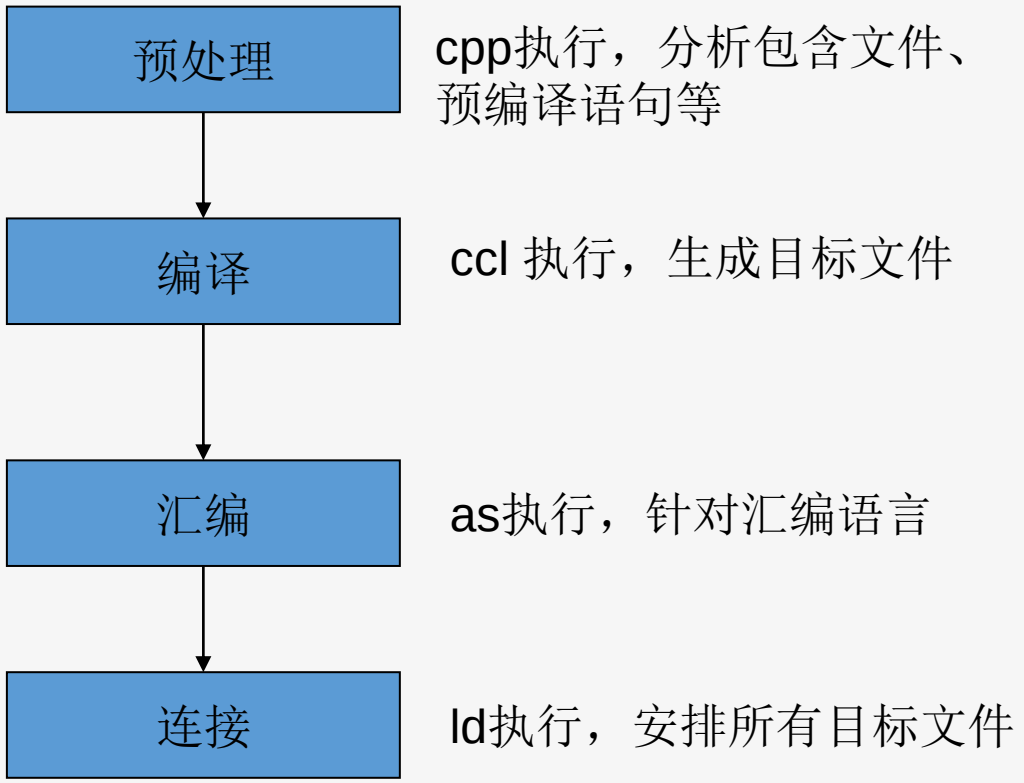
cp /etc/my.conf >/dev/null 2>&1

- /dev/null 空设备

- 此句命令的结果是：
  1. cp命令没有目标文件，应该输出错误。
  2. 2>&1表示错误重定向指向标准输出。
  3. >/dev/null又使标准输出重定向到空，就是不要输出信息
  - 即：一个错误的命令执行，什么功能都不实现，且没有任何信息或错误提示输出。

GCC的执行过程

- 虽然我们称gcc是C语言的编译器，但使用gcc由C语言源代码文件生成可执行文件的过程不仅仅是编译的过程，而是要经历四个相互关联的步骤：
  - 预处理(也称预编译，Preprocessing)
  - 编译(Compilation)
  - 汇编(Assembly)
  - 连接(Linking)。



设置输出的文件

- 在默认情况下，gcc编译出的程序为当前目录下的文件a.out。-o参数可以设置输出的目标文件。例如下面的命令，可以设置将代码编译成可执行程序HelloWorld。

```
gcc HelloWorld.c -o HelloWorld
```

- 也可以设置输出目录文件为不同的目录。例如下面的命令，是将目录文件设置成/tmp目录下的文件HelloWorld 。

```
gcc HelloWorld.c -o /tmp/HelloWorld
```

后缀名	所对应的语言
-c	只是编译不链接，生成目标文件“.o”gcc –c
-S	只是编译不汇编，生成汇编代码“.s”gcc –S
-E	只进行预编译，不做其他处理“.i”gcc –E –o
-g	在执行程序中包含标准调试信息
-o file	把输出文件输出到file里as a.s –o a.o
-v	打印编译器版本信息
-I dir	增加头文件的搜索范围
-L dir	增加库文件的搜索范围
-Wall	显示告警信息
-l	指定需要使用的库文件
-fpic/fPIC	生成位置无关的目标代码
-shared	产生共享库，在创建共享库时使用
-DM	相当于在程序中添加#define M 1(用于调试)

链接库

- 静态函数库
  - 这是最简单的函数库形式，静态函数库一般也叫做档案(archives)，以.a结尾，链接静态库的可执行文件中包含了库中所链接函数对应的二进制执行代码，因此在没有此函数库的情况下可以照常运行。
  - 比如/usr/lib/libc.a是标准的C函数库，而/usr/X11/libX11.a是X窗口函数库。

链接库

- 动态函数库
  - 静态函数库的缺点：当我们同时运行很多使用同一函数库中的函数的程序的时候，我们必须为每一个程序都复制一份一样的函数，这样占用了大量的内存和磁盘空间。
  - 动态函数库克服了这一缺点。如果某个函数使用动态函数库比如libc.so.N，那么此程序被链接到/usr/lib/libc.sa，这是一个特殊类型的函数库，它并不包含实际的函数，只是指向libc.so.N中的相应函数，并且只有在运行状态条用此函数时才将其调入内存。
  - 在Linux下可以用ldd命令查询某个程序使用了哪些动态库。
  - 注：在Windows系统中都是dll库。

链接库

•编译为静态链接库：

gcc -c add.c -o add.o

gcc -c sub.c -o sub.o

ar -crv libmylib.a add.o sub.o

ar的三个参数中：

- r代表将文件插入归档文件中，
- c代表建立归档文件，
- v 显示信息

使用系统默认库路径:

cp libmylib.a /lib/

gcc -o main main.c -static –lmylib

lib和.a都是系统指定的静态库文件的固定格式，mylib才是静态库的名称，编译时，链接器会在标准路径（/usr/lib;/lib）或者用户指定的路径下去找.a的文件。

gcc -o main main.c -static -L. –lmylib

//-L 在当前目录下查找库文件;

//-l 指定库名lib mylib.a;

链接库

•编译为动态链接库：

gcc -c *.c //支持统配符;

gcc -shared -fpic -o libmym.so *.o

//生成一个动态库

libmym.so; (与位置无关代码)

gcc main.c -lmym -L .

//链接动态库编译main.c的程序;

解决执行时找不到动态库的方法:

方法一:

cp libmym.so /lib

方法二：

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./

方法三:

vim /etc/ld.so.conf

>>/class/2c/src/5th

//追加这段内容;

ldconfig

//让配置文件生效;

一个带变量的Makefile的例子

- OBJS=prog.o code.o
- CC=gcc
- test： ${ OBJS }
- ${ CC } –o test ${ OBJS }
- prog.o： prog.c prog.h code.h
- ${ CC } –c prog.c –o prog.o
- code.o： code.c code.h
- ${ CC } –c code.c –o code.o
- clean：
- rm –f *.o

西安理工大学
XI'AN UNIVERSITY OF TECHNOLOGY

实验指导书中编程实验

Linux进程概念——进程示例

```
#include <sys/types.h>  /* 提供类型pid_t的定义,在PC机上与int型
   相同 */
#include  <unistd.h>    /* 提供系统调用的定义
main()
{      pid_t pid;       /*此时仅有一个进程*
        printf("PID before fork():%d\n",(
      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 p
   %d\n",getpid());
      else
            printf("I am the parent process, my process ID is
   %d\n",getpid());
}
```

为了区分父/子进程，fork()给两个进程返回不同的值。对父进程，fork()返回新创建子进程的进程标识符（PID），而对子进程，fork()返回值0，错误返回-1

进程描述符

★对进程进行全面描述的数据结构

Linux中把对进程的描述结构叫做task_struct:

```
struct task_struct {
  …
  …
        }
```

★传统上，这样的数据结构被叫做**进程控制块**PCB (process control block)

进程描述符

★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、计费等信息。

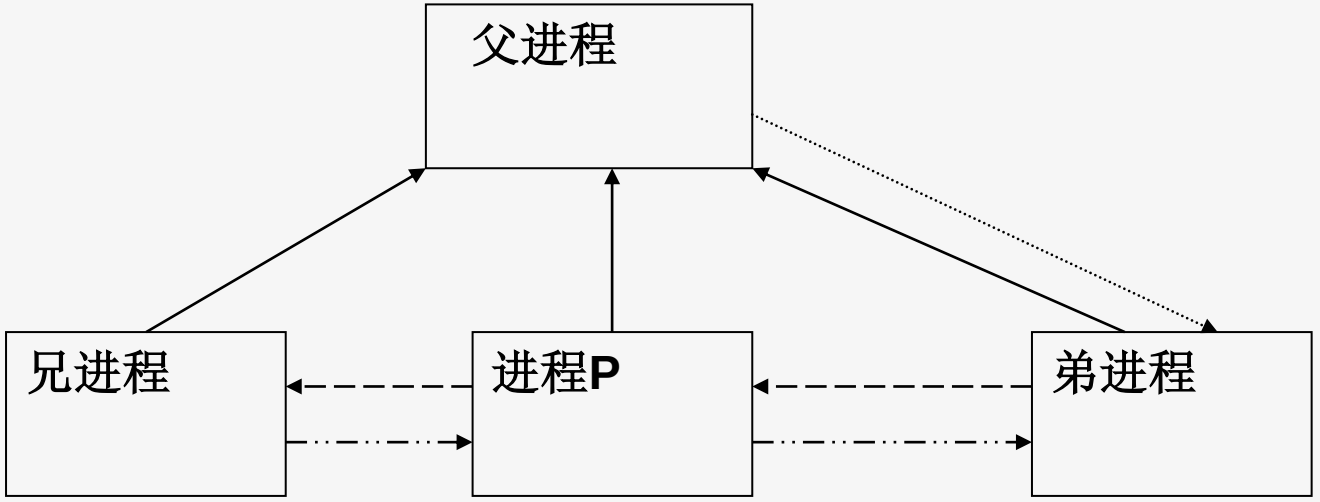
★调度信息－描述进程优先级、调度策略等信息。

★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)

进程描述符——进程间的族亲关系



指向父进程　　指向子进程

指向兄进程　　指向弟进程

进程描述符——进程间的族亲关系

- struct task_struct *p_opptr //指向祖先进程PCB的指针
- struct task_struct *p_pptr //指向父进程PCB的指针
- struct task_struct *p_cptr //指向子进程PCB的指针
- struct task_struct *p_ysptr //指向弟进程PCB的指针
- struct task_struct *p_osptr //指向兄进程PCB的指针

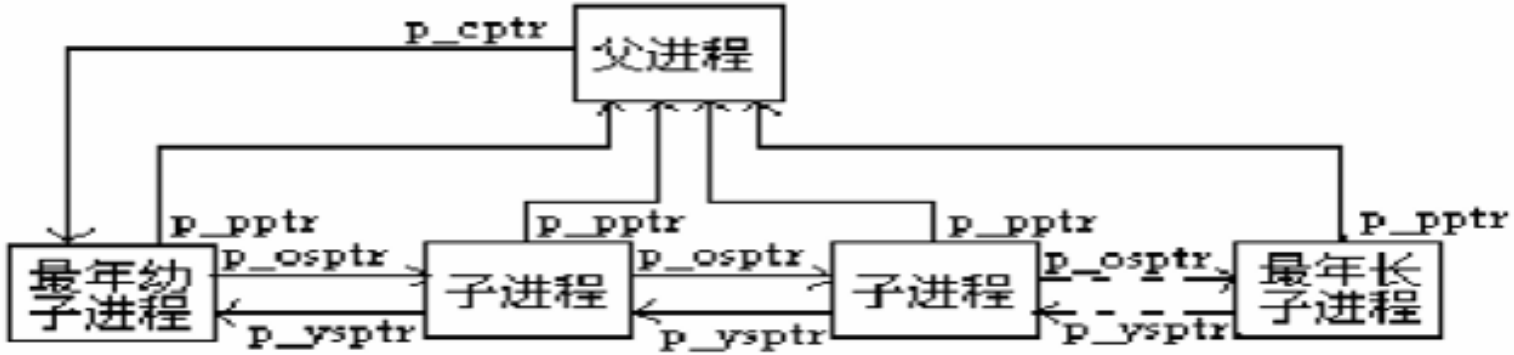


图　进程的族亲关系

相关系统调用

- **fork()** 通过复制调用进程来建立新的进程，是最基本的进程建立过程
- **exec()** 包括一系列系统调用，它们都是通过用一个新的程序覆盖原来的内存空间，实现进程的转变
- **wait()** 提供初级的进程同步措施，能使一个进程等待，直到另外一个进程结束为止。
- **exit()** 该系统调用用来终止一个进程的运行

进程的创建—fork()

★进程的创建fork()借用现实世界的“克隆”技术

★ 子进程克隆父进程，但不仅如此。而是采用了“写时复制 ”技术

★父进程并不是把自己的所有东西马上都给儿子，而是直到儿子真正需要时才给它。

★也就是当父进程或子进程试图修改某些内容时，内核才在修改之前将被修改的部分进行拷贝－这叫做写时复制。

★fork()的实际开销就是复制父进程的页表以及给子进程创建唯一的PCB

fork()功能小结

1. 为新进程分配task_struct内存空间;
2. 把父进程task_struct拷贝到子进程的task_struct;
3. 为新进程在其虚拟内存建立内核堆栈;
4. 对子进程task_struct中部分进行初始化设置
5. 把父进程的有关信息拷贝给子进程，建立共享关系;
6. 把子进程的counter设为父进程counter值的一半;
7. 把子进程加入到可运行队列中;
8. 结束do_fork()函数返回PID值.

进程的执行

- 在系统中创建一个进程的目的是需要该进程完成一定的任务，需要该进程执行它的程序代码。
- 在Linux系统中，使程序执行的唯一方法是使用系统调用exec()。
- 系统调用exec()有多种使用形式，称为exec()族，它们只是在参数上不同，而功能是相同的。如：

```
int execl(const char * path, const  char *arg0,...,
    const  char *argn,(char*)0)
```

  - path: 要执行的程序文件的完整路径名
  - arg0: 要执行程序的文件名或命令名
  - arg1,...,argn: 执行程序所需的参数，

进程的执行

- exec有一系列的系统调用：

```
#include <unistd.h>
extern char **environ;
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const  char*path,        const  char  *arg  , ..., char *
    const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int  execve(const  char      *filename,  char  *const  argv [], char
    *const envp[]);
```

  - 前面几个函数都是通过调用execve来实现的。请看教材中execve实现的分析。

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    pid_t pid;
    printf("Begin\n");
    pid=fork();
    if(pid==0)
    {
        execl("/bin/echo","echo","hello",0);
        printf("I am the child process\n");
    }
    else if(pid>0)
        printf("I am the parent process\n");
    return 0;
}
```

进程管理小结

★Fork()—父亲克隆一个儿子。执行fork()之后，兵分两路，两个进程并发执行。

★Exec()—新进程脱胎换骨，离家独立，开始了独立工作的职业生涯。

★Wait()—等待不仅仅是阻塞自己，还准备对僵死的子进程进行善后处理。

★Exit()—终止进程，把进程的状态置为“僵死”，并把其所有的子进程都托付给init进程，最后调用schedule()函数，选择一个新的进程运行。

调度策略—I/O和处理器消耗型

- I/O消耗型进程指进程的大部分时间用于提交I/O请求或是等待I/O请求。
- 处理器消耗型进程主要时间用于执行代码上。

•调度策略通常要在进程响应时间和最大系统利用率之间取得平衡。为此，调度程序通常采用一套非常复杂的算法以决定下一个运行的进程。Linux为了保证交互式应用更倾向于优先调度I/O消耗型进程以缩短系统响应时间。

调度策略——调度算法考虑的因素

★公平：保证每个进程得到合理的CPU时间。

★高效：使CPU保持忙碌状态，即总是有进程在CPU上运行。

★响应时间：使交互用户的响应时间尽可能短。

★周转时间：使批处理用户等待输出的时间尽可能短。

★吞吐量：使单位时间内处理的进程数量尽可能多。

调度策略——调度算法

★时间片轮转调度算法

- ❖系统使每个进程依次地按时间片轮流地执行

★优先权调度算法

- ❖非抢占式优先权算法
- ❖抢占式优先权调度算法

★多级反馈队列调度

- ❖优先权高的进程先运行给定的时间片，相同优先权的进程轮流运行给定的时间片

★实时调度

- ❖一般采用抢占式调度方式

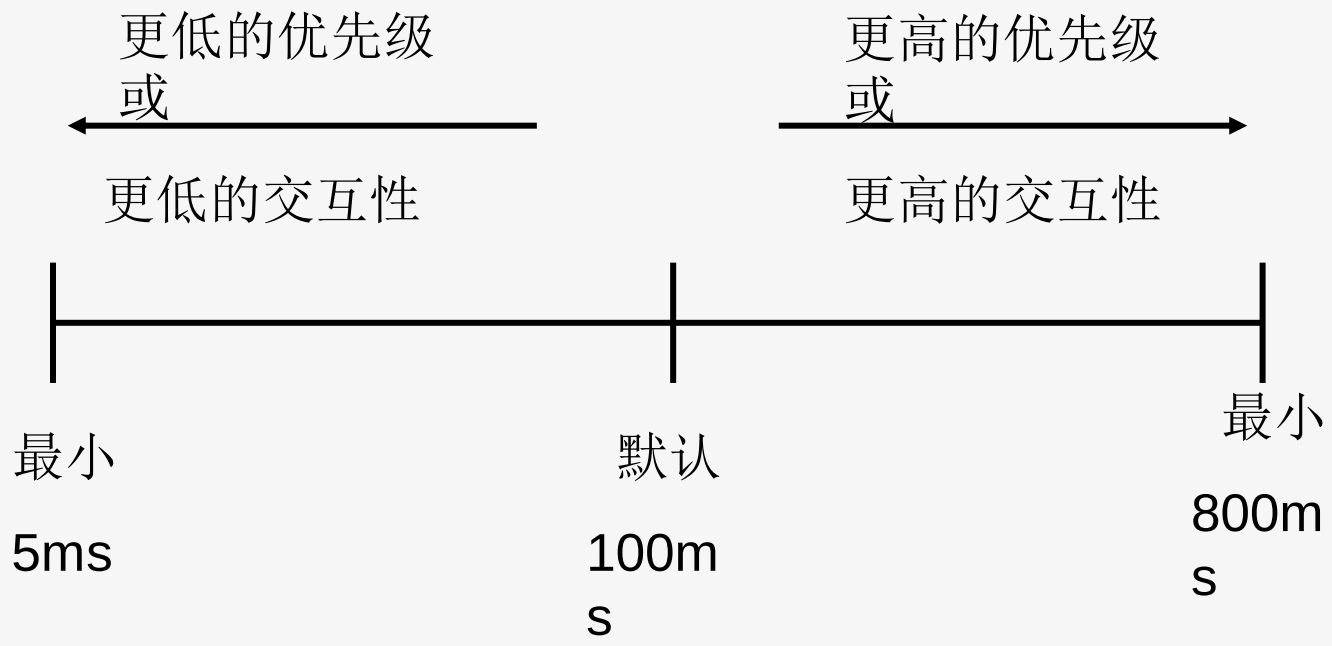
进程的优先级

Linux内核提供两组独立的优先级:

◆第一种是nice值，默认值0，范围为-20到19，值越大优先级越低，nice值小的进程优先执行。Nice值也用来决定分配给进程的时间片长短，值越小时间片越长，nice是所有Unix都用到的标准优先级范围。

◆第二种是实时优先级，默认的变化范围是0到99，任何实时进程的优先级都高于普通进程，Linux提供对POSIX实时优先级的支持。

调度策略——时间片



进程也可以分几次用完其所拥有的时间片。当进程的时间片耗尽时，就认为进程到期了。需等到所有进程都耗尽了它们的时间片，所有进程的时间片会被重新计算。

调度策略——进程抢占

当一个进程进入TASK_RUNNING状态，内核会检查其优先级是否高于当前正在执行的进程，若高，唤醒调度程序，抢占当前正在执行的进程，并运行新的可执行进程。

此外，当某个进程的时间片为0时，它也会被抢占。

调度策略——调度例子

一个文字编辑程序和一个视频编码程序，前者是I/O消耗型，后者是处理器消耗型。

此种情况下，文字编辑程序可被分配更高的优先级和更长的时间片，这样，既保证文字编辑程序有充足的时间片可用，还可使其在需要的时候实现抢占。同时提高了两种应用的性能。

调度相关的系统调用

与调度策略和优先级相关的系统调用

- ◇ sched_setscheduler()和 sched_getscheduler()：设置和获取进程的调度策略和实时优先级
- ◇ sched_setparam() 和sched_getparam()：设置和获取进程的实时优先级

与处理器绑定有关的系统调用

- ◇ sched_setaffinity ()和 sched_getaffinity () ：设置和获取进程可运行的处理器掩码（若允许在某处理器上运行，则置位该掩码）
- ◇掩码标志： task_struct中的cpus_allowed

放弃处理器时间

- ◇ sched_yieald ():将进程从活动队列转移到过期队列
- ◇ 若在内核空间，则直接调用yield()即可

  若在用户空间，则用sched_yieald ()系统调用

与调度相关的域

- ★need_resched：调度标志，以决定是否调用schedule( )函数。
- ★ counter：进程处于可运行状态时所剩余的时钟节拍（即时钟中断的间隔时间，为10ms或1ms）数。这个域也叫动态优先级。
- ★priority：进程的基本优先级或叫“静态优先级”
- ★rt_priority：实时进程的优先级
- ★ policy：调度的类型，允许的取值是：
  - ❖SCHED_FIFO：先入先出的实时进程
  - ❖SCHED_RR：时间片轮转的实时进程
  - ❖SCHED_OTHER：普通的分时进程。

中断处理的两步机制

1、上半部（top half）：即中断处理程序，接收到一个中断后立即执行。有严格时限，且所有中断被禁止

2、下半部（bottom half）：可以稍后再完成的工作，在适当的时候，开始执行。

如网卡：接收到数据报，为避免超时，迅速拷贝数据到内存；稍后再对其进行从容处理后交给合适的协议栈或应用程序。

中断相关数据结构

- 中断描述符表即中断向量表相当于一个数组，包含256个中断描述符，每个中断描述符8位，对应硬件提供的256个中断服务例程的入口，即256个中断向量。
- IDT的位置由idtr确定，idtr是个48位的寄存器，高32位是IDT的基址，低16位为IDT的界限(通常为2k=256*8)。