Linux系统分析

1.课程简介

张雷

zhangl@nju.edu.cn

南京大学计算机科学与技术系

About me

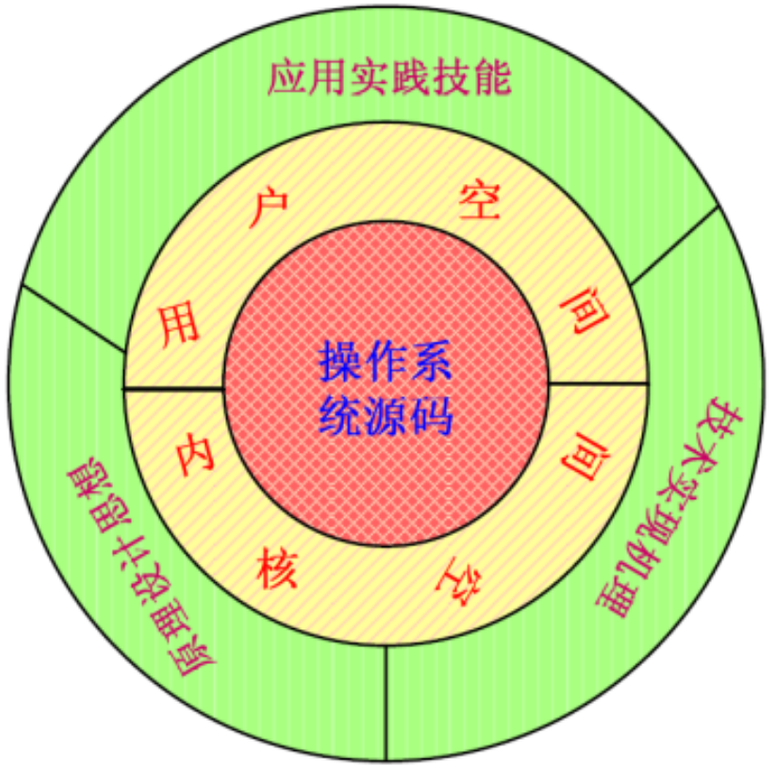
- 张雷，计算机系助理研究员（2014- ）
  - 主页：http://cs.nju.edu.cn/zhangl
  - 邮件：zhangl@nju.edu.cn
  - 房间：计算机楼1019
- 研究
  - 智能体与多智能体系统

课程概况

- 性质
  - 选修课
- 课时
  - 2学时/周
- 课程前导知识（推荐）
  - 操作系统原理
  - C语言编程
  - 汇编语言
  - Linux系统编程

课程目标

- 以Linux内核为基础，结合操作系统基本概念及原理：
  - 剖析实用操作系统的实现机制（内核角度）
  - 理解操作系统设计精髓（实现问题）
  - 强化应用实践技能（用户空间角度）



学习资源

- 课程主页：http://cs.nju.edu.cn/zhangl/linux/
- 参考资料
  - 《Linux操作系统实验教程》，费翔林主编，高等教育出版社，2009年4月
  - 《Linux Kernel Development》（第3版），机械工业出版社，2011年
  - 《深入理解Linux内核》（第3版），陈莉君等译，中国电力出版社，2007年
- 微信群
- 云主机
  - ssh 学号@210.28.133.11 -p 21317
  - 用户名密码：学号
  - 32 Core/64GB/512GB

学习资源

- Linux/UNIX系统编程
  - TCPL: The C Programming Language，Brian Kernighan and Dennis Ritchie，1978
  - APUE：Advanced Programming in the Unix Environment（UNIX环境高级编程）W.Richard Stevens，2006
  - The Linux Programming Interface: A Linux and UNIX System Programming Handbook (Linux/UNIX系统编程手册)，Michael Kerrisk， 2014
- Linux内核
  - Professional Linux Kernel Architecture，Wolfgang Mauerer，2008
  - 边学边干-Linux内核指导，第二版，李善平，浙江大学出版社，2008
- 其他
  - OSDI, USENIX Symposium on **Operating Systems** Design and Implementation
  - SOSP, ACM Symposium on Operating Systems Principles

教学方式

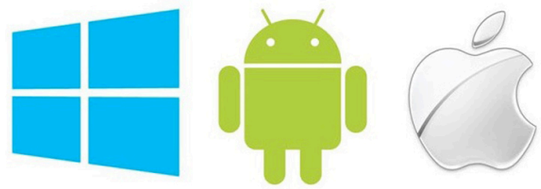
- 教学形式
  - 单周课堂讲座
  - 双周上机实验（地点：基础实验楼乙126）

- 课程考评方式
  - 实验（80%）：6个实验, 分数最低的实验不计，题目可自选
  - 课堂报告（15%）: 自选题目
  - 课堂参与（5%）

操作系统是什么

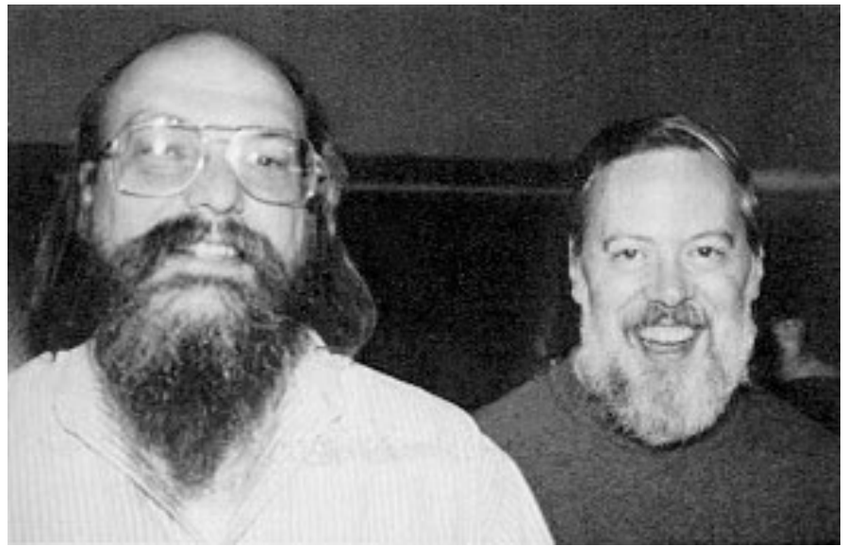
- Wikipedia：An **operating system** (**OS**) is system software that manages computer hardware and software resources and provides common services for computer programs.

- 操作系统：政府，警察，保安，助手，机器界面 …
- 操作系统：Windows，Android，OSX …
- …



Unix

- 1969, Bell Labs, AT&T
  - Ken Thompson, 1943-
  - Dennis Ritchie, 1941-2011

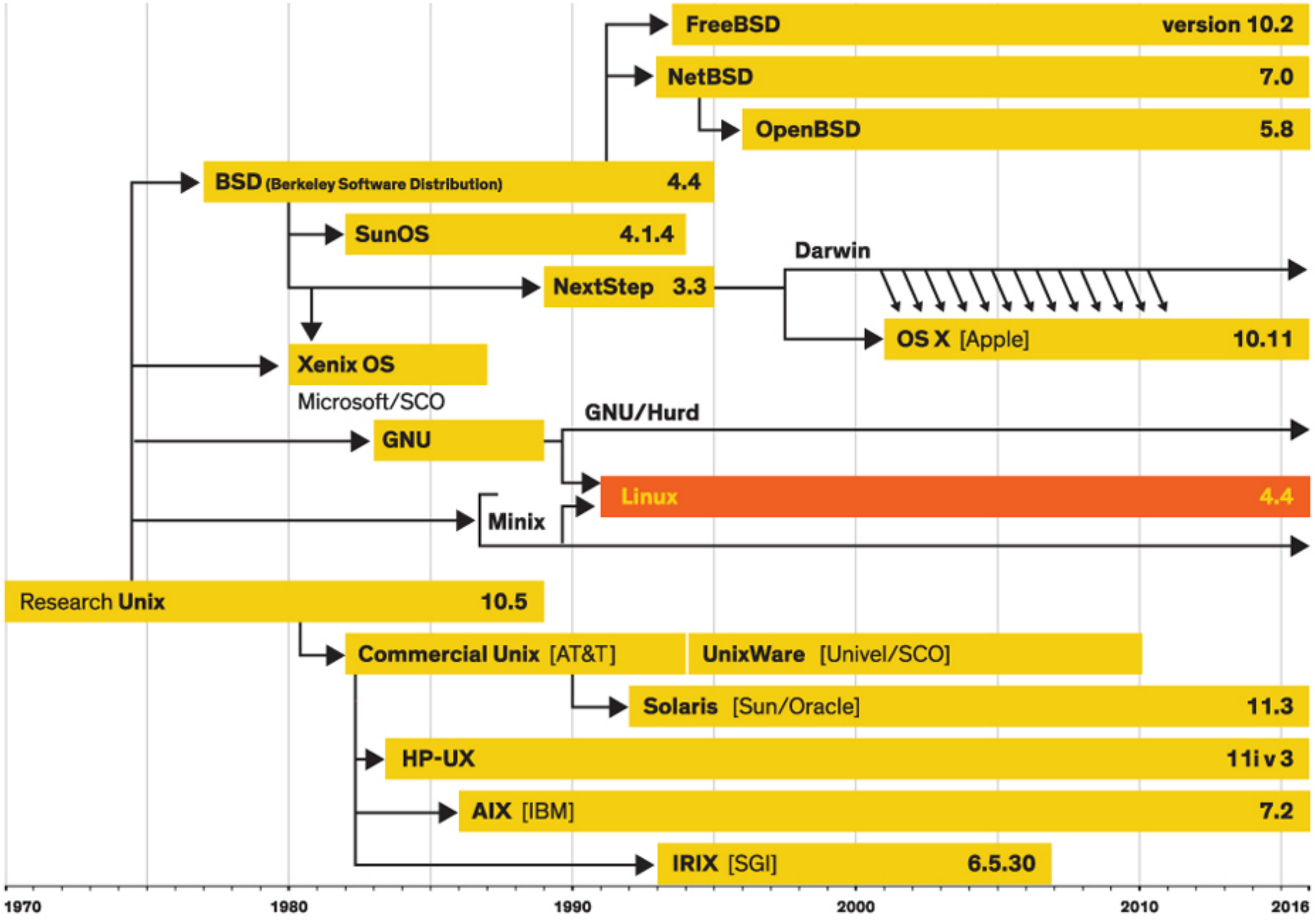


- Design: simplicity and elegance
  - everything is a file: open(), read(), write(), close()
    - file: array of bytes, unstructured
  - do one thing and do it well
  - separation between policy and mechanism

Unix

- AT&T monopoly
  - 1956美国DOJ和解协议， AT&T不进入计算机业，不销售任何与计算机有关的产品
  - 1974年 Berkeley获得Unix源码，BSD开始开发
  - 1974年美国DOJ起诉AT&T违反《反垄断法》
  - 1982年AT&T败诉分拆为7个公司，和解协议失效，AT&T进入计算机业，产生商业版Unix
- 80年代末，Intel 80x86芯片巨大发展，PC时代来临，OS平台化

Unix


FreeBSD version 10.2
NetBSD 7.0
OpenBSD 5.8
BSD (Berkeley Software Distribution) 4.4
SunOS 4.1.4
Darwin
NextStep 3.3
OS X [Apple] 10.11
Xenix OS
Microsoft/SCO
GNU/Hurd
GNU
Linux 4.4
Minix
Research Unix 10.5
Commercial Unix [AT&T]
UnixWare [Univel/SCO]
Solaris [Sun/Oracle] 11.3
HP-UX 11i v3
AIX [IBM] 7.2
IRIX [SGI] 6.5.30
1970
1980
1990
2000
2010
2016

Linux

- 1991年11月：Linux雏形代码
  - 芬兰赫尔辛基大学的Linus Torvalds（1969-）开发一个小程序（取名为Linux），希望借此实现一个操作系统“内核”
- 1993年：Linux 1.0诞生
- 1994年：Linux的第一个商业版 Slackware问世
- 1996年：符合POSIX标准(Linux 1.2.13)
- 2001年：Linux 2.4版内核发布
- 2003年：Linux 2.6版内核发布
- 2012年：Linux 3.2、3.4、3.5、3.6内核相继发布

Linux特征

- 符合POSIX标准规范的操作系统
  - Portable Operation System Interface of Unix：可移植的操作系统接口
  - 由IEEE开发，ANSI和ISO标准化
- Free or open software
  - GNU General Pubic Licese（GPL）2.0
  - 可在原程序基础上自由进行修改，开发自己的程序
  - 发布修改时需要提供源代码
  - 开源文化

Linux与Windows的区别

- 文件系统
  - Linux需要一个挂载根目录/的ext分区和一个作为虚拟内存的swap分区
  - Linux没有盘符，可通过设备名挂载，挂在信息在/dev/fstab，如
    - mount -t ntfs /dev/sda1 /mnt/win_c
- Linux将所有设备都映射成/dev目录下的一个文件
- 文件在使用中不影响文件删除
- 用户管理
  - 系统管理员是root，使用su命令切换
  - sudo passwd
- …

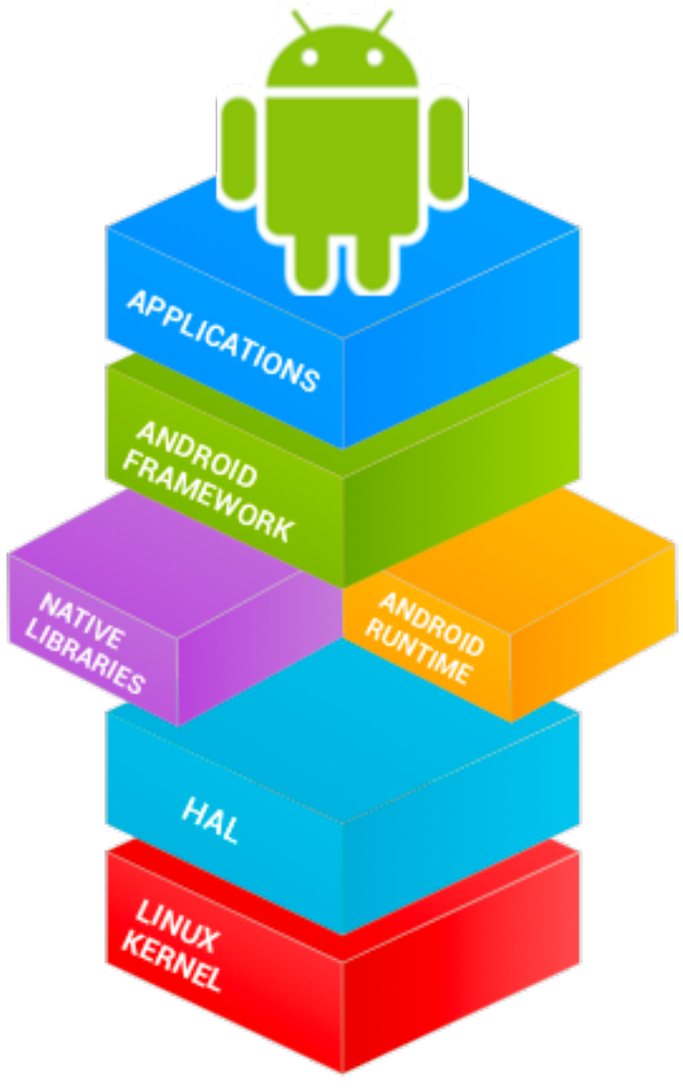
Market share of Linux

Category	Source	Date	Linux	Unix and Unix-like	Windows
Desktop, laptop, netbook(excluding Android)	Net Applications	April 2016	1.65% (Ubuntu, etc.)	9.57% (OS X)	88.77%(10,8.1, 7, Vista,XP and older)
Smartphone, tablet, smart TV,Wearable computer	StatCounterGlobal Stats	May 2016	**64.89%** (Android)	23.56% (iOS)	1.7% (WP, RT)
Server (web)	W3Techs	Sep 2014	**36.72%** (Debian,Ubuntu, CentOS,RHEL, Gentoo)	30.18% (AIX,FreeBSD, HP-UX,Solaris, OS X Server)	33.10% (W2K3,W2K8, W2K12)
Supercomputer	TOP500	Nov 2015	**98.8%** (Custom)	1.2%	
Mainframe	Gartner	Dec 2008	28% (SLES, RHEL)	72% (z/OS) UNIX System Services	
Gaming console (excl. Android)	StatCounterGlobal Stats	May 2016	0% (Steam Machine)	68.96% (PlayStation 4,PlayStation 3)	23.91% (Xbox One, Xbox 360)
Embedded	UBM Electronics	Mar 2012	**29.44%** (Android)	4.29% (QNX)	11.65% (WCE 7)

OS as an application platform connecting developers and end users

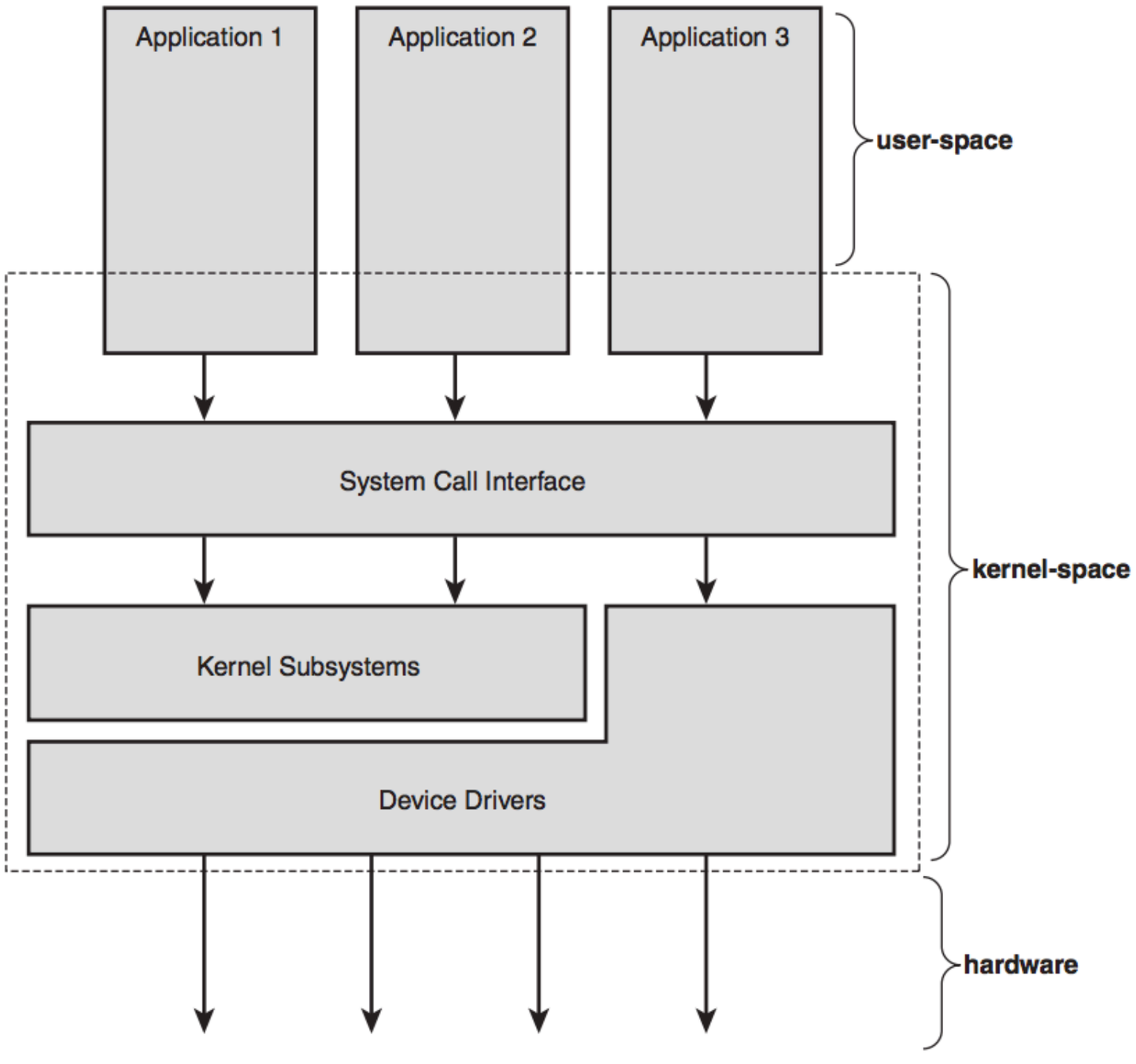
What’s Linux?

- Linux发行版：终端用户
  - Ubuntu
  - Centos
  - Android: EMUI, MIUI
  - …
- Linux系统管理：运维
- Linux应用程序：系统/后台开发
- Linux 内核：内核开发人员



本课程Linux == Linux kernel

Linux系统结构


Application 1
Application 2
Application 3
user-space
System Call Interface
Kernel Subsystems
Device Drivers
kernel-space
hardware

Linux layers

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

https://en.wikipedia.org/wiki/Template:Linux_layers

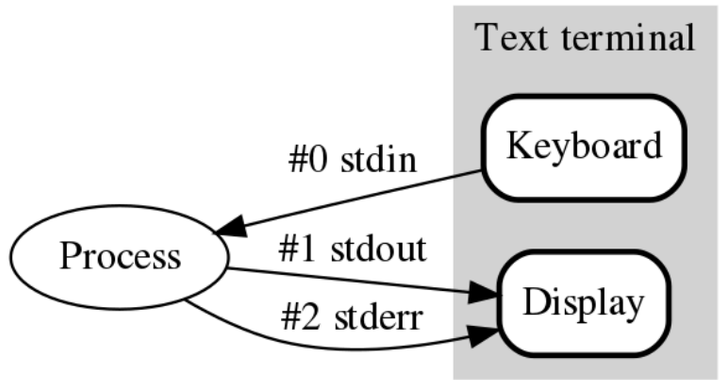
Shell，终端

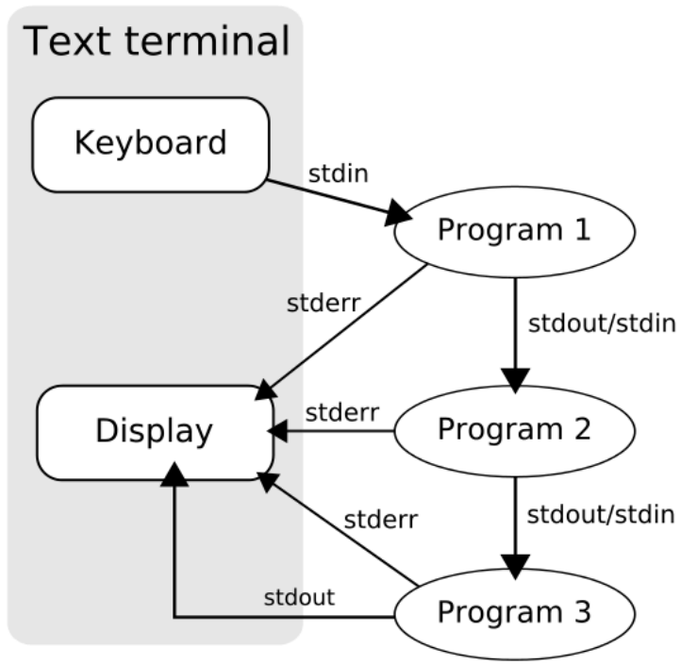
- Linux/Unix 命令行用户界面

```
cd
ls
ls > junk
ls >> junk
ls | wc -l
ls | wc -l > junk
sleep 10&
cat
fg
cat < foo > foo
cat < foo >> foo
cp ../user.txt .
ssh 161220076@210.28.133.11 -p 21317
scp -P 21317 161220076@210.28.133.11:~/users.txt ~
wget http://cs.nju.edu.cn/x.mp4
```

Shell，终端

- 管道pipe
  - process1 | process2 | process3
- IO重定向
  - prog A > fout
  - prog A < fin





Shell，终端

- man: manual, documentation for Linux/Unix systems
  - computer programs (including library and system calls), device files, file format and conventions…
- 脚本语言

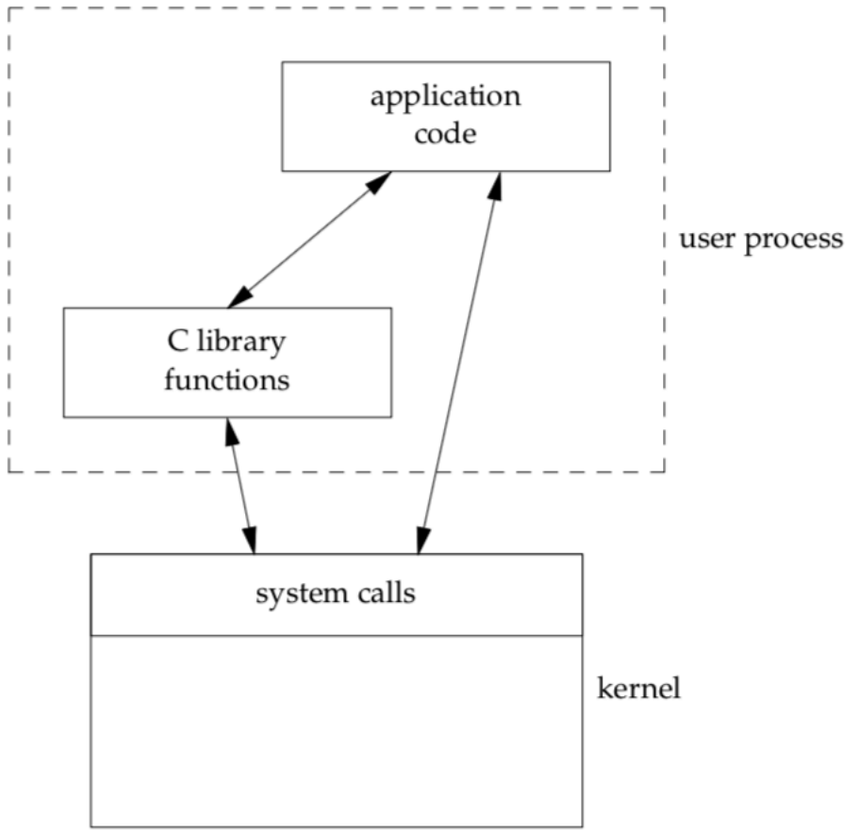
```
#!/bin/bash
for i in $( cat users.txt ); do
    useradd -m -p $i -s /bin/bash  $i
    echo "user $i added successfully!"
done
```

Linux系统的用户视图

<table>
<tr><td rowspan="5">User mode</td><td>User applications</td><td colspan="4">For example, bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, etc.</td></tr>
<tr><td>Low-level system components:</td><td>System daemons:
systemd, runit, logind, networkd, PulseAudio, ...</td><td>Windowing system:
X11, Wayland, SurfaceFlinger(Android)</td><td>Other libraries:
GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep, etc.</td><td>Graphics:
Mesa, AMD Catalyst, ...</td></tr>
<tr><td>C standard library</td><td colspan="4"><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="4"><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="4">Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter
Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="6">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

系统调用与函数库

- 系统调用
  - Linux内核的对外接口
  - 用户程序和内核之间唯一的接口



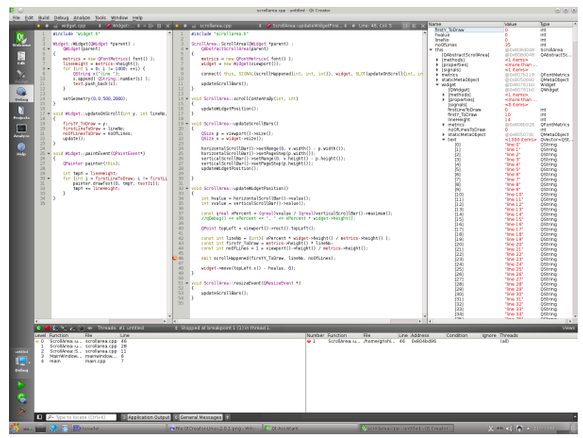
- 函数库
  - 依赖于系统调用
  - 标准函数库建立在系统调用的上层，提供的功能比系统调用强，使用也比较方便
    - 静态库(.a文件)
    - 动态库/共享库 (.so文件)

Linux支持的编程语言

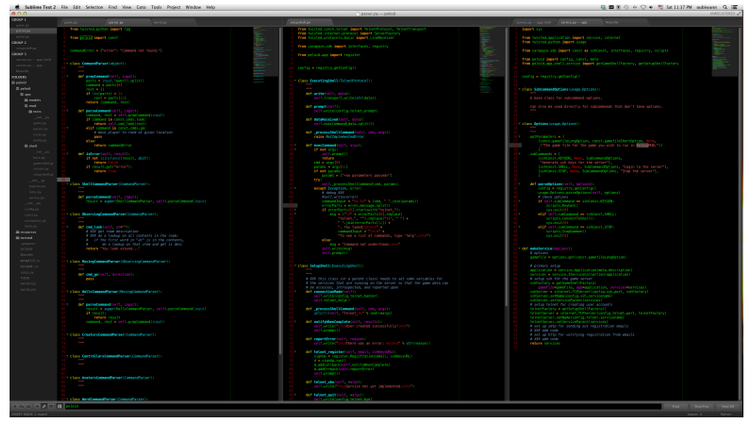
- 高级编程语言
  - C/C++, Java, Fortran…
  - ELF二进制格式
    - Executable and Linkable Format
    - 工具接口标准委员会(TIS)选择ELF体系作为不同操作系统之间可移植的二进制文件格式
- 脚本
  - Shell: sh/bash, csh, ksh
  - Perl, Python, tcl/tk, sed, awk…

集成环境

- 集成开发环境IDE
  - Emacs/xemacs
  - Qt creator: visual debugger
  - Eclipse
  - Kdevelop



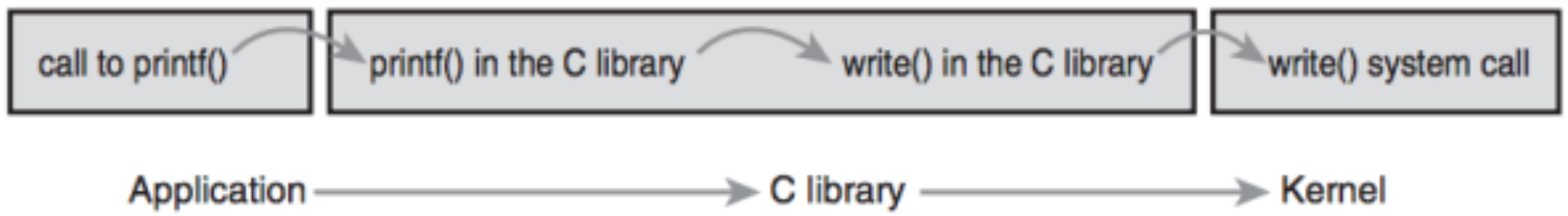
- 命令行开发环境
  - 编辑器
    - Sublime Text, vi/vim/gvim, emacs/xemacs, pico
  - 代码阅读器
    - Sublime Text, source navigator， vi/emacs+ ctags/etags
  - 配置工具
    - automake, autoconf, …



GNU Toolchain

- gcc
  - GNU C编译器
- gdb
  - GNU调试工具
  - gdb调试命令
- 二进制代码工具
  - as, ld, ar, ldd…
- Make
- Git, CVS
  - 版本控制

系统调用（syscall）



- 系统调用：获得OS服务的主要方式

- Linux 系统有300+个系统调用

- /bin/ls 使用了什么系统调用？ syscall追踪工具：

    - Linux： strace /bin/ls

    - OSX: sudo dtruss /bin/ls

系统调用举例

- pid_t getpid(void);
  - get process id
- pid_t getppid(void);
  - get parent process id
- char *getwd(char *buf);
  - get current working directory

系统调用举例

- 创建进程 pid_t fork(void);
  - 执行一次，返回两次
  - 父进程：返回子进程PID；子进程：返回0；

```
int main(int argc, char *argv[]) {
    printf("Greetings from process %d! (parent %d)\n",
getpid(),getppid());
    fork();
    printf("Bye-bye from process %d! (parent %d)\n”,
getpid(),getppid());
    return 0;
}
```

```
Greetings from process 31384! (parent 27623)
Bye-bye from parent process 31384! (parent 27623)
Bye-bye from child process 31385! (parent 31384)
```

系统调用举例

- 等待进程结束 pid_t waitpid(pid_t pid, int *status, int options);

```c
int main(int argc, char *argv[])
{
    pid_t pid = fork();

    if (pid == -1) {
        perror("fork failed");
        exit(-1);
    }
    else if (pid == 0) {
        printf("Hello from the child process!\n");
        exit(0);
    }
    else {
        int status;
        waitpid(pid, &status, 0);
    }
    return 0;
}
```

系统调用举例

- int execvp(char const *file, char const *argv[]);

```
int main( void ) {
   char *argv[3] = {"Command-line", ".", NULL};

   int pid = fork();

   if ( pid == 0 ) {
      execvp( "find", argv );
   }

   /* Put the parent to sleep for 2 seconds—let the child
finished executing */
   wait( 2 );

   printf( "Finished executing the parent process\n” );

   return 0;
}
```

系统调用举例

- int execlp(const char *file, const char *arg, ...);

```
int main(void)
{
    char    buf[MAXLINE]; /* from apue.h */
    pid_t   pid;
    int     status;
    printf("%% ");  /* print prompt (printf requires %% to print %) */
    while (fgets(buf, MAXLINE, stdin) != NULL) {
        if (buf[strlen(buf) - 1] == '\n')
            buf[strlen(buf) - 1] = 0; /* replace newline with null */
        if ((pid = fork()) < 0) {
            err_sys("fork error");
        } else if (pid == 0) {      /* child */
            execlp(buf, buf, (char *)0);
            err_ret("couldn't execute: %s", buf);
            exit(127);
        }

        /* parent */
        if ((pid = waitpid(pid, &status, 0)) < 0)
            err_sys("waitpid error");
        printf("%% ");
    }

    exit(0);

}
```

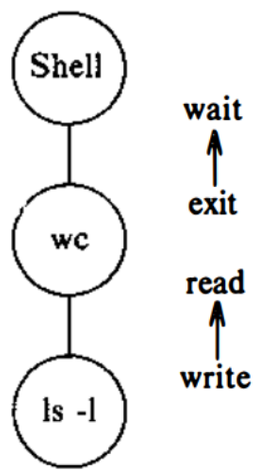
系统调用举例

- 创建管道 int pipe(int fd[2]);
  - fd[0]可读，fd[1]可写

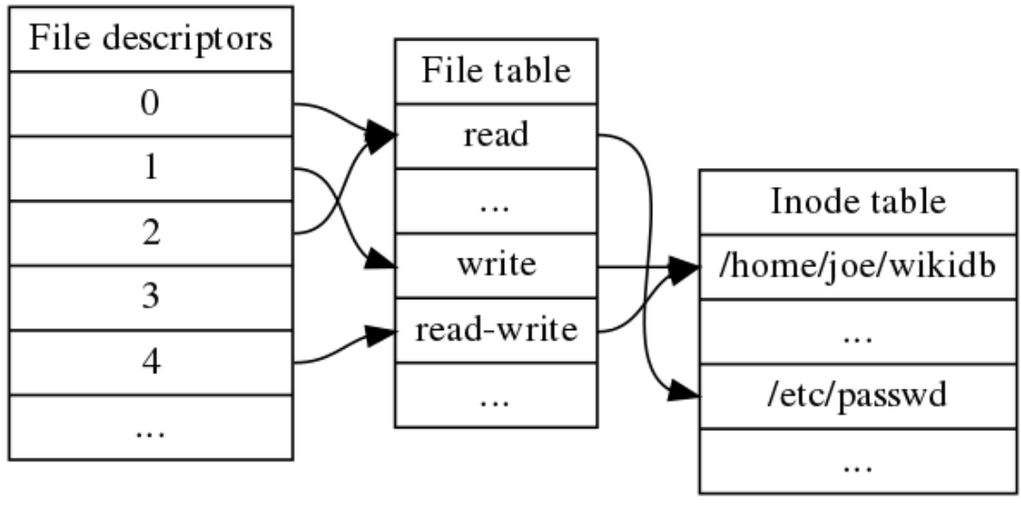
```c
int main(int argc, char *argv[]) {
    int fds[2];
    pipe(fds);

    pid_t pid = fork();
    if (pid == 0) {
        char buffer[6];
        read(fds[0], buffer, 6);
        printf("%s\n", buffer);
        return 0;
    }

    write(fds[1], "hello", 6);
    return 0;
}
```

管道，文件描述符

- ls -l | wc
- 每个进程都有自己的文件描述符表
  - 键盘stdin: 0
  - 屏幕stdout:1
  - 屏幕stderr: 2
- 每个进程还有一个文件表
  - 表项记录文件打开的模式
    - 只读，只写，读写
  - 表项记录当前offset
- fork之后父子进程

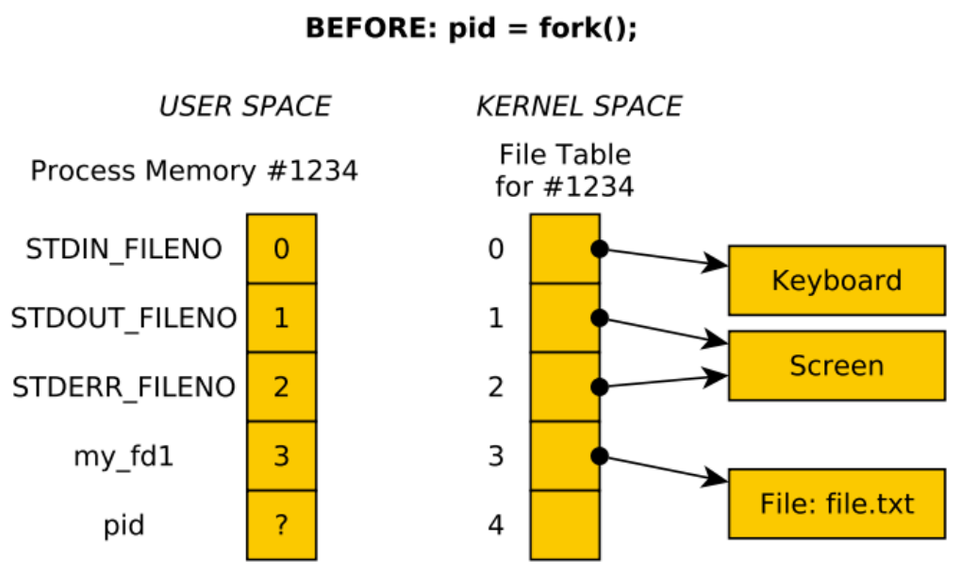


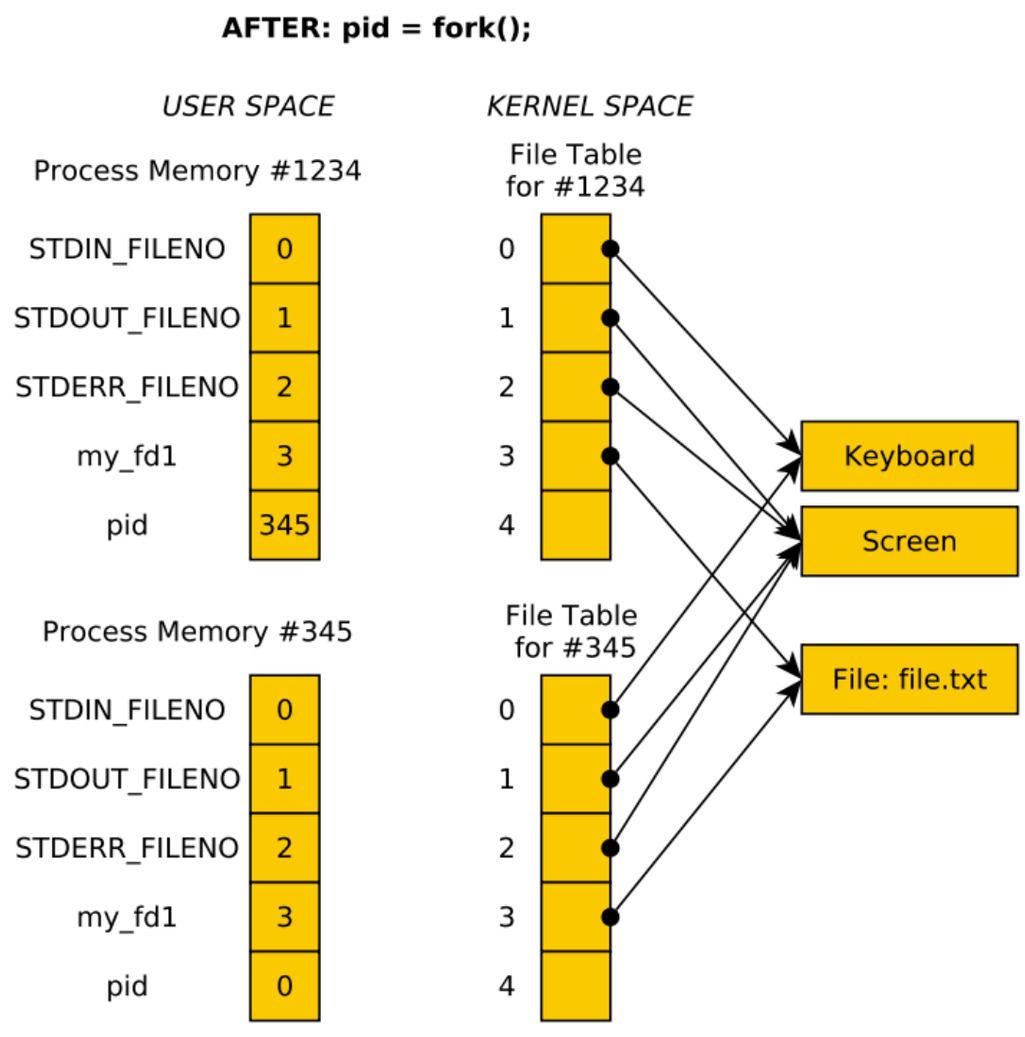
Relationship of Processes for ls −l | wc



管道，文件描述符

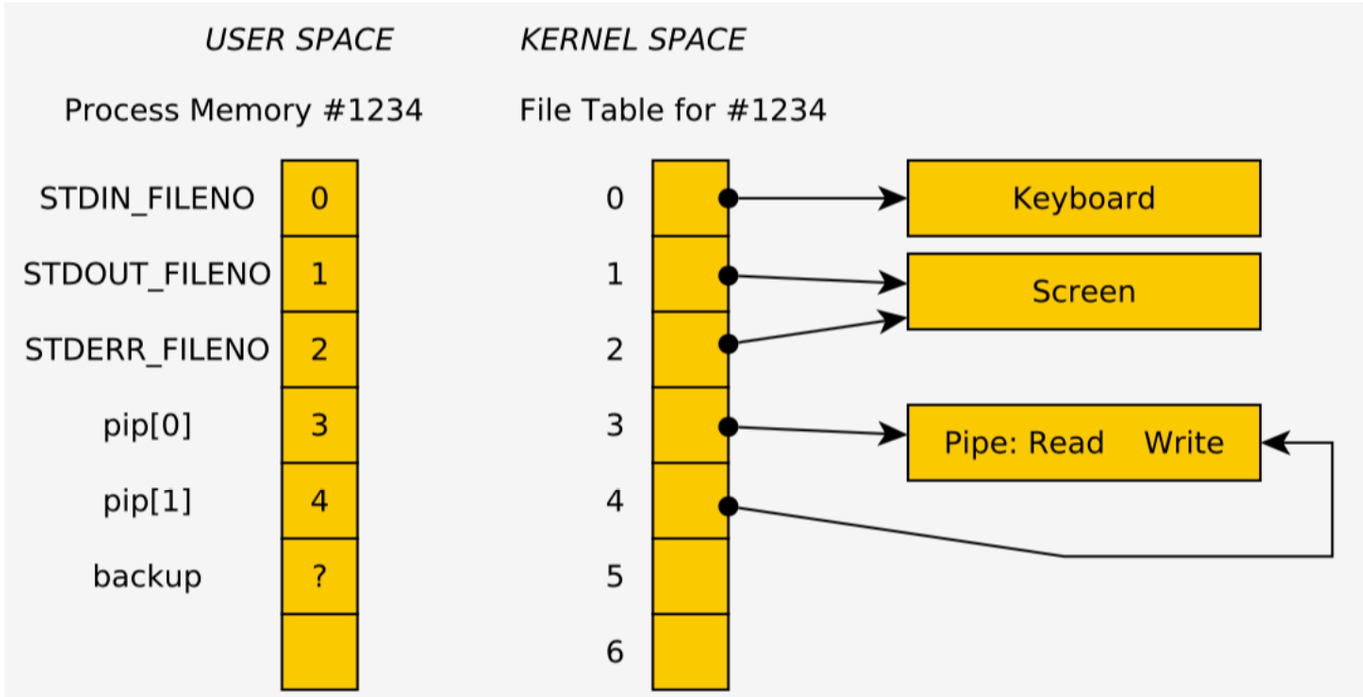
- pid=fork();





管道，文件描述符

- int pip[2];
- pipe(pip);



系统调用举例

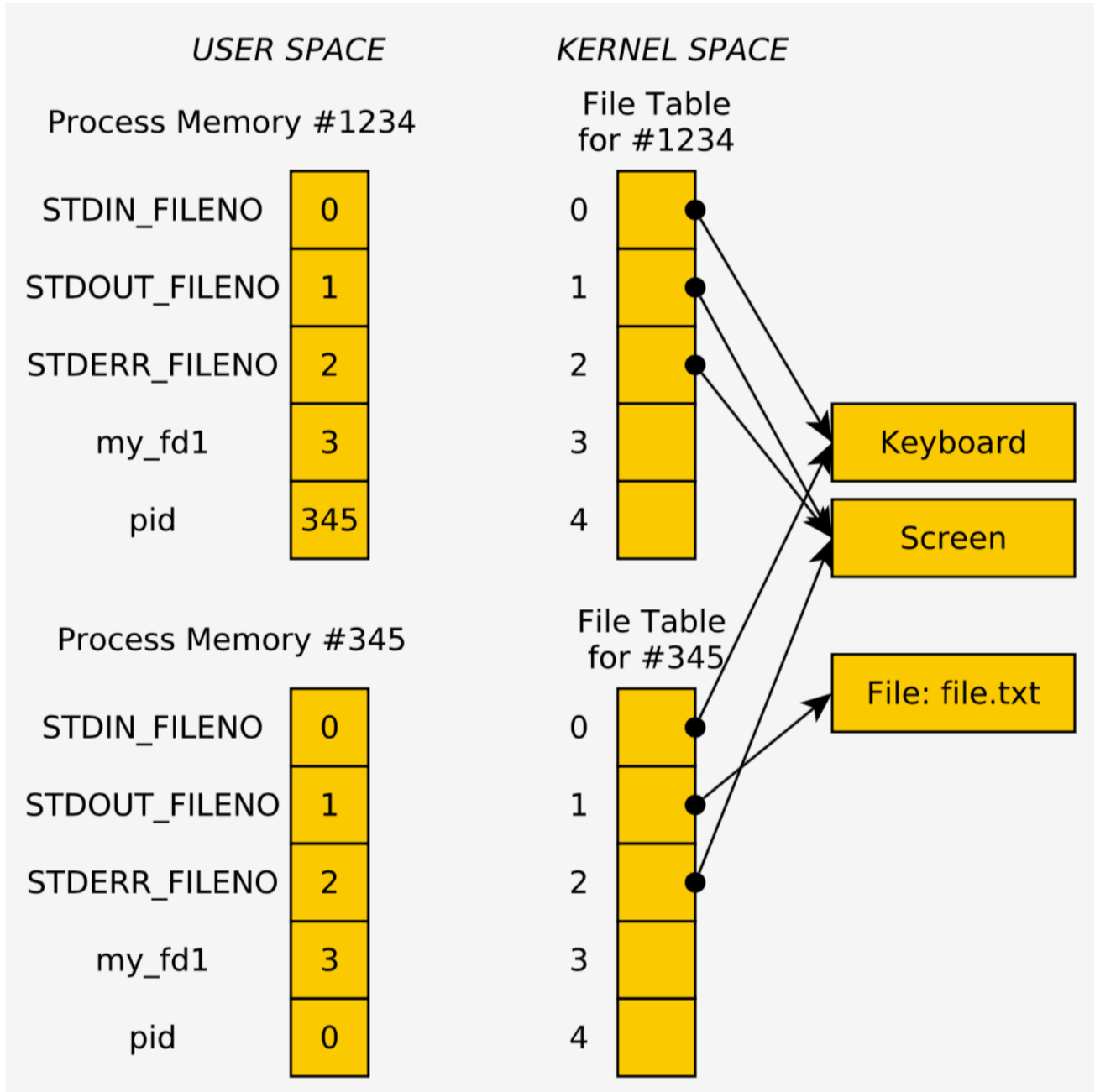
- int dup (int oldfd);
- int dup2 (int oldfd, int newfd);

```
int main(int argc, char *argv[]) {
    int fd= open(“myfile.txt”);
    close(1);
    dup(fd); //fd, 1等价
    printf("This will print in myfile.txt\n”);
    return 0;
}
```

```
int main(int argc, char *argv[]) {
    int fd= open(“myfile.txt”);
    dup2(fd,1); //fd, 1等价
    printf("This will print in myfile.txt\n”);
    return 0;
}
```

dup

- dup2(my_fd1,1);



pipe fork dup

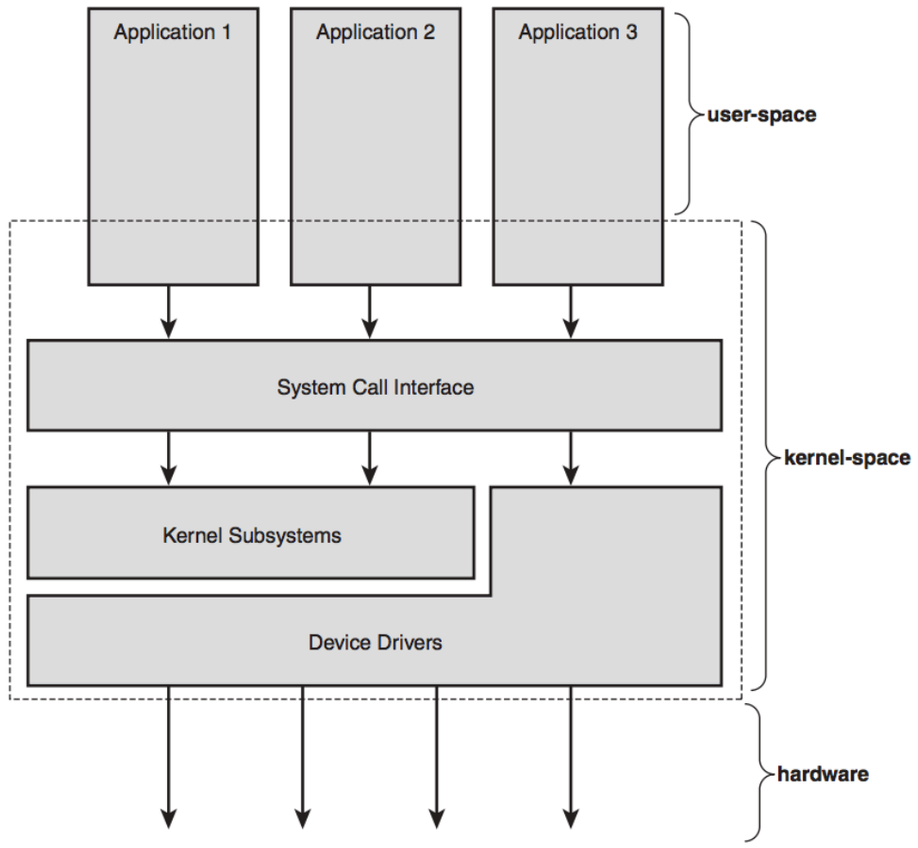
- int pip[2];
- pipe(pip);
- pid=fork();
- pid>0; //parent
  - dup2(pip[1],1);
- pid==0; //child
  - dup2(pip[0],0);

Linux内核

<table>
<tr><td rowspan="4">User mode</td><td>User applications</td><td colspan="4">For example, bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, etc.</td></tr>
<tr><td>Low-level system components:</td><td>System daemons:
systemd, runit, logind, networkd, PulseAudio, ...</td><td>Windowing system:
X11, Wayland, SurfaceFlinger(Android)</td><td>Other libraries:
GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep, etc.</td><td>Graphics:
Mesa, AMD Catalyst, ...</td></tr>
<tr><td>C standard library</td><td colspan="4"><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 colspan="2"></td></tr>
<tr><td rowspan="3">Kernel mode</td><td rowspan="3">Linux kernel</td><td colspan="4"><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="4">Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter
Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="6">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

划分用户态/内核态的必要性

- 不区分的缺陷（Microsoft DOS）
  - 用户直接修改操作系统数据
  - 用户直接调用操作系统内部函数
  - 用户直接操作外设
  - 用户任意读/写物理内存
- 区分意义：保护内核
  - 禁止用户程序和底层硬件直接打交道
    - 如果用户程序往硬件控制寄存器写入不恰当的值，可能导致硬件无法正常工作
  - 禁止用户程序访问任意物理内存，否则可能会破坏其他程序的正常执行
    - 如果对核心内核所在的地址空间写入数据，会导致系统崩溃



CPU对用户态/内核态划分的支持

- 硬件支持：现代CPU都提供不同指令执行级别
  - 在高执行级别状态下，代码可以执行特权指令，访问任意的物理地址，此时CPU执行级别就对应着内核态
  - 在低级别执行状态下，代码的掌控范围会受到限制，只能在对应级别允许的范围内活动
- 举例
  - intel x86 CPU有四种不同执行级别：0 ~ 3
  - Linux只使用0级和3级，分别表示内核态和用户态

用户态/核心态的区分方法

- CS寄存器最低两位表明当前代码的特权级
  - CPU每条指令的读取都是通过cs:eip这两个寄存器
    - cs：代码段选择寄存器
    - eip：偏移量寄存器
  - 上述判断由硬件完成
- Linux的逻辑地址空间特权等级划分标志
  - 0xc0000000以上地址空间：只能在内核态下访问
  - 0x00000000~0xbfffffff的地址空间：两种状态下都可访问

用户态/核心态的区分方法

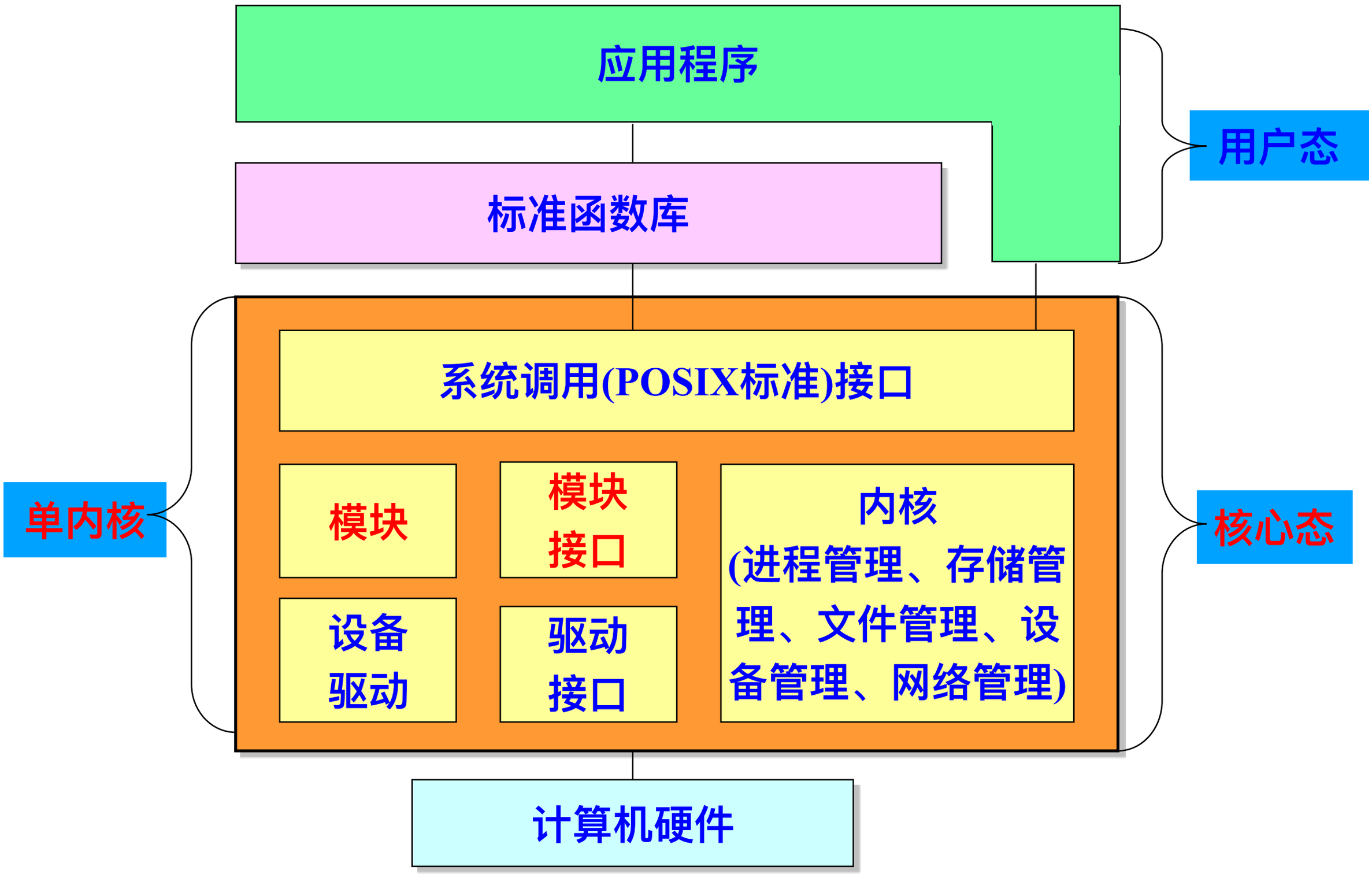
- ps命令，方括号内的都是内核线程

```
root@localhost:~# ps fax
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:02 [kthreadd]
    4 ?        S<     0:00  \_ [kworker/0:0H]
    6 ?        S<     0:00  \_ [mm_percpu_wq]
    7 ?        S      0:00  \_ [ksoftirqd/0]
    8 ?        S     24:39  \_ [rcu_sched]
    9 ?        S      0:00  \_ [rcu_bh]
   10 ?        S      0:00  \_ [migration/0]
   11 ?        S      0:37  \_ [watchdog/0]
   12 ?        S      0:00  \_ [cpuhp/0]
   13 ?        S      0:00  \_ [cpuhp/1]
   14 ?        S      0:39  \_ [watchdog/1]
   15 ?        S      0:00  \_ [migration/1]
   16 ?        S      5:33  \_ [ksoftirqd/1]
   18 ?        S<     0:00  \_ [kworker/1:0H]
   19 ?        S      0:00  \_ [cpuhp/2]
   20 ?        S      0:40  \_ [watchdog/2]
   21 ?        S      0:00  \_ [migration/2]
   22 ?        S      0:00  \_ [ksoftirqd/2]
   24 ?        S<     0:00  \_ [kworker/2:0H]
   25 ?        S      0:00  \_ [cpuhp/3]
   26 ?        S      0:29  \_ [watchdog/3]
```

Linux的内核特点

- Linux是单内核、多模块系统
  - Linux内核运行在单独的内核地址空间
  - 所有操作系统功能作为一个模块实现在内核中
    - 模块均运行在内核态，直接调用函数，无需消息传递
    - 模块化设计、抢占式 (Linux 2.6内核级抢占，Linux 2.4用户级抢占)、支持内核线程及动态装载内核模块的能力
    - 与Unix主要区别
      - Unix也是单内核系统，但Linux汲取了微内核设计思想（基于模块定制内核）
- Unix也是单内核系统
- Windows NT和Mach是微内核系统
  - 只提供基础功能，其他功能通过服务实现
  - 微内核被划分为多个独立过程，每个过程称为服务器

Linux单内核结构


应用程序
标准函数库
用户态
系统调用(POSIX标准)接口
单内核
模块
模块
接口
内核
(进程管理、存储管理、文件管理、设备管理、网络管理)
设备
驱动
驱动
接口
核心态
计算机硬件

Linux内核组成

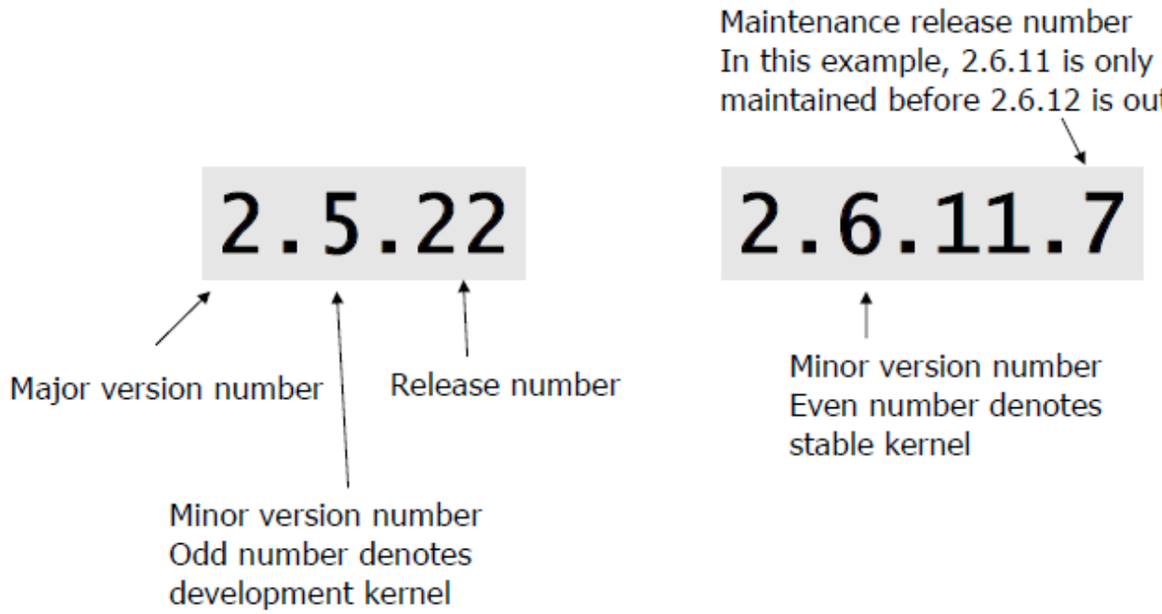
- 进程调度程序：负责控制进程访问CPU kernel/
- 进程间通信：为进程之间的通信提供实现机制 ipc/
- 内核管理程序：支持虚拟内存及多进程安全共享主存系统 mm/
- 虚拟文件系统：抽象异构硬件设备细节，提供公共文件接口 fs/
- 网络接口：提供对多种组网标准和网络硬件的访问 /net

Linux内核源码的获取

- 下载位置
  - http://www.kernel.org: 以GNU zip和bzip2形式发布
  - https://github.com/torvalds/linux
- 安装位置
  - 一般安装在/usr/src/linux，不要将该源码树用于开发
  - 在编译自己编写的C库所用的内核版本要链接到该树
  - 不要以root身份对内核进行修改，应先建立自己的主目录，仅以root身份安装新内核
  - 安装新内核应该保持/usr/src/linux原封不动
- 补丁
  - patch -p1< ../patch-x.y.z

Linux内核版本

- Linux内核版本号由Linux开发小组（Linus Torvalds总协调）确定
- Linux内核采用双树系统
  - 一棵是稳定树，主要用于发行
  - 另一棵是非稳定树（开发树）
    用于产品开发和改进
- Linux内核版本号由3/4位数字组成



Android版本

Code name	Version	Linux kernel version[1]	Initial release date	API level
(No codename)[2]	1.0	?	September 23, 2008	1
Petit Four[2]	1.1	2.6.X	February 9, 2009	2
Cupcake	1.5	2.6.27	April 27, 2009	3
Donut[3]	1.6	2.6.29	September 15, 2009	4
Eclair[4]	2.0 – 2.1	2.6.29	October 26, 2009	5 – 7
Froyo[5]	2.2 – 2.2.3	2.6.32	May 20, 2010	8
Gingerbread[6]	2.3 – 2.3.7	2.6.35	December 6, 2010	9 – 10
Honeycomb[7]	3.0 – 3.2.6	2.6.36	February 22, 2011	11 – 13
Ice Cream Sandwich[8]	4.0 – 4.0.4	3.0.1	October 18, 2011	14 – 15
Jelly Bean[9]	4.1 – 4.3.1	3.0.31 to 3.4.39	July 9, 2012	16 – 18
KitKat[10]	4.4 – 4.4.4	3.10	October 31, 2013	19 – 20
Lollipop[11]	5.0 – 5.1.1	3.16.1	November 12, 2014	21 – 22[12]
Marshmallow[13]	6.0 – 6.0.1	3.18.10	October 5, 2015	23
Nougat[14]	7.0 – 7.1.2	4.4.1	August 22, 2016	24 – 25
Oreo[15]	8.0 – 8.1	4.10	August 21, 2017	26 – 27
Pie[16]	**9.0**	4.4.107, 4.9.84, and 4.14.42	August 6, 2018	28

Linux内核源码结构

- 25 million lines of code
- Every file has a maintainer

```
SCHEDULER
M:   Ingo Molnar <mingo@redhat.com>
M:   Peter Zijlstra <peterz@infradead.org>
L:   linux-kernel@vger.kernel.org
T:   git git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git sched/core
S:   Maintained
F:   kernel/sched/
F:   include/linux/sched.h
F:   include/uapi/linux/sched.h
F:   include/linux/wait.h
```

Directory	Description
arch	Architecture-specific source
block	Block I/O layer
crypto	Crypto API
Documentation	Kernel source documentation
drivers	Device drivers
firmware	Device firmware needed to use certain drivers
fs	The VFS and the individual filesystems
include	Kernel headers
init	Kernel boot and initialization
ipc	Interprocess communication code
kernel	Core subsystems, such as the scheduler
lib	Helper routines
mm	Memory management subsystem and the VM
net	Networking subsystem
samples	Sample, demonstrative code
scripts	Scripts used to build the kernel
security	Linux Security Module
sound	Sound subsystem
usr	Early user-space code (called initramfs)
tools	Tools helpful for developing Linux
virt	Virtualization infrastructure

Linux内核编译

- 内核编译主要工具文件

文件类型	作用
Makefile	顶层**Makefile**文件
.config	内核配置文件
arch/$(ARCH)/Makefile	机器体系相关**Makefile**文件
scripts/Makefile.*	所有内核**Makefiles**共用规则
kbuild Makefiles	其它**Makefile**文件

- 内核编译后，会在/boot目录生产以下文件
  - bzImage文件：内核启动文件
  - initrd-x.y.z.img文件：用于临时引导硬件到实际内核vmlinuz能够接管并继续引导状态（用于支持内存镜像）
  - System.map文件：系统正常启动时不会读这个符号表；主要是为了内核引导出错时便于调试

Linux内核配置系统

- Makefile
  - 定义编译链接规则、位于linux源代码各目录
- 配置文件(config.in或kconfig)
  - 提供内核的配置选择和设置
- 配置工具
  - 文本命令行工具：make config
  - 基于ncurse的图形工具：make menuconfig
  - 基于X11的图形工具：make xconfig
  - 基于gtk+的图形工具：make gconfig
  - 创建默认配置：make defconfig
  - 配置工具输出文件
    - .config文件：用#include包括到主Makefile中
    - include/linux/autoconf.h：用#include包括到各个.c文件
      - 每个.c文件都有<#include/linux/autoconf.h>代码项

主Makefile功能

- 采用GNU编译工具对.config中的源文件列表编译
  - 完成内核文件的配置、依赖关系及模块的生成，随后调用Rules.make编译文件
    - Rules.make定义所有Makefile共用的编译规则
  - Makefile支持的make命令
    - make mrproper：检查.o文件及文件依赖关系的正确性
    - make config：配置内核并生成配置文件
    - make dep：根据配置文件创建相应的依赖关系树
    - make clean：清除旧版本的目标文件
    - make zImage：编译并用gzip压缩成1MB以下的内核
      - 未压缩的文件是vmlinuz
    - make bzImage：编译并用gzip压缩成1MB以上的内核
    - make modules：编译模块
    - make modules_install：安装模块
    - depmod –a：生成模块之间的依赖关系

Linux内核的编译、安装

- 配置内核
  - 获取默认.config文件：cp /boot/config-`uname -r` .config
  - 生成配置文件：make config
- 编译
  - 编译内核：make -jn
  - 编译模块：make modules
- 安装内核
  - 安装模块：make modules_install
  - 生成模块依赖关系：depmod –a
  - 安装内核：make install

/vmlinux: kernel in hard disk

- statically linked executable file that contains the Linux kernel
- /unix: kernel file for unix
- vmlinux or vmlinuz
  - vm for virtual memory
  - z for compressed (gzipped)
- 位置
  - /boot/vmlinux-x.y.z
  - /vmlinux-x.y.z

Linux 内核编程

- 内核编程与系统编程有很大不同

- The kernel has access to neither the C library nor the standard C headers.

- no printf(), string function…

- printk(“Hello world! A string '%s' and an integer '%d'\n", str, i);

- printk(KERN_ERR "this is an error!\n");

Linux 内核编程

- 使用GNU C，不是ANSI C
  - if(unlikely(error)){ /*.....*/ }
  - if(likely(error)){ /*.....*/ }
- 尽量避免使用浮点数
- 栈大小固定：4KB/8KB
- 抢占式内核，需考虑同步并发

课程内容组织

- Linux简介: lab1 实现一个简单的shell
- 进程管理：lab2 增加系统调用，实现ps功能
- 进程调度：lab3 新调度策略，提升每个用户/APP首次运行程序优先级
- 进程同步：lab4 一组基于加速度的同步原语
- 虚拟内存：lab5 获取每个用户物理页面集合
- 文件系统：lab6 实现一个地理标记文件系统
- lab2-lab6 在安卓模拟器或安卓手机上运行，可组队（1-3人）

实验题 Lab1

一. 简单Shell

- 内置命令：cd，exit，pwd
- 管道：至少支持两个进程
- IO重定向：至少1个进程
- 后台运行

二. 内核编译

- 在ubuntu环境下，下载内核源码，编译并生成自定义的内核
  - 安装编译好的自定义内核
  - 参考https://kernelnewbies.org/KernelBuild

实验题 Lab1

- Linux系统
  - 自己机器/乙126机房
    - 虚拟机：Windows VMware Workstation, Mac VMware Fusion
- 鼓励交流
- 不能抄袭
  - 第一次警告并扣除当次作业分数
  - 再犯。。

谢谢大家！

下次课：基础实验楼乙126