欢迎大家来到第二阶段课程

UC Day01

预习课

预习内容

- 什么是UC
- 操作系统的历史
- Linux的发行版本

什么是UC

什么是UC

知识讲解

- U是指unix操作系统。Unix操作系统是使用C语言实现的系统级软件。
- C是指C语言，就是大家上门课程学习的语言。
- UC是指使用C语言在Unix操作系统上的用户编程。unix系统向用户提供了大量的接口。用户通过系统提供的接口，使用操作系统提供的服务。
- 想要写出功能强大的程序，一定要借助操作系统提供的功能。
  - 网络通信、线程管理、文件系统等。

操作系统的历史

操作系统的历史

- 1961 - 1969：史前时代
  - CTSS(Compatible Time-Sharing System，兼容分时系统)，以MIT为首的开发小组，小而简单的实验室原型。
  - Multics(Multiplexed Information and Computing System，多路信息与计算系统)，庞大而复杂，不堪重负。
  - Unics(Uniplexed information and Computing System，单路信息与计算系统)，返璞归真，走上正道。

知识讲解

操作系统的历史

- 1969 - 1971：创世纪
  - 1969年，AT&T退出了Multics项目，来自贝尔实验室的Ken Thompson为了在PDP-7上实现他的星际旅行游戏，编写了一系列实用程序，成为后来Unix系统的核心。
  - 1970年，Ken Thompson在BCPL语言的基础上发明了B语言，当时的Unix系统用汇编语言和B语言混合编写。
  - 1971年，第一个Unix应用程序nroff诞生，为贝尔实验室的文字处理工作提供支持，Unix系统开始向PDP-11移植。

操作系统的历史

- 1971 - 1979：出谷记
  - 1971年，Dennis Ritchie在B语言的基础上发明了C语言。
  - 1973年，Dennis Ritchie用C语言重写了整个Unix系统，极大地提升了Unix系统的可读性、可维护性和可移植性。
  - 1974年，Ritchie和Thompson在《美国计算机通信》上发表论文，第一次公开展示Unix。
  - 1979年，贝尔实验室发布Unix V7版本，成为Unix世界公认的第一个真正意义上的Unix操作系统。

操作系统的历史

- 1980 - 1985：第一次Unix战争
  - 1980年，美国国防部高级研究计划局(DARPA)决定在加州大学伯克利分校开发的BSD Unix上实现TCP/IP协议栈。
  - 1983年，AT&T的拆分使其得以将Unix System V商业化。
  - 1984年，第一次Unix战争在AT&T的Unix System V和伯克利的BSD Unix之间全面爆发。
  - 1985年，以IEEE POSIX为核心的一系列技术标准最终弥合了Unix System V和BSD Unix之间的裂痕。

操作系统的历史

- 1988 - 1990：第二次Unix战争
  - 1988年，AT&T持有Sun公司20%的股份，宣告两家公司正式联姻。
  - 1989年，IBM、DEC、HP等二线厂商创建开放软件基金会(OSF)并结成盟友，以与AT&T/Sun轴心对抗，第二次Unix战争爆发。
  - 1990年，Windows 3.0发布，Unix世界从酣战中幡然醒悟，崭新的帝国正在崛起，真正的敌人来自Redmond。

知识讲解

操作系统的历史

知识讲解

- 1991 - 2000：尘埃落定
  - 1991年，芬兰大学生Linus Torvalds宣布了Linux项目。
  - 1993年，Linux已达到产品级操作系统的水准。
  - 1993年，AT&T将Unix系统实验室卖给了Novell。
  - 1994年，Novell将Unix商标卖给了X/Open标准组。
  - 1995年，Novell将UnixWare卖给了SCO。
  - 2000年，SCO将Unix源码卖给了Caldera—Linux发行商。

操作系统的历史

知识讲解

- Linux介绍
  - Linux是一款类Unix操作系统，免费开源，Linux的不同发行版本都使用相同的内核。
  - Linux可以运行在手机、平板、路由器、视频游戏控制器、个人电脑、大型计算机、超级计算机等多种硬件平台上
  - 严格意义上的Linux仅指操作系统内核，但一般被用于指称某个具体的发行版本

操作系统的历史

知识讲解

- Linux介绍
  - Linux隶属于GNU工程，整套GNU工具包从一开始就内置其中，可提供高质量的开发工具。
  - Linux的发明人是Linus Torvalds，同时他也是Linux商标的合法持有者
  - 严格意义上的Linux仅指操作系统内核，但一般被用于指称某个具体的发行版本

操作系统的历史

- Linux系统的特点
  - 遵循GNU/GPL。
  - 开放性。
  - 多用户。
  - 多任务。
  - 设备独立性。
  - 丰富的网络功能。
  - 可靠的系统安全。
  - 良好的可移植性

Linux的发行版本

操作系统的历史

直播课见

UC

C/C++教学体系

目录

计算机系统分层

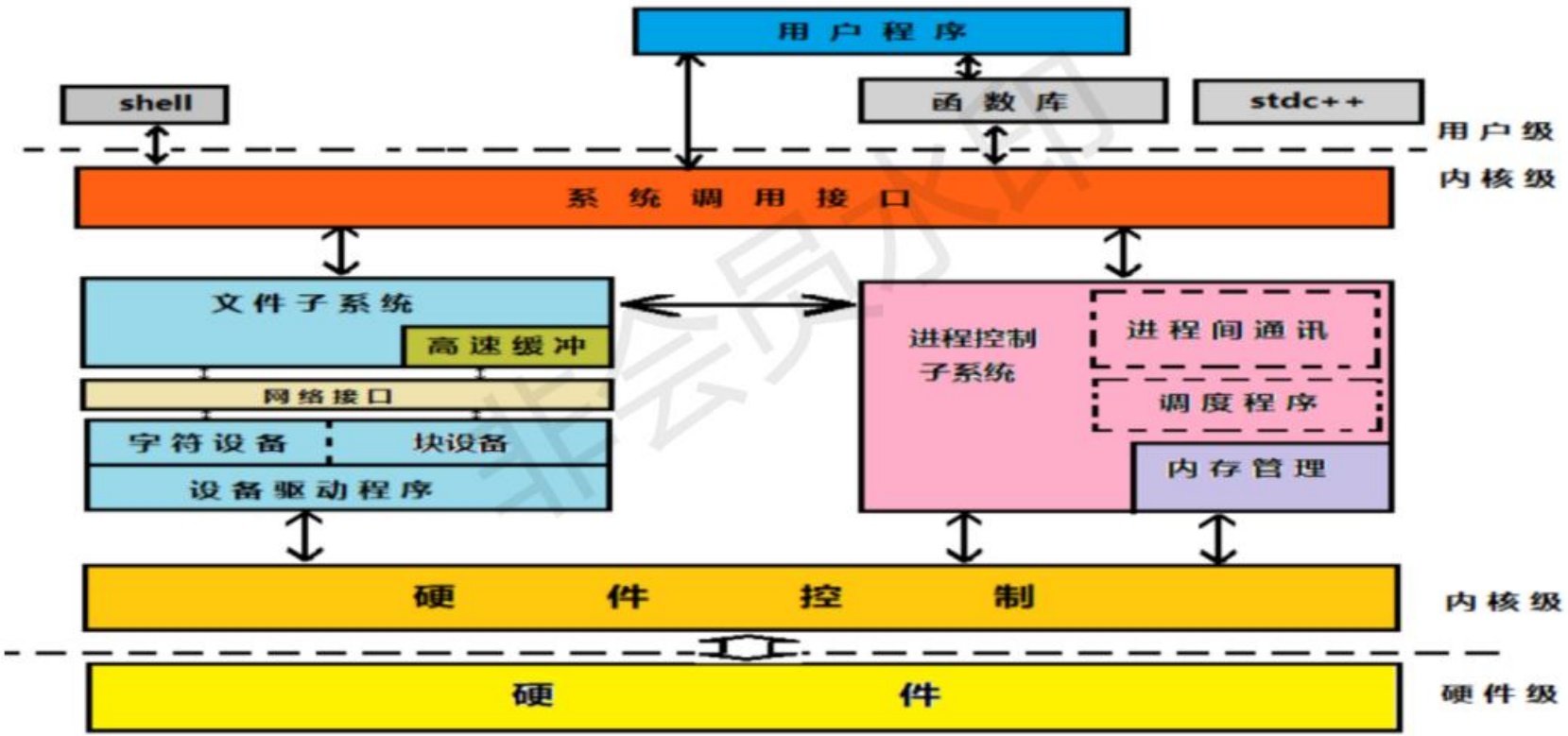
计算机系统分层

知识讲解

- 什么是操作系统
  - 操作系统是管理计算机硬件资源和软件资源的一款系统软件。
  - 操作系统简称OS。

计算机系统分层

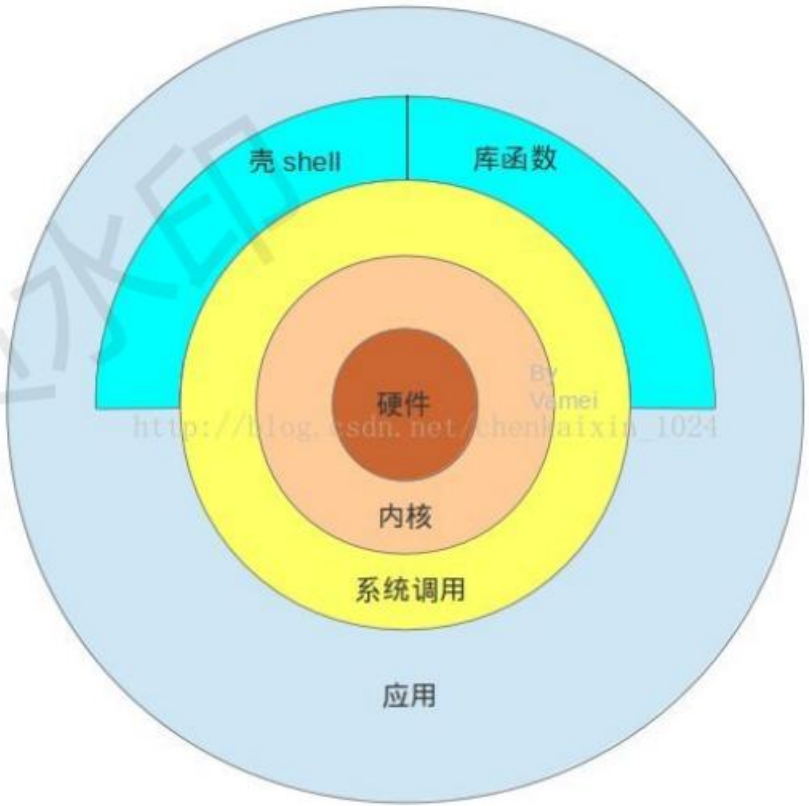
知识讲解



计算机系统分层

知识讲解

- 操作系统通过驱动程序管理着计算机的硬件资源
- 通过系统调用和用户进行交互
- 在很多书籍中，对操作系统层次的描述也有类似右图



环境变量

环境变量

- 什么是环境变量
  - bash用一个叫做环境变量的特性来存储有关工作环境的信息。
  - 进程可以通过环境变量访问计算机的资源。
  - 在终端下输入env命令，可以查看环境变量列表
  - 通过echo $name 可以查看某个环境变量的值

知识讲解

环境变量

- 环境变量的添加
  - 在终端窗口中输入 键=值 形式的内容，回车。
  - 比如 FOOD=guobaorou，表示在当前bash中，添加名为 FOOD，值为 guobaorou的环境变量。
  - 如果环境变量FOOD存在，则更改其值。
  - 强调，在添加环境变量时，登号左右两侧不要添加空格。

知识讲解

环境变量

- 常见环境变量

  – PATH环境变量

  PAHT=/home/tarena/Qt5.4.1/5.4/gcc_64/bin:/home/tarena/Qt5.4.1/Tools/QtCreator/bin:/home/tarena/bin:/home/tarena/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

  – 该环境变量所记录的是bash进程对命令的检索路径

  格式为“:”分割的多个路径。当在bash下输入命令的时候，首先，在第一个路 径下找该命令的可执行程序，找到就执行，不再向后找；如果找不到，在第二个路径下找，找到就执行，不再向后找；如果找不到，继续下一个路径。如果到最后一个路径都找不到，就提示该命令不能找到的错误

环境变量

- 常见环境变量
  - 如果想要执行自己的程序，而又不想添加”./“，该如何做呢?
    ./a.out --> a.out
  - 在PATH环境变量中，添加当前路径，再执行程序时，即可省略”./“。
    PATH=&PATH:.

知识讲解

环境变量

- 常见环境变量
  - 如果没有特殊操作，对环境变量的设置仅对当前shell进程有效，开启新的终端，之前的操作不会被保留。
  - 在家目录下有名为.bashrc的脚本文件，每次bash进程启动前，都会执行该脚本文件的内容。如果希望环境变量的设置对每个bash进程都有效，可以将环境变量的设置写在该脚本文件中。
  - 执行 source ~/.bashrc 命令，可以使文件立即对当前bash生效。

知识讲解

环境变量表

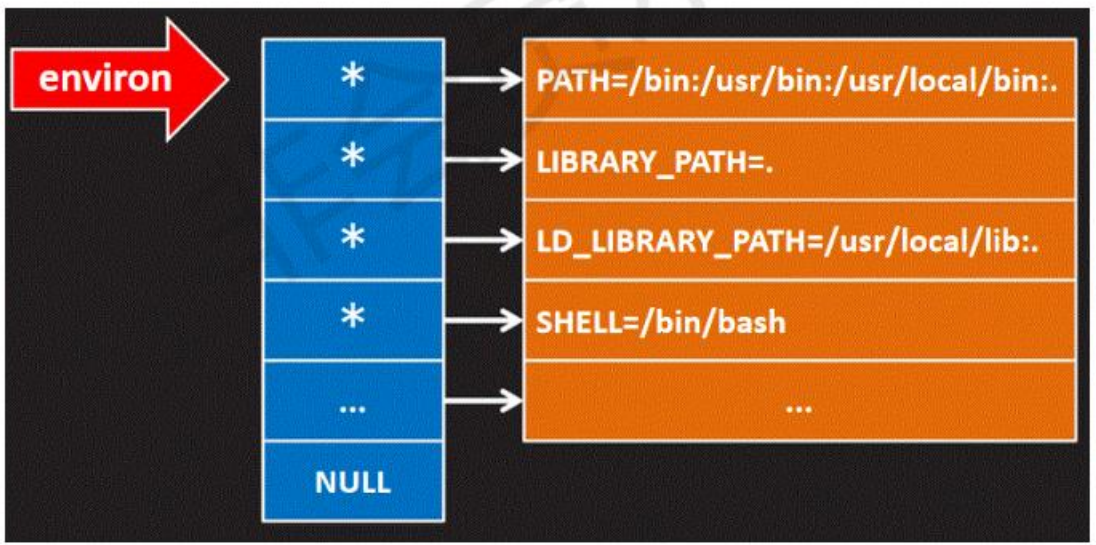
环境变量表

知识讲解

- 每个进程都有一张独立的环境变量表，其中的每个条目都是一个形如“键=值”形式的环境变量

环境变量表

- 所谓环境变量表就是一个以NULL指针结束的字符指针数组，其中的每个元素都是一个字符指针，指向一个以空字符结尾的字符串，该字符串就是形如“键=值”形式的环境变量。该指针数组的地址保存在全局变量environ中



知识讲解

环境变量表

- 通过全局环境变量表指针environ可以访问所有环境变量

```
extern char** environ;
char** pp;
for (pp = environ; *pp; ++pp){
        printf ("%s\n", *pp);
}
```


知识讲解

环境变量表

- 通过main函数的第三个参数也可以访问到进程的环境变量，main函数的第三个参数就是环境变量表的起始地址

知识讲解

```
int main (int argc, char* argv[], char* envp[]) {
        char** pp;
        for (pp = envp; *pp; ++pp){
                printf ("%s\n", *pp);
        }
        return 0;
}
```

错误处理

错误处理

知识讲解

- 针对因为运行环境、人为操作等原因会导致程序执行时发生错误，那么如何获取具体的错误原因呢?

- 我们一般会采取下列几种方式

错误处理

知识讲解

- 通过错误号了解具体的错误原因
  - 系统于定义的整数类型全局变量errno中存储了最近一次系统调用的错误编号
  - 知道了错误编号，也就知道了错误原因
  - 头文件/usr/include/errno.h中包含了对errno全局变量的外部声明
  - 在头文件/usr/include/asm-generic/errno-bashe.h中包含各种错误号的宏定义

错误处理

知识讲解

- 通过strerror()函数了解错误原因
  - #include<string.h>
  - char* strerror(int errnum)
  - 功能：将整数形式的错误号转换为有意义的字符串
  - 参数：errnum　错误号
  - 返回值：返回与参数错误号对应的描述字符串

错误处理

知识讲解

- 通过perror()函数了解错误原因
  - #include <stdio.h>
  - void perror(char const* tag)
  - 功能：在标准出错设备上打印最近一次函数调用的错误信息
  - 参数：tag　为用户自己制定的提示内容，输出时，会自动在该提示内容和错误信息之间添加冒号进行分隔

谢谢

UC Day01

复习课

关于常用环境变量的扩展

关于常用环境变量的扩展

知识讲解

- 环境变量 PS1
- 该环境变量决定了窗口命令行的提示符内容

```
echo $PS1
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:
\[\033[01;34m\]\w\[\033[00m\]\$ '
```

- 可以通过对PS1变量的设置来精简提示符内容

```
PS1='\W\$'
```

只显示最后一级目录

关于错误处理需注意问题

关于错误处理需注意问题

知识讲解

- 虽然所有的错误编号都不是零，但是因为在函数执行成功的情况下存储错误编号的全局变量errno并不被清零，所以不能用该变量的值是否为零作为最近一次函数调用是否成功的判断条件。正确的做法是，先根据函数的返回值判断其是否出错，在确定出错的前提下，再根据errno的值判断具体出了什么错

下节课见