1. Linux内核概述

授课教师：游伟 副教授

授课时间：周二14:00 – 15:30（立德楼807）

课程主页：https://www.youwei.site/course/kernel

引子：HelloWorld程序的运行

每个C语言程序设计初学者，入门的第一个程序HelloWorld，其源代码如下：

```
1.   #include <stdio.h>

2.  int main(int argc, char *argv[])
3.  {
4.      puts("hello world");
5.      return 0;
6.  }
```

问题：HelloWorld程序运行的过程中，操作系统都做了些什么？

1. 用户告诉操作系统执行hello程序
2. 操作系统找到该程序，检查其类型
3. 检查程序首部，找出正文和数据的地址
4. 文件系统找到第一个磁盘块
5. 父进程需要创建一个新的子进程，执行hello程序
6. 操作系统需要将执行文件映射到进程结构
7. 操作系统设置CPU上下文环境，并跳到程序开始处
8. 程序的第一条指令执行，失败，缺页中断发生
9. 操作系统分配一页内存，并将代码从磁盘读入，继续执行
10. 更多的缺页中断，读入更多的页面
11. 程序执行系统调用，在文件描述符中写一字符串
12. 操作系统检查字符串的位置是否正确
13. 操作系统找到字符串被送往的设备
14. 设备是一个伪终端，由一个进程控制
15. 操作系统将字符串送给该进程
16. 该进程告诉窗口系统它要显示字符串
17. 窗口系统确定操作合法性，然后将字符串转换成像素
18. 窗口系统将像素写入存储映像区
19. 视频硬件将像素表示转换成一组模拟信号控制显示器
20. 显示器发射电子束
21. 你在屏幕上看到hello world

目录

1. 操作系统概述
2. 内核概述
3. Linux内核源码
4. Linux内核实验环境

1.1 操作系统概述

- 定义：一组主管并控制计算机操作、协调和运用硬软件资源、提供公共服务来组织用户交互的相互关联的系统软件程序
- 主题：虚拟化 + 并发 + 持久性 + 提供服务
  - 虚拟化：涉及进程管理与调度、内存管理与进程地址空间
  - 并发：涉及内核同步、时间管理
  - 持久性：涉及文件系统、磁盘管理
  - 提供服务：涉及中断、异常、系统调用

1.1.1 虚拟化

- 虚拟化CPU：每个进程拥有一个独立的虚拟CPU

```c
// File: chapter1/Virtualization/cpu.c

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/time.h>

double GetTime()
{
  struct timeval t;
  double time;
  int rc = gettimeofday(&t, NULL);
  assert(rc == 0);
  time = (double)(t.tv_sec) +
         (double)(t.tv_usec) / 1e6;
  return time;
}

void Spin(int howlong)
{
  double t = GetTime();
  while ((GetTime() - t) < (double)(howlong));
}

int main(int argc, char **argv)
{
  if (argc != 2)
  {
    fprintf(stderr, "usage: cpu <string>\n");
    exit(1);
  }

  char *str = argv[1];
  while (1)
  {
    Spin(1);
    printf("%s\n", str);
  }
}
```

```
GuestOS > cd /tmp/share/chapter1/Virtualization
GuestOS> gcc -o cpu cpu.c
GuestOS > ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 207
[2] 208
[3] 209
[4] 210
C
B
D
A
B
C
...
```

1.1.1 虚拟化

- 虚拟化内存：每个进程拥有一个独立的虚拟地址空间

```c
// File: chapter1/Virtualization/mem.c

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/time.h>

double GetTime()
{
  struct timeval t;
  double time;
  int rc = gettimeofday(&t, NULL);
  assert(rc == 0);
  time = (double)(t.tv_sec) +
         (double)(t.tv_usec) / 1e6;
  return time;
}

void Spin(int howlong)
{
  double t = GetTime();
  while ((GetTime() - t) < (double)(howlong));
}

int main(int argc, char **argv)
{
  int *p = malloc(sizeof(int));
  assert(p != NULL);
  printf("(%d) memory address of p: %08x\n",
         getpid(), (unsigned)(p));
  *p = 0;
  while (1)
  {
    Spin(1);
    *p = *p + 1;
    printf("(%d) p: %d\n", getpid(), *p);
  }
}
```

```
GuestOS > cd /tmp/share/chapter1/Virtualization
GuestOS > gcc -o mem mem.c
GuestOS > echo 0 > /proc/sys/kernel/randomize_va_space
GuestOS > ./mem & ./mem &
[1] 213
[2] 214
(214) Memory address of p: 00405160
(213) Memory address of p: 00405160
(214) p: 1
(213) p: 1
(213) p: 2
(214) p: 2
(213) p: 3
...
```

1.1.2 并发

- 逻辑上的并行：多个进程/线程分时共享计算资源

```c
// File: chapter1/Concurrency/thread.c

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

volatile int counter = 0;
int loops;

void * worker(void *arg){
  int i;
  for (i = 0; i < loops; i++) counter++;
  return NULL;
}
```

```c
int main(int argc, char **argv)
{
  pthread_t p1, p2;

  if (argc != 2)
  {
    fprintf(stderr, "usage: threads <value>\n");
    exit(1);
  }

  loops = atoi(argv[1]);
  printf("Initial value: %d\n", counter);

  pthread_create(&p1, NULL, worker, NULL);
  pthread_create(&p2, NULL, worker, NULL);
  pthread_join(p1, NULL);
  pthread_join(p2, NULL);

  printf("Final value: %d\n", counter);
}
```

```
GuestOS > cd /tmp/share/chapter1/Concurrency
GuestOS > gcc -o thread thread.c -lpthread
GuestOS > ./thread 1000
Initial value: 0
Final value: 2000
prompt> ./thread 100000
Initial value: 0
Final value: 112448
prompt> ./thread 100000
Initial value: 0
Final value: 138413
```

1.1.3 持久性

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

int main(int argc, char **argv)
{
  int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
  assert(fd > -1);
  int rc = write(fd, "hello world\n", 13);
  assert(rc == 13);
  close(fd);
  return 0;
}
```

1.1.4 提供服务

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>

unsigned int counter;

void sig_handler()
{
  counter++;
  printf("current counter: %d, current value: %d\n", counter, 1/counter);
}

int main(int argc, char **argv)
{
  scanf("%d", &counter);
  signal(SIGALRM, sig_handler);
  while (1)
  {
    alarm(1);
    sleep(2);
  }
}
```

系统调用，系统服务例程响应

可能触发异常

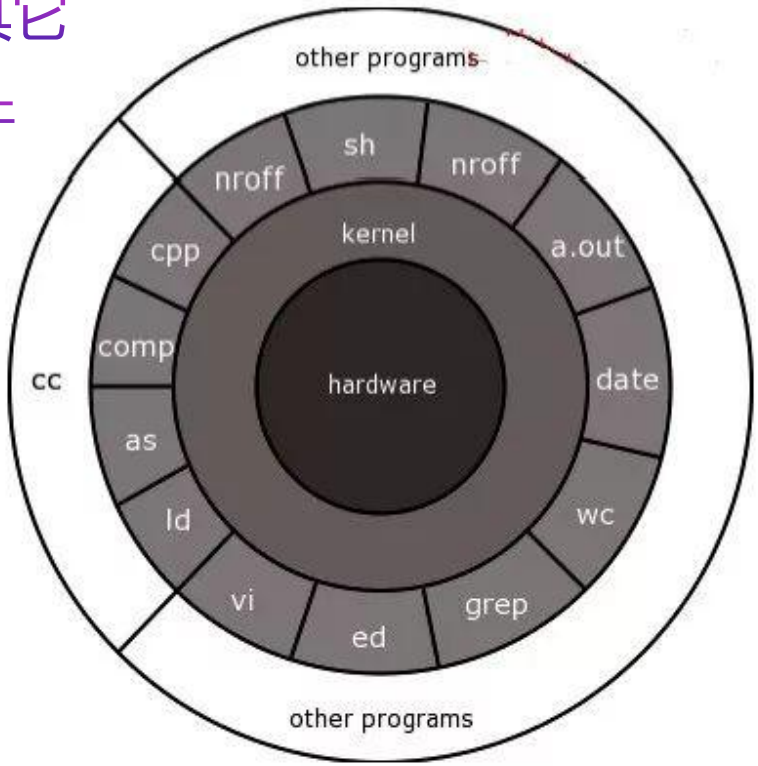
I/O中断，中断处理例程响应

时钟中断，中断处理例程响应

1.2 内核概述

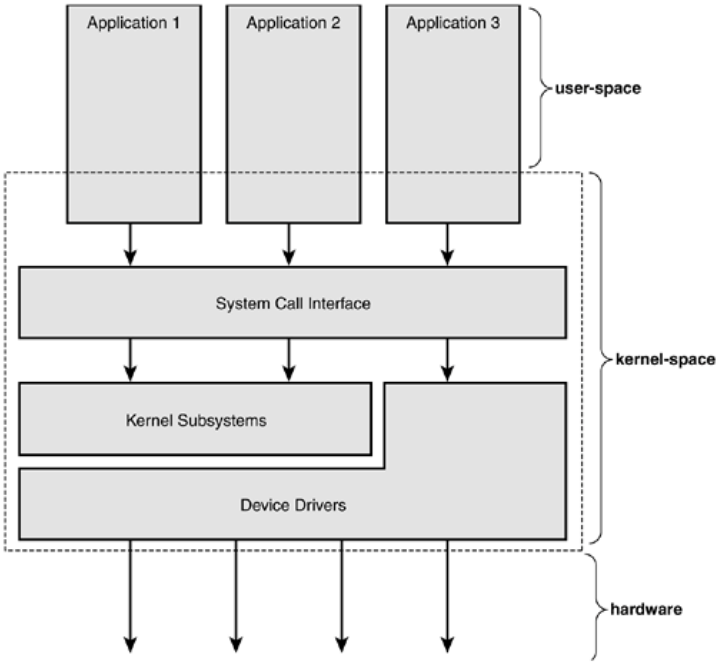
- 操作系统：整个系统中负责完成最基本功能和系统管理的那些部分，包括内核、设备驱动程序、启动引导程序、命令行Shell或其它类型用户界面、基本文件管理工具、系统工具……

- 内核：操作系统的内在核心，系统的其它部分必须依赖其提供服务，例如管理硬件设备和分配系统资源，包括调度程序、设备驱动程序、进程间通信、内存管理、文件系统、程序装载……



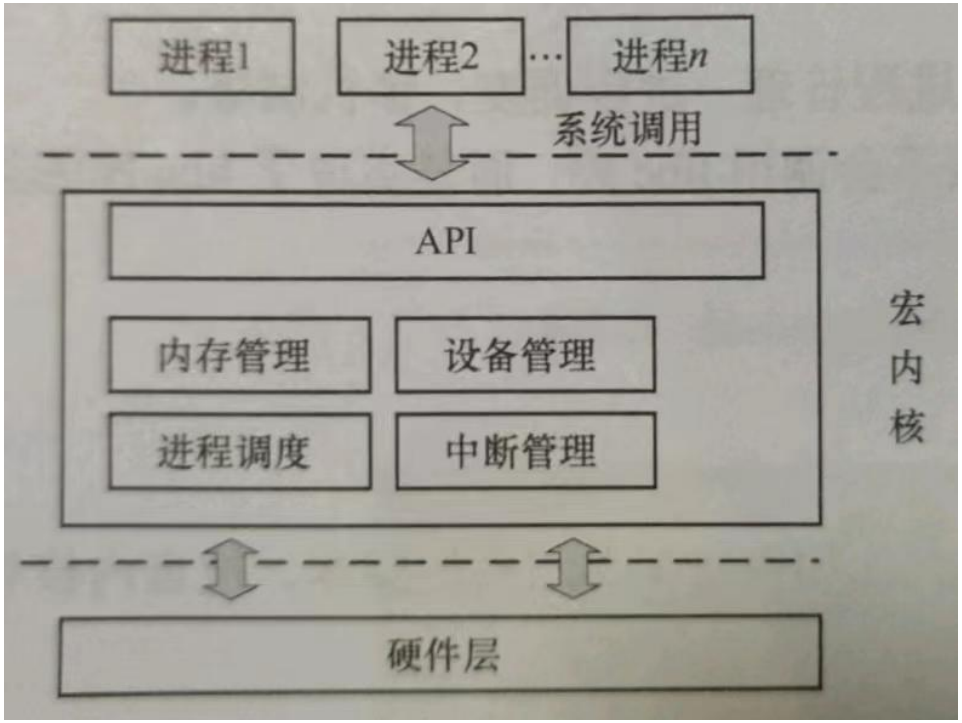
1.2.1 系统态与用户态

- 内核运行于**系统态**，有受保护的内存空间（内核空间，Kernel Space）和访问硬件的所有权限
- 应用程序运行于**用户态**，只被允许访问其所使用到的内存空间（用户空间，User Space）和部分系统资源，不能直接访问硬件



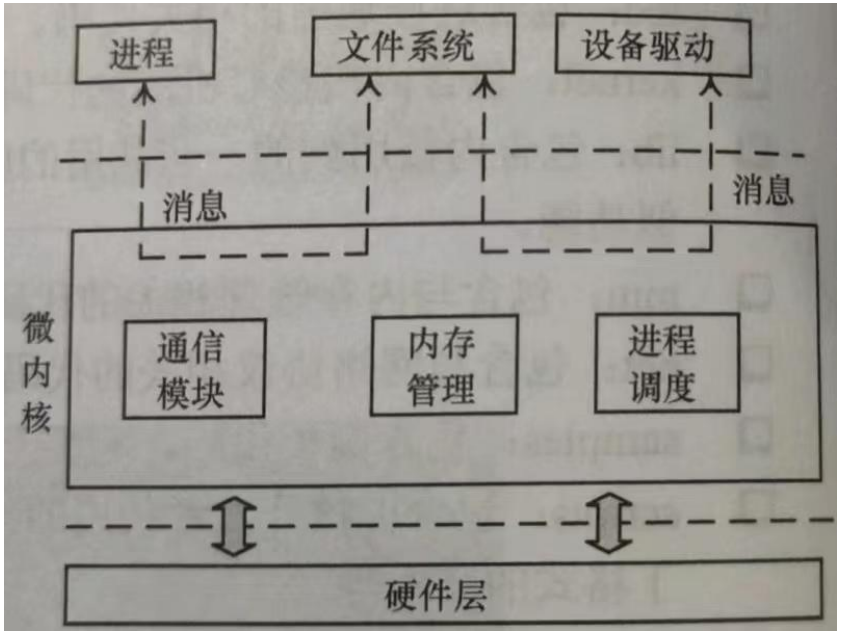
1.2.2 宏内核与微内核

- 宏内核：所有的内核代码都被编译成二进制文件，所有的内核代码都运行在一个大的内核地址空间里
  - 优点（性能）：所有内核代码都能直接访问和调用其它内核代码
  - 缺点（安全性）：任何一个功能模块的漏洞都能影响整个内核的安全

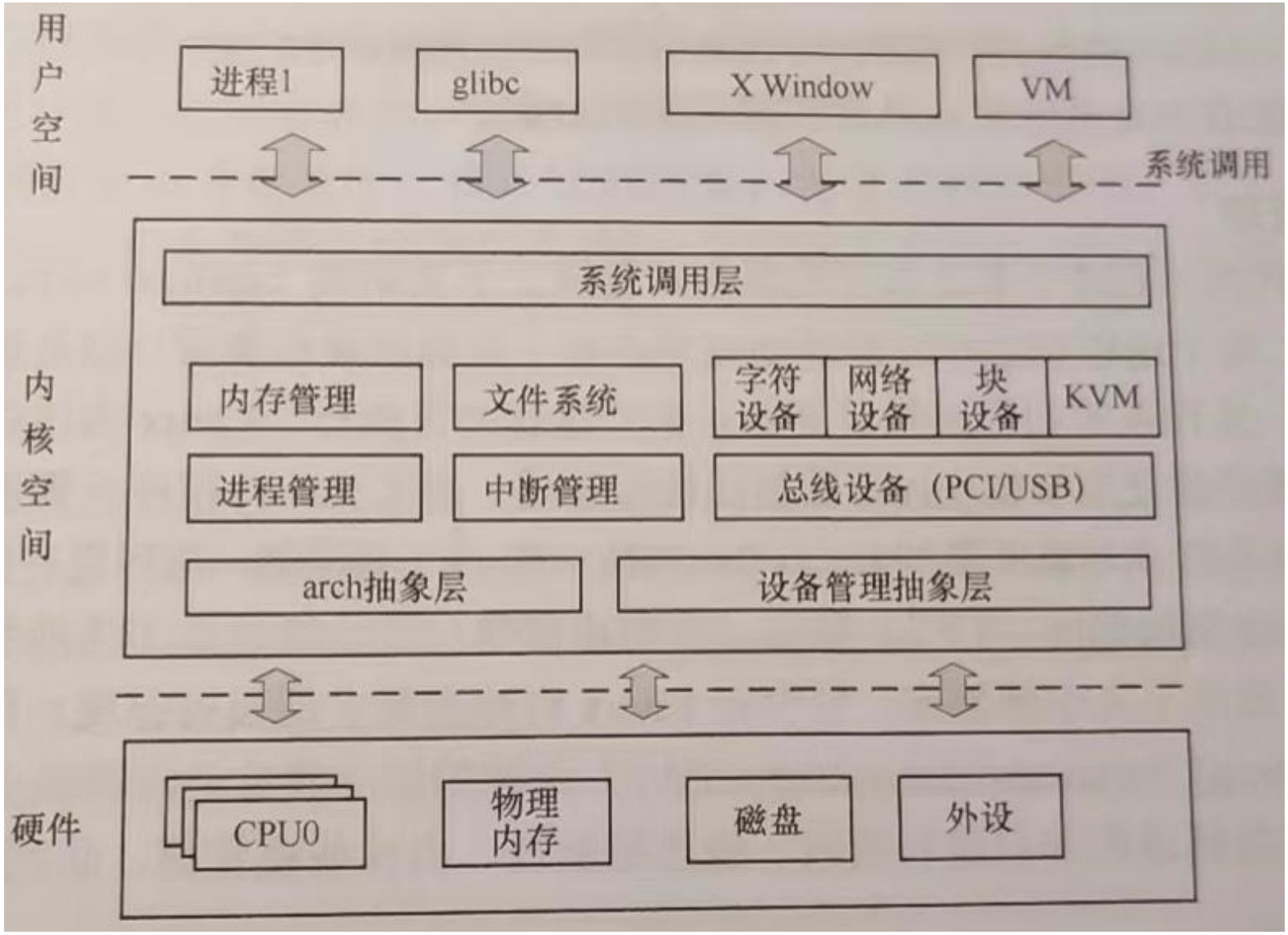


1.2.2 宏内核与微内核

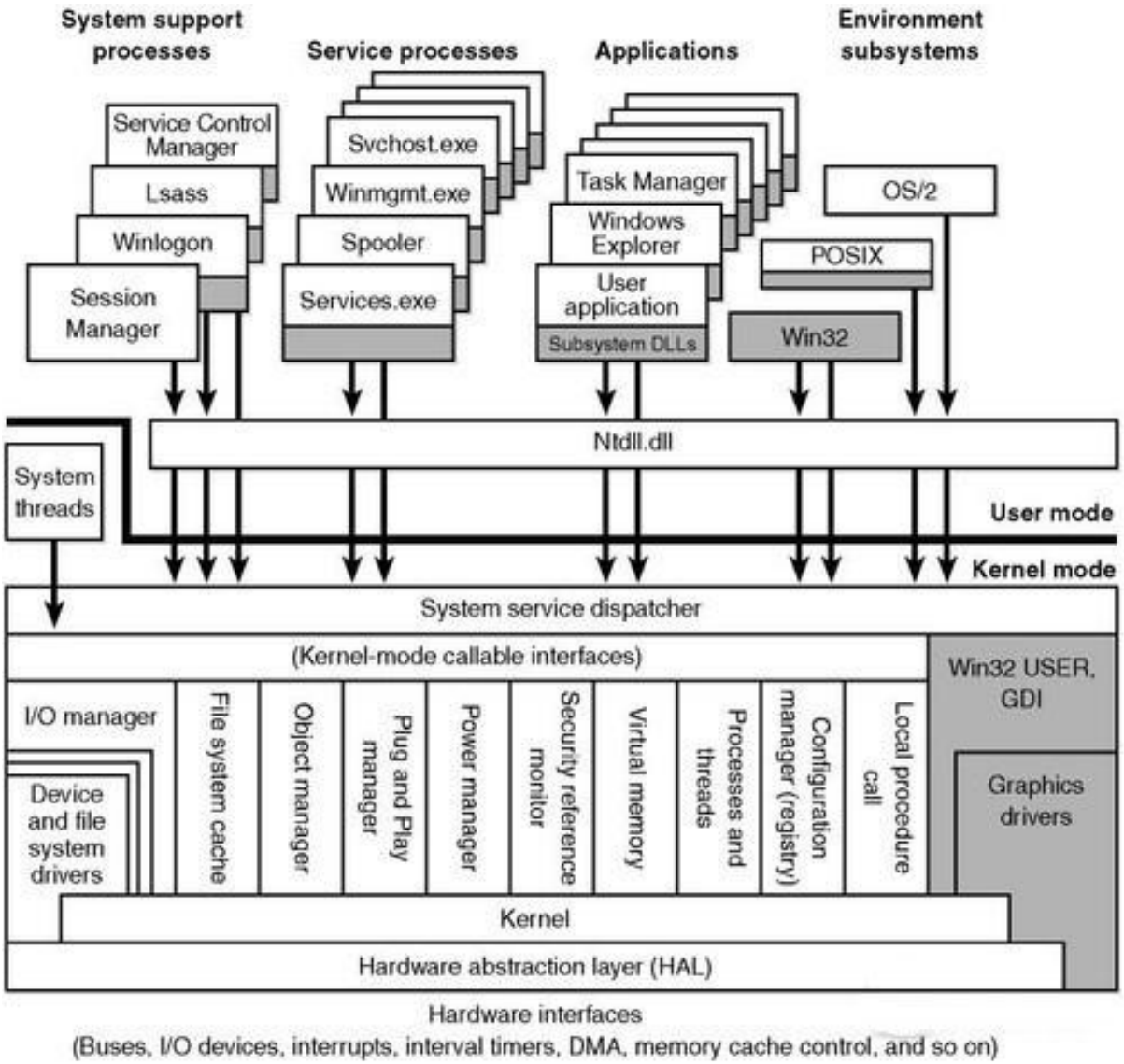
- 微内核：把操作系统分成多个独立的功能模块，最基本的功能由中央内核（微内核）实现，其它功能都委托给一些独立进程，这些进程通过明确定义的通信接口与中央内核通信
  - 优点（安全性）：一个功能模块漏洞的影响范围被限定在单个模块中
  - 缺点（性能）：每个功能模块之间的访问需要通过消息来完成



Linux内核（宏内核）


用户空间
进程1
glibc
X Window
VM
系统调用
内核空间
系统调用层
内存管理
文件系统
字符设备
网络设备
块设备
KVM
进程管理
中断管理
总线设备（PCI/USB）
arch抽象层
设备管理抽象层
硬件
CPU0
物理内存
磁盘
外设

Windows内核（微内核）



1.3 Linux内核源代码

- 下载内核源代码： https://www.kernel.org/

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Release
6.1.4

mainline:	**6.2-rc3**	2023-01-08	[tarball]		[patch]	[inc. patch]	[view diff]	[browse]	
stable:	**6.1.4**	2023-01-07	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable:	**6.0.18**	2023-01-07	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**5.15.86**	2022-12-31	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**5.10.162**	2023-01-04	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**5.4.228**	2022-12-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**4.19.269**	2022-12-14	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**4.14.302**	2022-12-14	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	**4.9.337 [EOL]**	2023-01-07	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
linux-next:	**next-20230112**	2023-01-12						[browse]	

1.3 Linux内核源代码

- 查看内核源代码：https://elixir.bootlin.com/linux/latest/source

linux

Filter tags

- v6
  - v6.2
  - v6.1
    - v6.1.4
    - v6.1.3
    - v6.1.2
    - v6.1.1
    - v6.1
    - v6.1-rc8
    - v6.1-rc7
    - v6.1-rc6
    - v6.1-rc5
    - v6.1-rc4
    - v6.1-rc3

https://elixir.bootlin.com/linux/latest/source/include/linux/atomic

/ include / linux

All s

task_struct

- amba
- atomic
- avf
- bcma
- byteorder
- can
- ceph
- clk
- comedi
- crush
- decompress
- device
- dma
- dsa

Defined in 4 files as a struct:

include/linux/sched.h, line 737 *(as a struct)*
tools/testing/selftests/bpf/progs/strobemeta.h, line 14 *(as a struct)*
tools/testing/selftests/bpf/progs/test_core_reloc_kernel.c, line 27 *(as a struct)*
tools/testing/selftests/bpf/progs/test_core_retro.c, line 7 *(as a struct)*

```
737  struct task_struct {
738  #ifdef CONFIG_THREAD_INFO_IN_TASK
739          /*
740           * For reasons of header soup (see current_thread_info()),
741           * must be the first element of task_struct.
742           */
743          struct thread_info              thread_info;
744  #endif
745          unsigned int                    __state;
746
747  #ifdef CONFIG_PREEMPT_RT
748          /* saved state for "spinlock sleepers" */
749          unsigned int                    saved_state;
750  #endif
```

linux v6.1.4 powered by Elixir 2.1

1.3.1 源代码版本

- 内核版本号：由3个数字组成：A.B.C
  - A：内核主版本号。这是很少发生变化，只有当发生重大变化的代码和内核发生才会发生
  - B：内核次版本号。是指一些重大修改的内核。偶数表示稳定版本；奇数表示开发中版本
  - C：内核修订版本号。是指轻微修订的内核。这个数字当有安全补丁,bug修复，新的功能或驱动程序，内核便会有变化
- 发行版本号：major.minor.patch-build.desc
  - major : 主版本号，有结构变化才变更
  - minor : 次版本号，新增功能时才发生变化
  - patch : 补丁包数或次版本的修改次数
  - build : 编译（或构建）的次数
  - desc : 当前版本的特殊信息

1.3.1 源代码版本

- 例如：用命令uname -a查看内核版本号

```
Linux localhost 3.2.0-67-generic #101-Ubuntu SMP Tue Jul 15 17:46:11 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
#有的是3.2.0-67-generic-pae
```

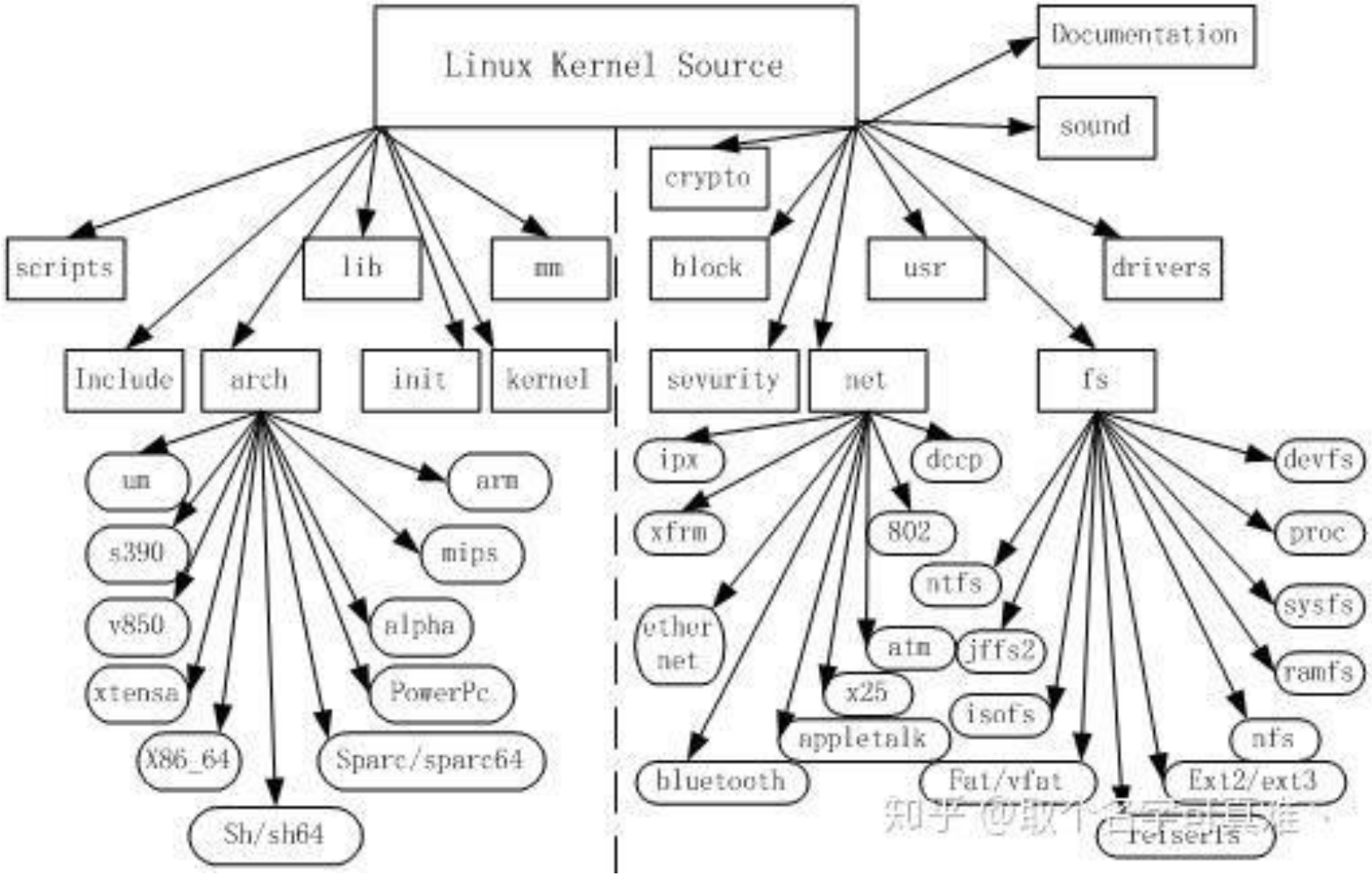
说明如下：

```
第一个组数字：3，主版本号
第二个组数字：2，次版本号，当前为稳定版本
第三个组数字：0，修订版本号
第四个组数字：67，当前内核版本（3.2.0）的第67次微调patch
generic：当前内核版本为通用版本，另有表示不同含义的server（针对服务器）、i386（针对老式英特尔处理器）
pae（Physical Address Extension）：物理地址扩展，为了弥补32位地址在PC服务器应用上的不足而推出，表示此32位系统可以支持超过
x86_64：采用的是64位的CPU
SMP：对称多处理机，表示内核支持多核、多处理器
Tue Jul 15 17:46:11 UTC 2014：内核的编译时间（build date）为 2014/07/15 17:46:11
```

1.3.1 源代码版本

- 内核版本分类
  - mainline：主线版本，正在开发的版本
  - stable：稳定版本，由mainline在时机成熟时发布，稳定版会在相应版本号的主线上提供bug修复和安全补丁，但内核社区人力有限，因此较老版本会停止维护，而标记为EOL(End of Life)的版本表示不再支持的版本
  - longterm：长期支持版，长期支持版的内核不再支持时会标记EOL
  - linux-next，snapshot：代码提交周期结束之前生成的快照用于给Linux代码贡献者们做测试

1.3.2 源代码目录结构


Linux Kernel Source
Documentation
sound
crypto
scripts
lib
mm
block
usr
drivers
Include
arch
init
kernel
sevurity
net
fs
um
s390
v850
xtensa
X86_64
Sh/sh64
Sparc/sparc64
PowerPc
alpha
mips
arm
ipx
xfrm
ether net
bluetooth
appletalk
x25
atm
802
dccp
devfs
proc
sysfs
ramfs
nfs
Ext2/ext3
reiserfs
Fat/vfat
isofs
jffs2
ntfs

1.3.2 源代码目录结构

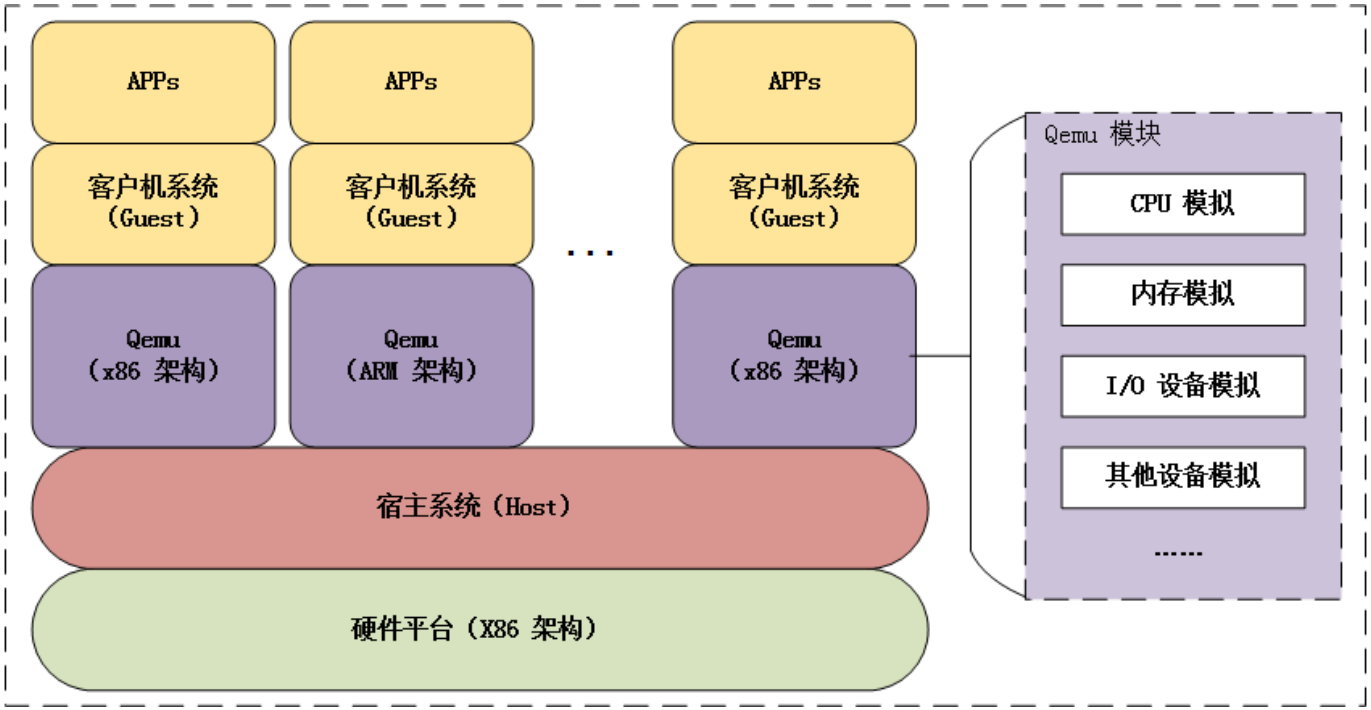
- arch：包含与硬件体系结构相关的代码
  - 每种平台对应一个目录（32位PC相关的代码存放在i386目录下，ARM平台相关的代码存放在arm目录下）
  - 在每个平台目录中还有一些子目录，比较重要的包括kernel（内核核心）、mm（内存管理）、lib（硬件相关工具函数）和boot（引导程序）等
- crypto：常用加密和散列算法，还有一些压缩和CRC校验算法
- Documentation：关于内核各部分的通用解释和注释
- drivers：设备驱动程序，每个不同的驱动占用一个子目录
- fs：各种支持的文件系统，如ext、fat、ntfs等
- include：内核头文件
  - include/linux：内核基本的头文件
  - include/asm-*/：体系结构相关的头文件（*表示体系结构的名称）

1.3.2 源代码目录结构

- init：内核C语言部分的初始化代码（注意不是系统引导代码）
- ipc：进程间通信的代码
- lib：各种库文件代码
- mm：内存管理代码
- net：网络相关代码，实现了各种常见的网络协议
- scripts：用于配置内核文件的脚本文件
- security：实现内核安全机制
- sound：常用音频设备的驱动程序等

1.4 Linux内核实验环境

- Linux内核版本：v5.0
- 使用QEMU全系统模拟方式运行Linux内核
  - 支持多种体系结构（x86、ARM……）
  - 可以在同一台机器上进行内核调试



1.4 Linux内核实验环境

- 目录结构（Host）
  - ${HOME}/Kernel：内核
    - compile：内核编译脚本
    - git：内核源代码仓库
    - image：硬盘镜像（区分i386和x86_64，支持多个Ubuntu版本）
    - share：Host与Guest的共享文件夹（Host端）
    - v5.0：Linux内核v5.0源代码和可执行程序（区分i386和x86_64）
  - ${HOME}/Workdir：模拟器
    - run、debug、script：运行和调试脚本
    - scp、ssh：远程连接和远程文件传输脚本（Host与Guest互连、互传文件）
    - compile、src、build、exe：QEMU编译脚本、源代码、中间文件和可执行文件
- 目录结构（Guest）
  - /tmp/linux：内核源代码（从Host端映射到Guest端）
  - /tmp/share：Host与Guest的共享文件夹（Guest端）

1.4 Linux内核实验环境

- 准备环境
  - sudo apt-get install gcc-8 g++-8
  - sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8
  - sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 90 --slave /usr/bin/g++ g++ /usr/bin/g++-9
  - sudo update-alternatives --config gcc
- 编译内核
  - cd ${HOME}/Kernel
  - ./compile v5.0 i386 #编译v5.0内核（i386：32位，x86_64：64位）
- 运行内核
  - cd ${HOME}/Workdir
  - ./run v5.0 i386 buster #在Ubuntu buster版本运行v5.0内核
  - ./ssh i386 buster 10000 #通过10000端口进行远程连接（可选步骤）
- 调试内核
  - cd ${HOME}/Workdir
  - ./run v5.0 i386 buster -dbg 20000 #指定调试端口为20000
  - ./debug v5.0 i386 20000 vmlinux_script #通过20000端口调试运行的内核
  - source module_script #当需要调试的内核模块载入后，在gdb中执行该命令