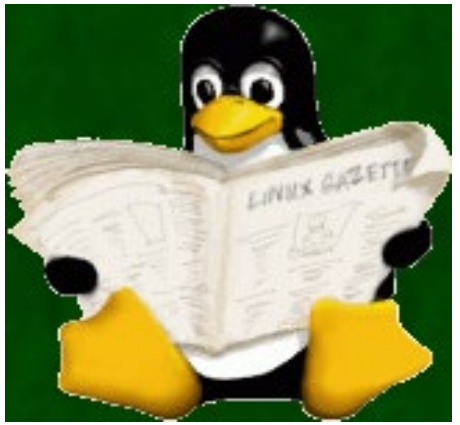
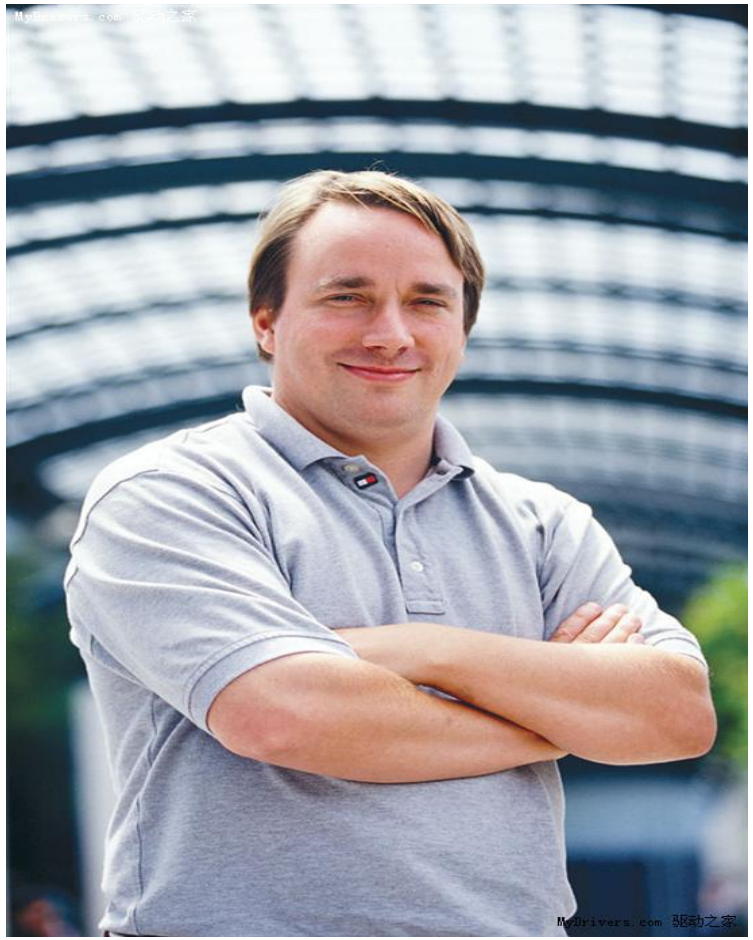
Linux概述



Linux之父Linus Torvalds



- 林纳斯·本纳第克特·托瓦兹（1969年12月28日—），生于芬兰赫尔辛基市，拥有美国国籍。他发起了Linux内核的开源项目，并以此广为人知，是当今世界最著名的电脑程序员、黑客之一。他还发起了Git这个开源项目，并为主要的开发者。
- 他毕业于赫尔辛基大学计算机科学系，现任职于Linux基金会（http://www.linuxfoundation.org）。

http://zh.wikipedia.org/wiki/Linus_Torvalds

什么是Linux?

工业界是这样认为的

Linux指的是Linux内核

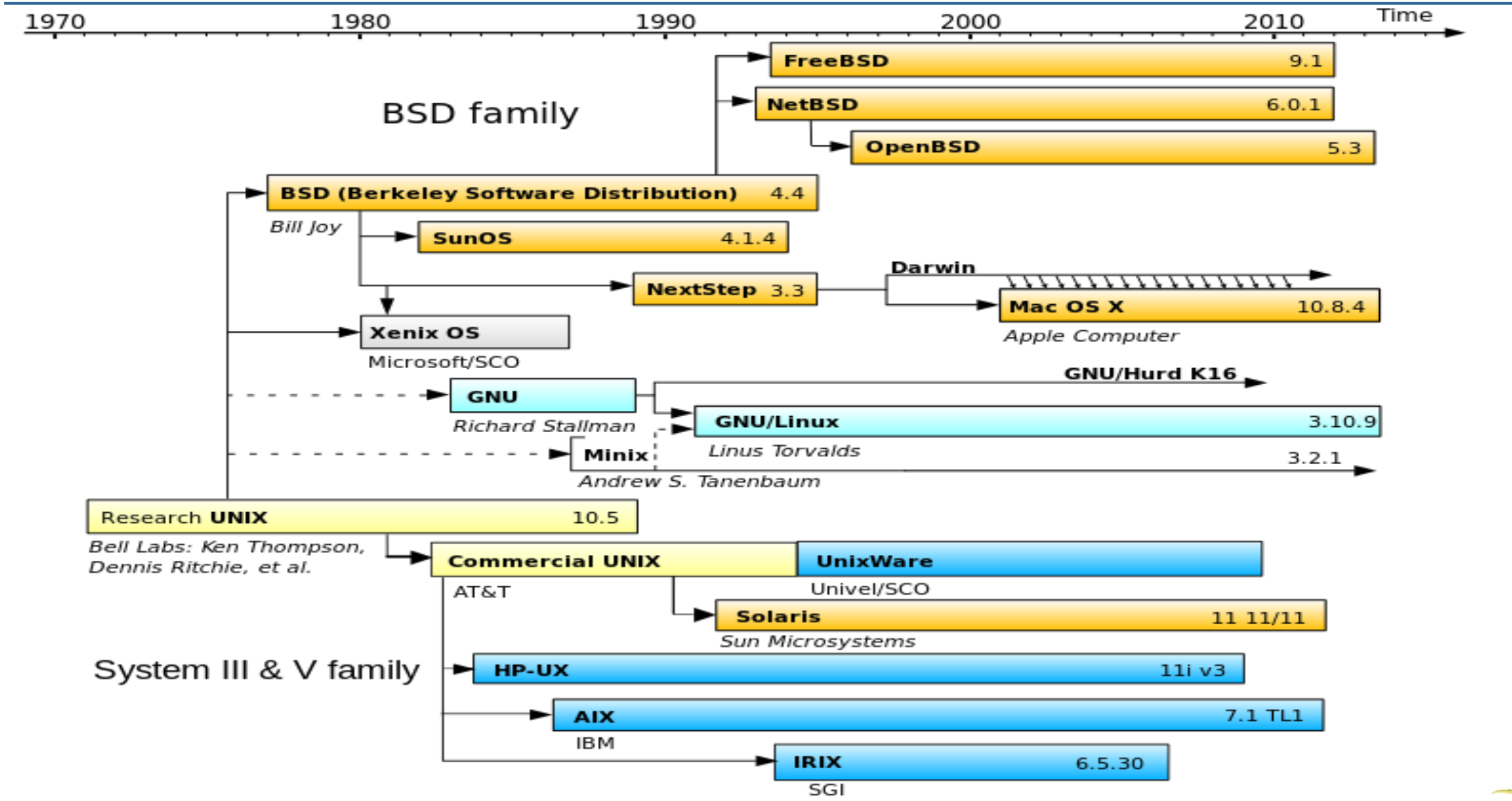
Linux操作系统指的是GNU/Linux 系统（基于 Linux 的 GNU 系统）

- Linux系统的组成：内核、c库、编译器、工具集和系统的基本工具、各种硬件设备驱动程序、X Windows系统、登录程序和shell、各种应用软件包括字处理软件、图象处理软件等；
- Linux系统（发行版）：GNU软件28% +linux内核3%+其他部件。——www.gnu.org

- Linux是一种类UNIX的操作系统，Linux克隆了Unix，但不是Unix。
- Linux是遵守GNU的GPL/LGPL/AGPL协议的软件。
- 本课程使用Linux这个词，多数时候是指Linux内核。

Fedora core 9 代码204,500,946行，Linux kernel 2.6.27代码10,000,000行

UNIX大家庭


1970
1980
1990
2000
2010
Time
BSD family
FreeBSD 9.1
NetBSD 6.0.1
OpenBSD 5.3
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
NextStep 3.3
Darwin
Mac OS X 10.8.4
Apple Computer
Xenix OS
Microsoft/SCO
GNU/Hurd K16
GNU
Richard Stallman
GNU/Linux 3.10.9
Linus Torvalds
Minix 3.2.1
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson, Dennis Ritchie, et al.
Commercial UNIX
UnixWare
Univel/SCO
AT&T
Solaris 11 11/11
Sun Microsystems
System III & V family
HP-UX 11i v3
AIX 7.1 TL1
IBM
IRIX 6.5.30
SGI

2. GNU与Linux

www.gnu.org

- GNU is an operating system that is free software–that is, it respects users' freedom. The development of GNU made it possible to use a computer without software that would trample your freedom.
- The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system.
- GNU's kernel wasn't finished, so GNU is used with the kernel Linux. The combination of GNU and Linux is the GNU/Linux operating system, now used by millions. (Sometimes this combination is incorrectly called Linux.)

GNU

自由软件之父

- 1984年，在Richard Stallman的组织下，提出开发基于自由软件思想的GNU project—GNU(GNU是GNU is Not Unix的递归缩写)，它的意思是“不是Unix的Unix”， 即功能与UNIX完全兼容，但源代码全部重新编写的新操作系统。
  - www.stallman.org
- 为了GNU的推行，Richard建立了美国自由软件基金会FSF （Free Software Foundation，）并制定了一份公用版权协议GPL （General Public License）。
  - www.fsf.org
- GPL是开放源代码opensource的一部分,开源中有各种各样的协议。
  - www.opensource.org

Richard Stallman-自由软件之父

- 1953年，Richard Stallman出生于美国纽约。1971年，他进入哈佛大学学习。同年，他受聘于麻省理工学院（MIT）人工智能实验室，专业从事软件开发工作，并且一直在那里工作了10多年
- Stallman从事自由软件工作得到了认可，他曾获得多项大奖和荣誉：
  - 1990年度麦克阿瑟奖（MacArthur Fellowship）
  - 1991年度美国计算机协会颁发的Grace Hopper Award以表彰他所开发的的Emacs文字编辑器
  - 1996年获颁瑞典皇家理工学院荣誉博士学位
  - 1998年度电子前线基金会（Electronic Frontier Foundation）先锋奖
  - 1999年Yuri Rubinsky纪念奖
  - 2001年在苏格兰获颁格拉斯哥大学荣誉博士学位，2001年武田研究奖励赏（武田研究奖励赏）
  - 2002年成为美国国家工程院院士
  - 2003年在比利时获颁布鲁塞尔大学荣誉博士学位
  - 2004年在阿根廷获颁国立沙尔塔大学荣誉博士学位，2004年获得秘鲁国立Ingeniería大学荣誉教授
  - 2007年获颁秘鲁印加大学荣誉教授，获颁Universidad de Los Angeles de Chimbote荣誉博士学位，获颁帕维亚大学荣誉博士学位

Richard Stallman真正的力量还是他的思想。在他的理论下，用户彼此拷贝软件不但不是“盗版”，而是体现了人类互助的美德。

GNU Free Software

www.gnu.org

- Free Software自由软件：自由软件意味着使用者有运行、复制、发布、研究、修改和改进该软件的自由。
  - 自由软件是权利问题，不是价格问题。要理解这个概念，你应该考虑“自由”是“言论自由”中的“自由”；而不是“免费啤酒”中的“免费”。

- 自由软件赋予软件使用者 四项基本自由：
  - 不论目的为何，有运行该软件的自由（自由之零）。
  - 有研究该软件如何运行，以及按需改写该软件的自由（自由之一）。取得该软件源代码为达成此目的之前提。
  - 有重新发布拷贝的自由，这样你可以借此来敦亲睦邻（自由之二）。
  - 有改进该软件，以及向公众发布改进的自由，这样整个社群都可受惠（自由之三）。取得该软件源码为达成此目的之前提。

GNU与Linux(续)

- GNU也有自己的版权声明Copyleft， 与一般意义的版权Copyright相区别 。
- Copyleft 是一种让程序或其它作品保持自由的通用方法，它要求所有对Copyleft程序的修改和扩展都保持自由。
- Copyleft 的中心思想是给予每个人运行该程序、拷贝程序、修改程序和散布其修改版本的许可 -- 但是没有增加他们自己的限制的许可。

- 大约在1992年，将 Linux 与不是非常完整的 GNU 系统相结合产生了一个完整的自由软件操作系统-Linux系统。一个 GNU 系统的版本。

3. 开放源代码 (Open Source)

- OSI（Open Source Initiative，开放源代码促进会）定义：基于社区开发的、非私有的代码，可令成本更低、开发效率更高、商业应用更加灵活。
- 其应具备如下特征:
  - 自由发布，源代码开放
  - 赋予使用者修改演绎作品的权利
  - 可以要求修改后的版本以原始源代码和一组补丁文件的方式发布
  - 不得歧视任何个人和团体
  - 不得歧视人和应用领域
  - 所有的权利必须跟随再发布的软件版本一同授于使用者
  - 许可证适用于全部程序以及其中的全部组件
  - 许可证不应限制其他软件，允许开放源代码程序和封闭源代码程序一同发布

开放源代码= 代码+ 许可证 + 管理机制

Free Software与Open Source

- Richard Stallman论述Free Software（自由软件）与Open Source（开放源码）的区别：
  - 自由软件和开放源码是基于两种不同哲学理念而发起的运动，自由软件的目的在于自由的“分享”与“协作”。我认为non-free（非自由）软件是反社会的，因为它们的理念践踏了用户的自由，所以我提倡发展自由软件从而摆脱那些束缚。
  - 开放源码运动通常旨在提高技术等级，是一种技术等级发展模式，其所带来的价值跟微软所提倡的一样，都是狭窄的实际价值（narrowly practical values）。
  - 自由软件与开放源码目前都是软件许可的标准，虽然许可效果都差不多，但两个标准的注解区别却非常大，这之间最大的区别是哲学理念上的区别。
  - 为什么哲学理念会产生影响？因为人们不重视他们的自由必将失去自由，如果你给人们自由而不告诉他们重视自由，他们所拥有的自由必定不长久。所以仅仅传播自由软件远不足够，还要教导人们去渴求自由，这样或许才能让我们解决现今看来无法解决的问题。

Linux内核

- Linus领导下的开发小组开发出的系统内核 是所有Linux 发布版本的核心
- Linus最初设计Linux三原则：
  - 实用 、有限目标 、简单设计
- Linux内核目前有全球数万人参与项目开发。

- 对于 Linux 来说，最为重要的决策之一是采用 GPL（GNU General Public License）。在 GPL 保护之下，Linux 内核可以防止商业使用，并且它还从 GNU 项目（Richard Stallman 开发，其源代码要比 Linux 内核大得多）的用户空间开发受益。这允许使用一些非常有用的应用程序，如 GCC（GNU Compiler Collection）和各种 shell 支持。

Linux的版本

- Linux有版本两种表现形式：
  - 内核（Kernel）版，
  - 发行（Distribution）版 。
- Linux内核版本：
  - Linux的内核，由Linus等人在不断地开发和推出新的内核。Linux内核的官方版本由Linus本人发布。官方网站：www.kernel.org
  - Linux 内核3.0版本以后的约定：
    - 前二个数字表示版本号。修订用第三个数字表示，如：4.13.3。
    - 测试版用rc（Release Candidate 候选版本）表示，如：4.14-rc5
  - 目前基本2个多月发布一个新版本，至2020年9月23日，Latest Stable Kernel: 5.8.11

    http://www.kernel.org/

Linux的版本(续)

Linux发行版本：

- 我们将完整的Linux系统包称为发行版。有很多不同的Linux发行版来满足可能存在的各种运算需求。大多数发行版是为某个特定用户群定制的，比如商业用户、多媒体爱好者、软件开发人员或者普通家庭用户。每个定制的发行版都包含了支持特定功能所需的各种软件包，比如为多媒体爱好者准备的音频和视频编辑软件，为软件开发人员准备的编译器和集成开发环境（IDE）。
- 不同的Linux发行版通常归类为3种：
  - 完整的核心Linux发行版
  - 定用途的发行版
  - LiveCD测试发行版:可引导的Linux CD发行版,它无需安装就可以看到Linux系统是什么样的。

Linux的版本(续)

- Linux常见发行版本：
  发行版本是各个公司推出的版本，所有发行版本的内核最初都来自于kernel.org，目前常见的Linux发行版本有：
  - Red Hat http://www.redhat.com
  - Fedora core http://fedoraproject.org
  - Debian http://www.debian.org
  - SuSELinux http://www.suse.com http://www.novell.com/linux/suse
  - Ubuntu http://www.ubuntu.com/ http://www.ubuntu.org.cn/
  - Linux mint http://www.linuxmint.com/
  - CentOS http://www.centos.org/

  - 红旗Linux http://www.redflag-linux.com
  - 中软Linux http://www.cs2c.com.cn/
  - 优麒麟 http://www.ubuntukylin.com/

Linux内核

- Linus领导下的开发小组开发出的系统内核 是所有Linux 发布版本的核心
- Linus设计Linux三原则：
  - 实用 、有限目标 、简单设计
- Linux 从一个个人项目进化成为一个全球数千人参与的开发项目。
- 对于 Linux 来说，最为重要的决策之一是采用 GPL（GNU General Public License）。在 GPL 保护之下，Linux 内核可以防止商业使用，并且它还从 GNU 项目（Richard Stallman 开发，其源代码要比 Linux 内核大得多）的用户空间开发受益。这允许使用一些非常有用的应用程序，例如 GCC（GNU Compiler Collection）和各种 shell 支持。

Linux操作系统的安装

- 获取Linux发行版的方法：
  - 通过Internet下载
  - 购买Linux光盘
- 推荐的Linux发行版：
  - ubuntu， http://www.ubuntu.com
  - Fedora core， http://fedoraproject.org/

浙江大学镜像
http://mirrors.zju.edu.cn/

阿里云镜像
http://mirrors.aliyun.com/

Linux安装的常用方法：

- 1、计算机上只安装linux系统
- 2、计算机上安装多个操作系统。假设计算机已经安装了Windows　。多操作系统引导的安装方法：
  - 在windows下用分区工具（如PQmagic）在你的硬盘上划出一些空闲空间　。这些空闲空间至少分成两个分区：Linux的root分区（12~20GB）和swap分区（1GB）
  - 光盘安装，用Linux光盘启动，安装系统　。
  - 硬盘安装，从WINDOWS系统启动安装Linux。需要做一些设置工作。
- 3、虚拟机安装，虚拟机(for Windows)：VMWare Play、 VirtualBox 、Microsoft Hyper-V（Windows 7或者更新的版本，并且是Professional或是旗舰版的用户）——推荐使用这种方法

MAC OS:VMware fusion、VirtualBox

建议你使用的虚拟机和Linux发行版本

虚拟机：

- VMware Play　https://www.vmware.com/cn

  安装VMware tools后更方便与host operating system通信

- VirtualBox　https://virtualbox.org/

Linux系统发行版本（套件）：

- Ubuntu http://ubuntu.com.cn/
- http://fedoraproject.org/

浙江大学镜像：http://mirrors.zju.edu.cn/

阿里云：http://mirrors.aliyun.com/

Linux操作系统使用

- Linux操作系统的使用
  - “边干边学-Linux内核指导”教材第1-8章

Linux系统结构

<table>
<tr><td rowspan="3">User mode</td><td>User applications</td><td colspan="5">For example, bash, LibreOffice, Apache OpenOffice, Blender, 0 A.D., Mozilla Firefox, etc.</td></tr>
<tr><td>Low-level system components:</td><td>System daemons:
systemd, runit, logind, networkd, soundd, ...</td><td>Windowing system:
X11, Wayland, Mir, SurfaceFlinger (Android)</td><td>Other libraries:
GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep, etc.</td><td colspan="2">Graphics:
Mesa, AMD Catalyst,
...</td></tr>
<tr><td>C standard library</td><td colspan="5"><code>open()</code>, <code>exec()</code>, <code>sbrk()</code>, <code>socket()</code>, <code>fopen()</code>, <code>calloc()</code>, ... (up to 2000 subroutines)
glibc aims to be POSIX/SUS-compatible, uClibc targets embedded systems, bionic written for Android, etc.</td></tr>
<tr><td rowspan="3">Kernel mode</td><td rowspan="3">Linux kernel</td><td colspan="5"><code>stat</code>, <code>splice</code>, <code>dup</code>, <code>read</code>, <code>open</code>, <code>ioctl</code>, <code>write</code>, <code>mmap</code>, <code>close</code>, <code>exit</code>, etc. (about 380 system calls)
The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS-compatible)</td></tr>
<tr><td>Process scheduling subsystem</td><td>IPC subsystem</td><td>Memory management subsystem</td><td>Virtual files subsystem</td><td>Network subsystem</td></tr>
<tr><td colspan="5">Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter
Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="7">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

Linux系统结构(续)

你必须要了解的内容

Linux系统的文件系统构成。

- 在系统启动后，进入系统所能观察到的就是一系列目录（使用ls或dir）。Linux系统根目录下包含（不同的发行版会有所区别）：
  - bin：该目录存放最常用的基本命令，比如拷贝命令cp、编辑命令vi、删除命令rm等。
  - boot:该目录包含了系统启动需要的配置文件、内核（vmliuxz）和系统镜像(initrd….img)等。
  - dev：该目录下存放的是Linux中使用或未使用的外部设备文件（fd代表软盘，hd代表硬盘等），使用这些设备文件可以用操作文件的方式操作设备。
  - etc：该目录下包含了所有系统服务和系统管理使用的配置文件；比如系统日志服务的配置文件syslog.conf，系统用户密码文件passwd等

Linux系统结构(续)

- home:该目录下包含了除系统管理员外的所有用户的主目录，用户主目录一般以用户登陆帐号命名。
- lib:该目录下包含了系统使用的动态连接库（*.so）和内核模块(在modules下)。
- host+found:该目录包含了磁盘扫描检测到的文件碎片，如果你非法关机，那么下次启动时系统会进行磁盘扫描，将损坏的碎片存到该目录下。
- mnt:该目录下包含用户动态挂载的文件系统。如果要使用光盘，U盘都一般应该将它们安装到该目录下的特定位置。
- proc：该目录属于内存影射的一个虚拟目录，其中包含了许多系统现场数据，比如进程序数，中断情况，cpu信息等等，它其中的信息都是动态生成的，不在磁盘中存储。

Linux系统结构(续)

- root:该目录是系统管理员（root用户）的主目录。
- sbin:该目录下包含系统管理员使用的系统管理命令,比如防火墙设置命令iptable，系统停机命令halt等
- tmp：该目录下包含一些临时文件。
- usr：该目录下一般来说包含系统发布时自带的程序（但具体放什么东西，并没有明确的要求），其中最值得说明的有三个子目录
  - /usr/src：Linux内核源代码就存在这个目录
  - /usr/man：Linux中命令的帮助文件
  - /usr/local：新安装的应用软件一般默认在该目录下
- var：该目录中存放着在不断扩充着的信息，比如日志文件。

linux编程

- Linux的编辑器
- Shell程序设计
- Linux的C程序设计
- Linux的汇编语言
- 开发工具

Linux的编辑器

- 命令行方式
  - vi (vim): 最令UNIX类操作系统初学者裹足不前的editor,然而只要你习惯于操作，你会觉得它比任何的editor都好用，且功能强大。
  - emacs： linux编辑器，功能强大的全屏幕编辑器。
- X-window
  - gedit、kedit 全屏幕文本编辑程序
  - emacs 编程编辑器
  - 各种发行版都有各自的编辑器

shell程序设计

- shell基本语法规则
- shell变量
- shell控制流：test，if，case，while，until，for
- shell函数
- shell程序的编写和执行，参考资料：
  - 教材 第8章

Linux的C程序设计

- Linux内核的主体是以GNU的C语言编写的。GNU中的C/C++语言编译工具是gcc。GNU对C语言本身（在ANSI C基础上）作了不少扩充。

GNU C的扩充　　如：

- 吸收了C++中的inline和const
- 为了支持64位CPU，增加了新的基本数据类型long long int
- 分支声明
  - 使用likely()和unlikely()宏对条件选择进行优化
    - if(foo){…}
    - 当foo大多数时间都会为1时：if(likely(foo)){…}
    - 当foo大多数时间都会为0时if(unlikely(foo)){…}
- 许多C语言支持属性描述符，如“aligned”，“packed”等。由于这些在ANSI C中不是保留字，所以可能引起冲突。GNU C支持在前后加上“__”来区分。
  - 如“__inline__”等于保留字“inline”。

程序扩展名—约定

.c —c语言源程序

.C .cc .cxx —c++语言源程序

.s .S —汇编语言源程序

.h —头文件

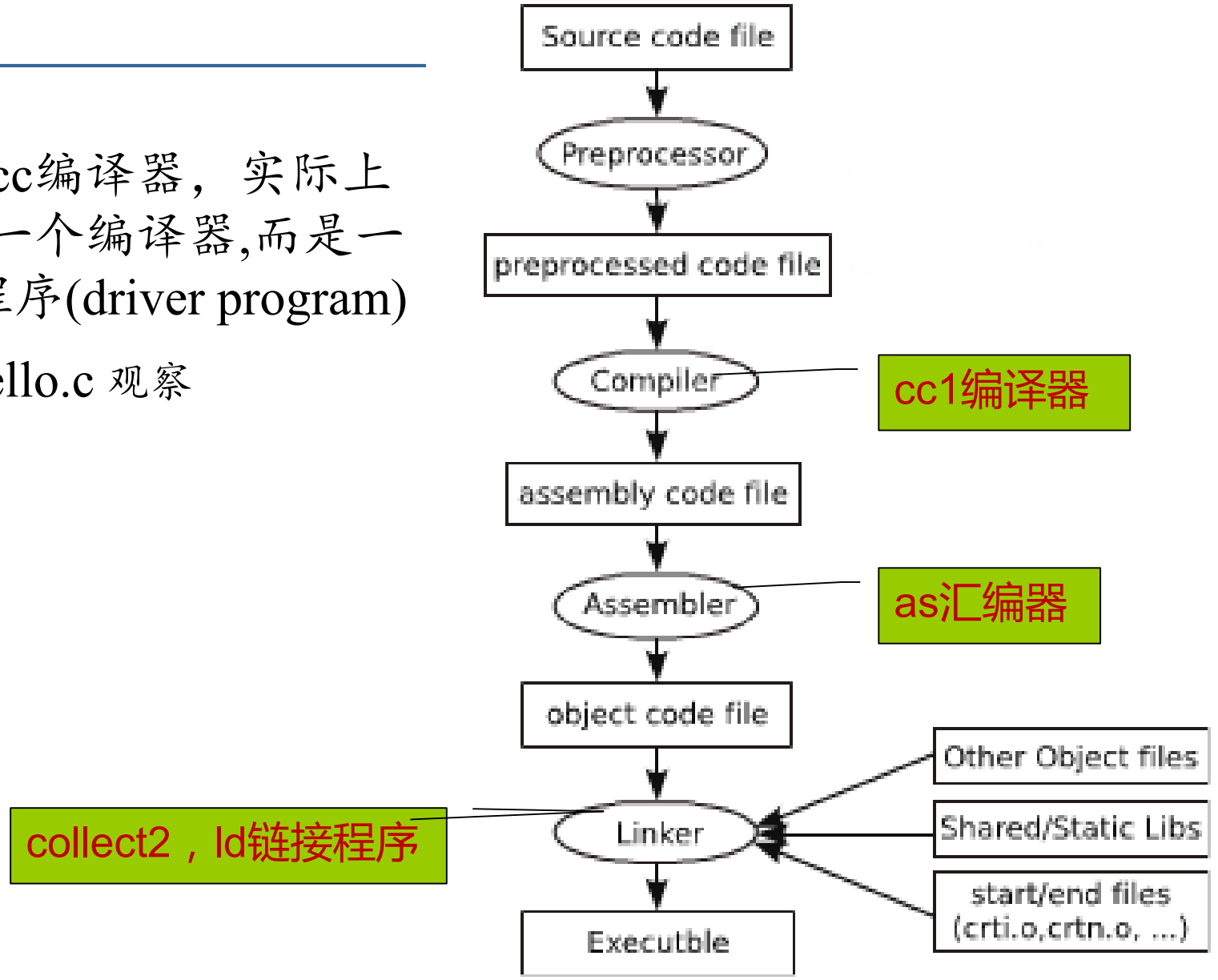
.o —目标文件（可执行文件）

.a .so .sa —库文件

gcc

- GNU CC（简称为gcc，GNU compiler collection）是GNU项目中符合ANSI C标准的**编译器集合**，能够编译用C、C++和Object C等语言编写的程序。
- gcc不仅功能强大，而且可以编译如C、C++、Object C、Java、Fortran、Pascal、Modula-3和Ada等多种语言，而且gcc又是一个交叉平台编译器，它能够在当前CPU平台上为多种不同体系结构的硬件平台开发软件，因此尤其适合在嵌入式领域的开发编译。
- 教材第7章

- 通常称gcc编译器，实际上gcc不是一个编译器,而是一个驱动程序(driver program)
- gcc –v hello.c 观察



gcc所支持后缀名解释

后 缀 名	所对应的语言	后 缀 名	所对应的语言
.c	C 原始程序	.s/.S	汇编语言原始程序
.C/.cc/.cxx	C++原始程序	.h	预处理文件（头文件）
.m	Objective-C 原始程序	.o	目标文件
.i	已经过预处理的 C 原始程序	.a/.so	编译后的库文件
.ii	已经过预处理的 C++原始程序		

gcc --- 常用选项

选项	含义
-c	只编译不链接，生成目标文件“.o”
-S	只编译不汇编，生成汇编代码
-E	只进行预编译，不做其他处理
-g	在可执行程序中包含标准调试信息
-o file	指定将 file 文件作为输出文件
-v	打印出编译器内部编译各过程的命令行信息和编译器的版本
-I dir	在头文件的搜索路径列表中添加 dir 目录

gcc ---库选项

选　　项	含　　义
-static	进行静态编译，即链接静态库，禁止使用动态库
-shared	1．可以生成动态库文件 2．进行动态编译，尽可能地链接动态库，只有没有动态库时才会链接同名的静态库（默认选项，即可省略）
-L dir	在库文件的搜索路径列表中添加 dir 目录
-lname	链接称为 libname.a（静态库）或者 libname.so（动态库）的库文件。若两个库都存在，则根据编译方式（-static 还是-shared）而进行链接
-fPIC（或-fpic）	生成使用相对地址的位置无关的目标代码（Position Independent Code）。然后通常使用 gcc 的-static 选项从该 PIC 目标文件生成动态库文件。

如： -lpthread

gcc

gcc命令

- 语法：gcc [选项] 文件列表
- 常用选项/功能：

  -c　　编译成目标（.o）文件

  -l 库文件名　连接库文件

  -o 文件名　将生成的可执行文件保存到指定文件中，默认是a.out

C程序的编译

- mypro1.c文件：

```
#include <stdio.h>
int main() {
    printf("hello world!\n");
    }
```

$为操作系统缺省提示符，root账号的提示符通常为#

- 利用如下的命令可编译生成可执行文件：

```
$ gcc -o mypro1 mypro1.c
```

- 生成了mypro1可执行文件，运行这个程序输入

```
$ ./mypro1
Hello  world!
```

- 如果没有-o选项，则生成a.out执行文件

```
$ gcc mypro1.c
$ ./a.out
```

头文件

- GCC编译器缺省的头文件目录是/usr/include目录及其子目录下。
- 那些依赖于特定 Linux版本的头文件通常可在目录/usr/include/sys和/usr/include/linux中找到。
- 其他编程系统也有各自的include文件，并将其存储在可被相应编译器自动搜索到的目录里。例如，X视窗系统的/usr/include/X11目录和GNU C++的/usr/include/c++目录
- 调用C语言编译器时，我们可以使用-I标志来包含保存在子目录或非标准位置中的include文件。例如：

  gcc –I /usr/openwin/include power.c

  - 它指示编译器不仅在标准位置，也在/usr/openwin/include目录中查找程序power.c中包含的头文件。

库文件

a代表传统的静态函数库；so代表共享函数库。

- 使用gcc的-l选项可以连接已有的程序库
  - 数学库libm.a链接到power.o

gcc –o power power.o -lm

gcc –o power power.o /usr/lib/libm.a

文件名中“lib”以后，扩展名以前的部分

  - 链接pthreads线程库

gcc–o p1-1 p1-1.c -lpthread

- 请参看C语言编译器的使用手册（man gcc）以了解更多细节。

Linux中的汇编语言

- 汇编语言程序一般以.S为扩展名。也可以以“嵌入式”汇编的方式出现在C语言的.c程序中。
- Linux的汇编语言，采用的是AT&T的386汇编语言。与Intel的汇编语言相比，二者所基于的硬件知识是相同的，但是，在语法上有一定的差异。
- Linux与Intel 汇编语言，主要有以下一些差别：
  - 在Intel格式中大多使用大写字母，而在AT&T格式中都使用小写字母；
  - 在AT&T格式中，寄存器名要加上“%”作为前缀，而在Intel格式中则不带前缀
  - 在AT&T的386汇编语言中，指令的源操作数与目标操作数的顺序与在Intel386汇编语言中正好相反。
  - 在AT&T格式中，访内存指令的操作数大小由操作码名称的最后一个字母来决定。而Intel格式则要在表示内存单元的操作数前加上属性保留字来表示。

Linux中的汇编语言

- 在AT&T格式中，直接立即数要加上“$”作为前缀，而Intel格式中不要；
- 在AT&T格式中，绝对转移或调用指令jump/call的操作数要加上“*”做为前缀，而在Intel中不带；
- 远程的转移指令和子程序调用指令的操作码名称，在AT&T格式中为“ljump”和“lcall”,而在Intel格式中为“JMP FAR”和“CALL FAR”；

Linux 开发工具

- make工具，与之有关的makefile or Makefile文件
- gdb 功能强大调式器
- 数据库
- 各种语言
- GUI

make工程管理器

- make程序：是一个命令工具，是一个解释makefile中指令的命令工具
- make工具的使用参考教材的第8章

句法：	make［选项］［目标］［宏定义］
用途：	make工具根据名为makefile或Makefile的文件中指定的依赖关系对系统进行更新。[选项][目标][宏定义]可以以任意顺序指定。
常用选项/特性：	-d 显示调试信息 -f 文件 此选项告诉make使用指定文件作为依赖关系文件，而不是默认的makefile或Makefile，如果指定的文件名是"-"，那么make将从标准输入读入依赖关系。 -h 显示所有选项的帮助信息 -n 测试模式，并不真的执行任何命令，只是显示输出这些命令 -s 安静模式--不输出任何提示信息。

make程序

- make工具依赖一个特殊的，名为makefile的文件，这个文件描述了系统中各个模块之间的依赖关系。
- GNU make的主要功能是读进一个文本文件makefile并根据makefile的内容执行一系列的工作。
- makefile的默认文件名为GNUmakefile、makefile或Makefile，当然也可以在make的命令行中指定别的文件名。如果不特别指定，make命令在执行时将按顺序查找默认的makefile文件。
- 多数Linux程序员使用第三种文件名Makefile。因为第一个字母是大写，通常被列在一个目录的文件列表的最前面。

Linux 内核编程风格

- Linux内核缩进风格是8个字符。
- Linux内核风格采用K&R标准，将开始的大括号放在一行的最后，而将结束的大括号放在一行的第一位。
- 命名尽量简洁。不应该使用诸如ThisVariableIsATemporaryCounter之类的名字。应该命名为tmp，这样容易书写，也不难理解。但是命名全局变量，就应该用描述性命名方式，例如应该命名“count_active_users()”，而不是“cntusr()”。本地变量应该避免过长。
- 函数最好短小精悍，一般来说不要让函数的参数多于１０个，否则应该尝试分解这个过于复杂的函数。
- 通常情况，注释说明代码的功能，而不是其实现原理。避免把注释插到函数体内，而写到函数前面，说明其功能，如果这个函数的确很复杂，其中需要有部分注释，可以写些简短的注释来说明那些重要的部分，但是不能过多。

Linux内核使用GNU C和AT&T汇编。

系统调用

- 系统调用：在用户进程和硬件设备之间添加了一个中间层。应用程序调用操作系统提供的功能模块（函数）。
  - 对文件和设备进行访问和控制需要使用系统调用实现
  - Linux程序员通过API函数使用系统调用
- C函数库(glibc)
  - 依赖于系统调用
  - 一般来说，标准函数库建立在系统调用的上层，提供的功能比系统调用强，使用也比较方便。
  - 例：标准I/O库：fopen函数 。系统调用为open
- Linux的系统调用作为c库的一部分提供。c库中实现了Linux的主要API，包括标准c库函数和系统调用。

POSIX

- **POSIX**(Portable Operating System Interface)可移植操作系统接口, the IEEE's Portable Application Standards Committee,
  - https://collaboration.opengroup.org/external/pasc.org/plato/
- POSIX API:定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称

fork函数

- **fork:创建一个新子进程**

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

  - 返回值：子进程中为0，父进程中为子进程号PID，出错为-1
  - 该函数被调用一次，但返回两次。两次返回的区别是子进程的返回值是0，而父进程的返回值则是新子进程的进程ID。

所需头文件	#include<unistd.h>
函数功能	建立一个新的进程
函数原型	pid_t fork(void);
函数传入值	无
函数返回值	执行成功则在子进程中返回0，在父进程会返回新建立子进程的进程号（PID），失败则返回－1，失败原因存于errno中

exec函数

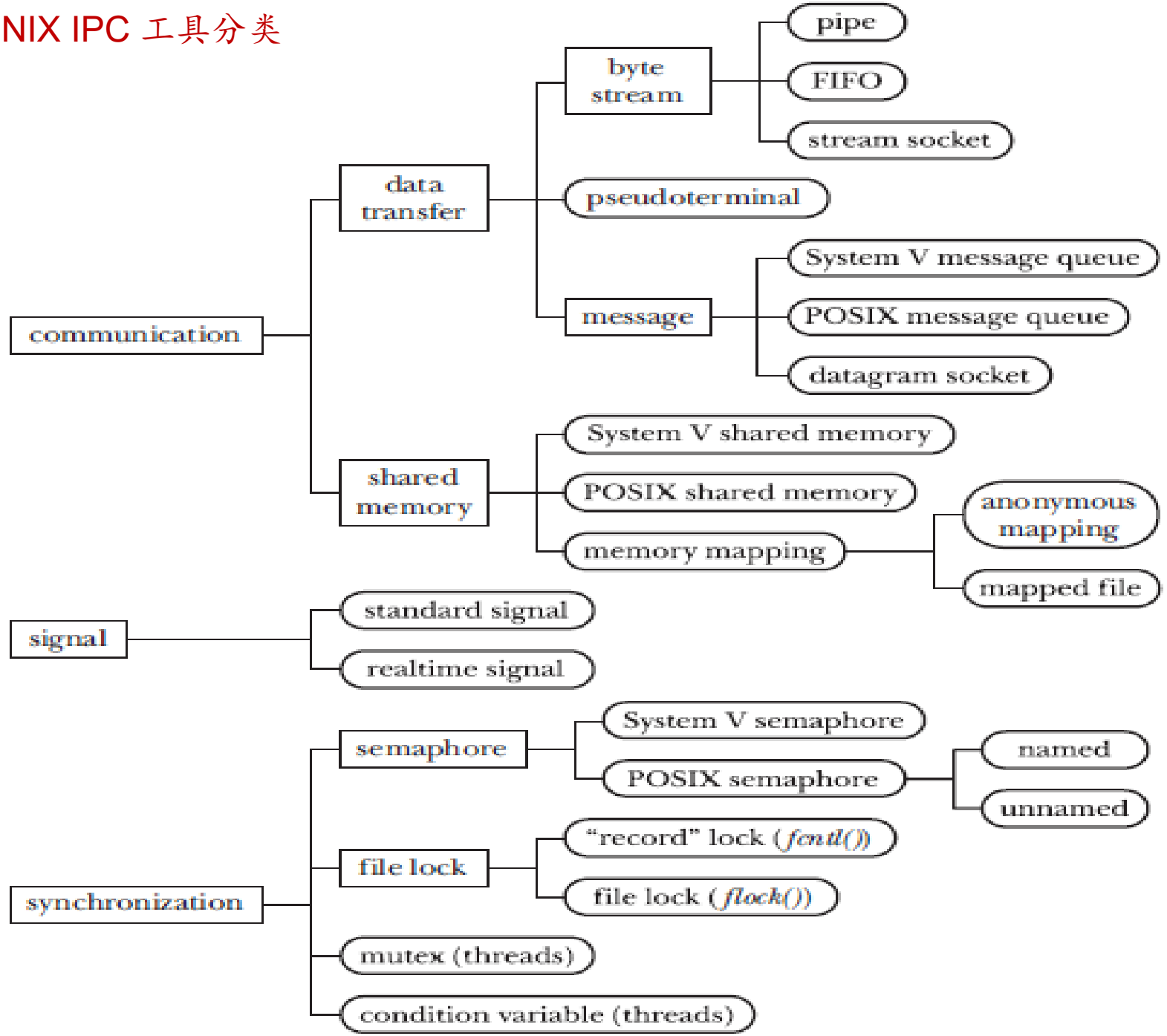
- 用fork函数创建子进程后，子进程往往要调用一种exec函数以执行另一个程序。
- 当进程调用一种exec函数时，该进程完全由新程序代换，而新程序则从其main函数开始执行。
- 调用exec并不创建新进程，所以前后的进程PID并未改变。exec只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。

Linux进程通信机制

Linux进程通信机制

- Linux实现进程间通信(IPC Inter Process Communication)：
  - System V IPC机制:
    - 信号量、消息队列、共享内存
  - 管道（pipe）、命名管道
  - 套接字（socket）
  - 信号(signal)
  - 文件锁(file lock)
  - POSIX线程：
    - 互斥锁(互斥体、互斥量)（mutex）、条件变量(condition variables)
  - POSIX：
    - 消息队列、信号量、共享内存

UNIX IPC 工具分类



System V IPC相关的系统调用

- 用信号量对进程要访问的临界资源进行保护:
  - semget()：创建或打开信号量；
  - semop()：增加或减少一个信号量的值；
  - semctl()：对信号量进行控制。
- 用消息队列在进程间以异步方式发送消息:
  - msgget()：创建新消息队列或打开现有消息队列；
  - msgsnd()：发送消息；
  - msgrcv()：接收消息；
  - msgctl()：对消息队列进行控制，如删除消息队列。
- 提供一块预留出的共享内存区域供进程之间交换数据:
  - shmget()：创建或打开共享内存区。
  - shmat ()：连接共享内存区。
  - shmdt ()：拆除共享内存区连接
  - shmctl()：共享内存储区控制

管道通信

UNIX系统在操作系统的发展中最重要贡献之一是该系统首创了管道（pipes）。

- 管道是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称为pipe文件。向管道（共享文件）提供输入的发送进程（即写进程），而接收管道输出的接收进程（即读进程）可从管道中接收数据。管道通信是基于文件系统形式的一种通信方式。
- 管道分为无名管道和有名管道两种类型。无名管道是一个只存在于打开文件机构中的一个临时文件，从结构上没有文件路径名，不占用文件目录项。无名管道利用系统调用pipe(filedes)创建。

管道

- pipe使用

pipe系统调用的语法格式是：

```
int filedes [2];
int pipe (filedes);
```

核心创建一条管道完成下述工作：

- 分配一个隶属于root文件系统的磁盘和内存索引结点inode
- 在系统打开文件表中分别分配一读管道文件表项和一写管道文件表项
- 在创建管道的进程文件描述表中分配二表项，filedes[0]和filedes[1]分别指向系统打开文件表的读和写管道文件表项
- 系统调用所涉及的数据结构如下

Linux线程和线程库

Linux线程

- 线程(thread)是允许应用程序并发执行多个任务的一种机制
- Linux 2.6内核支持clone()系统调用创建线程。
- POSIX threads, often known as Pthreads.
- Pthreads是POSIX标准为线程操作定义的API

Pthreads数据类型

- Pthreads API定义了一些数据类型

Data type	Description
pthread_t	线程ID
pthread_mutex_t	互斥锁对象
pthread_mutexattr_t	互斥锁属性对象
pthread_cond_t	条件变量
pthread_condattr_t	条件变量属性对象
pthread_attr_t	线程属性对象

POSIX线程库

- POSIX线程库(Pthreads)：
  - pthread_create()：创建线程函数；
  - pthread_exit()：主动退出线程；
  - pthread_join()：用于将当前线程挂起来等待线程的结束。这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源就被收回。
  - pthread_cancel()：终止另一个线程的执行。

线程创建

所需头文件	#include< pthread.h >
函数功能	创建一个线程
函数原型	**int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void *(*start_routine)(void*), void * arg);**
函数参数	**第1个参数：产生线程的标识符** **第2个参数：所产生线程的属性，通常设为NULL** **第3个参数：新的线程所执行的函数代码** **第4个参数：新的线程函数的参数**
函数返回值	执行成功则返回0，失败返回-1，而错误号保存在全局变量errno中。

例：**pthread_create(&tid,&attr,runner,argv[1]);**

线程终止 pthread_exit

所需头文件	#include< pthread.h >
函数功能	终止一个线程
函数原型	**void pthread_exit(void *value_ptr);**
函数传入值	value_ptr：用户定义的指针，用来存储被等待线程的返回值。
函数返回值	执行成功则返回0，失败返回-1，而错误号保存在全局变量errno中。
备注	主线程中使用pthread_exit只会使主线程自身退出，产生的子线程继续执行；用return则所有线程退出。子线程或主线程使用exit，则整个线程全部终止。

等待线程结束

所需头文件	#include< pthread.h >
函数功能	等待线程结束
函数原型	**int pthread_join(pthread_t thread, void **value_ptr);**
函数传入值	thread：等待线程的标识符 value_ptr：用户定义的指针，用来存储被等待线程的返回值。
函数返回值	执行成功则返回0，失败返回错误号。
备注	

创建线程例

```
void * thread_fun1(void *arg){
    printf("thread %d  return\n",arg);
     ......
}
int main(void){
       pthread_t   tid1, tid2;
      pthread_create(&tid1, NULL, thread_fun, (void *)1); /*创建第1个线程*/
      pthread_create(&tid2, NULL, thread_fun, (void *)2);  /*创建第2个线程*/
      ......
      pthread_join(tid1, NULL);          /*等待第1个线程结束*/
       pthread_join(tid2, NULL);          /*等待第2个线程结束*/
      ......
}
```

例：设计使用pthreads线程库创建2个新线程，在父进程（也可以称为主线程）和新线程中分别显示进程id和线程id，并观察线程id数值。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>  /*pthread_create()函数的头文件*/
#include <unistd.h>
pthread_t ntid1,ntid2;
void printids(const char *s) {/*各线程共享的函数*/
        pid_t     pid;
        pthread_t  tid;
        pid = getpid();
        tid = pthread_self();
        printf("%s  pid=  %u  tid= %u  （0x%x）\n", s, (unsigned int)pid,        (unsigned int)tid, (unsigned int)tid);
}
void *thread_fun(void *arg) {/*新线程执行代码*/
        printids(arg);
        return NULL;
}
```

```
int main(void){
        int err;
                /*下列函数创建线程*/
        err = pthread_create(&ntid1, NULL, thread_fun, "我是新线程1: ");
        if (err != 0) {printf(stderr, "创建线程1失败);
        exit(1);     }
        err = pthread_create(&ntid2, NULL, thread_fun, "我是新线程2: ");
        if (err != 0) {printf("创建线程2失败);
        exit(1);}
        printids("我是父进程:");
        pthread_join(ntid1, NULL);        /*等待第1个线程结束*/
        pthread_join(ntid2, NULL);        /*等待第2个线程结束*/
        return 0;
}
```

编译pthreads线程方法

- 编译pthreads线程程序的命令：

  gcc -o ptest ptest.c –lpthread ——— ubuntu

  gcc –lpthread -o ptest ptest.c

Pthreads同步互斥机制

Pthreads互斥锁

- 互斥锁是用一种简单的加锁方法来控制对共享资源的原子操作。这个互斥锁只有两种状态，也就是**上锁**和**解锁**。在同一时刻只能有一个线程掌握某个互斥锁，拥有上锁状态的线程能够对共享资源进行操作。若其他线程希望上锁一个已经被上锁的互斥锁，则该线程就会等待，直到上锁的线程释放掉互斥锁为止。互斥锁保证让每个线程对共享资源按顺序进行原子操作。
- 互斥锁机制主要包括下面的基本函数
  - 互斥锁初始化：pthread_mutex_init()
  - **互斥锁上锁：pthread_mutex_lock()**
  - **互斥锁解锁：pthread_mutex_unlock()**
  - 消除互斥锁：pthread_mutex_destroy()

Pthreads互斥锁

所需头文件	#include <pthread.h>	
函数原型	int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr)	
函数传入值	mutex：互斥锁	
函数传入值	Mutexattr	PTHREAD_MUTEX_INITIALIZER：创建快速互斥锁
		PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP：创建递归互斥锁
		PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP：创建检错互斥锁
函数返回值	成功：0	
	出错：返回错误码	

```
pthread_mutex_t mutex;   //声明
pthread_mutex_init (&mutex, NULL); //初始化

或：
pthread_mutex_t mutex =PTHREAD_MUTEX_INITIALIZER;
```

Pthreads互斥锁

所需头文件	#include <pthread.h>
函数原型	int pthread_mutex_lock(pthread_mutex_t *mutex,) int pthread_mutex_trylock(pthread_mutex_t *mutex,) int pthread_mutex_unlock(pthread_mutex_t *mutex,) int pthread_mutex_destroy(pthread_mutex_t *mutex,)
函数传入值	mutex：互斥锁
函数返回值	成功：0
	出错：-1

例

```
pthread_mutex_lock(&mutex);
//…….
pthread_mutex_unlock(&mutex);
```

Pthreads 条件变量(Condition Variables)

- **条件变量表示一种等待原因。**条件变量允许一个线程就某个共享变量（或其他资源）的状态变化通知其它线程，允许其它等待（阻塞）这一通知（allows the other threads to wait (block) for such notification）。
- 线程可以对一个条件变量执行等待操作。如果线程A正在等待一个条件变量，它会被阻塞，直到另外一个线程B向同一个条件变量发送信号以改变其状态。线程A必须在B发送信号之前开始等待。如果B在A执行等待操作之前发送了信号，这个信号就丢失了，同时A会一直阻塞直到其它线程再次发送信号到这个条件变量。

条件变量

- 条件变量的类型为：**pthread_cond_t**

- **pthread_cond_init** 初始化一个条件变量
- **pthred_cond_wait** 调用的线程阻塞直到条件变量收到信号
- **pthread_cond_signal** 向一个条件变量发送信号
- **pthread_cond_broadcast** 将所有等待该条件变量的线程解锁

条件变量

- 条件变量静态分配：

  pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

- 条件变量初始化：

  int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);

- 等待操作：

  int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

- 发送信号操作：

  int pthread_cond_signal(pthread_cond_t *cond);

条件变量

例：

```
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

……
pthread_mutex_lock(&mutex);
  pthread_cond_wait(&cond, &mutex);
  //当执行pthread_cond_wait本线程进入等待后，pthread_cond_wait会调pthread_mutex_unlock释放互斥锁，以使其他线程使用互斥锁
pthread_mutex_unlock(&mutex);
```

- 生产者消费者例：prod_condvar.c

Pthreads信号量机制

- sem_init()用于创建一个信号量，并初始化它的值。
- sem_wait()和sem_trywait()都相当于P操作，在信号量大于零时它们都能将信号量的值减一，两者的区别在于若信号量小于零时，sem_wait()将会阻塞进程，而sem_trywait()则会立即返回。
- sem_post()相当于V操作，它将信号量的值加一同时发出信号来唤醒等待的进程。
- sem_getvalue()用于得到信号量的值。
- sem_destroy()用于删除信号量。

Pthreads信号量机制

所需头文件	#include <semaphore.h>
函数原型	int sem_init(sem_t *sem,int pshared,unsigned int value)
函数传入值	sem：信号量指针
	pshared：决定信号量能否在几个进程间共享。由于目前 Linux 还没有实现进程间共享信号量，所以这个值只能够取 0，就表示这个信号量是当前进程的局部信号量
	value：信号量初始化值
函数返回值	成功：0
	出错：−1
所需头文件	#include <pthread.h>
函数原型	int sem_wait(sem_t *sem) int sem_trywait(sem_t *sem) int sem_post(sem_t *sem) int sem_getvalue(sem_t *sem) int sem_destroy(sem_t *sem)
函数传入值	sem：信号量指针
函数返回值	成功：0
	出错：−1

Q&A