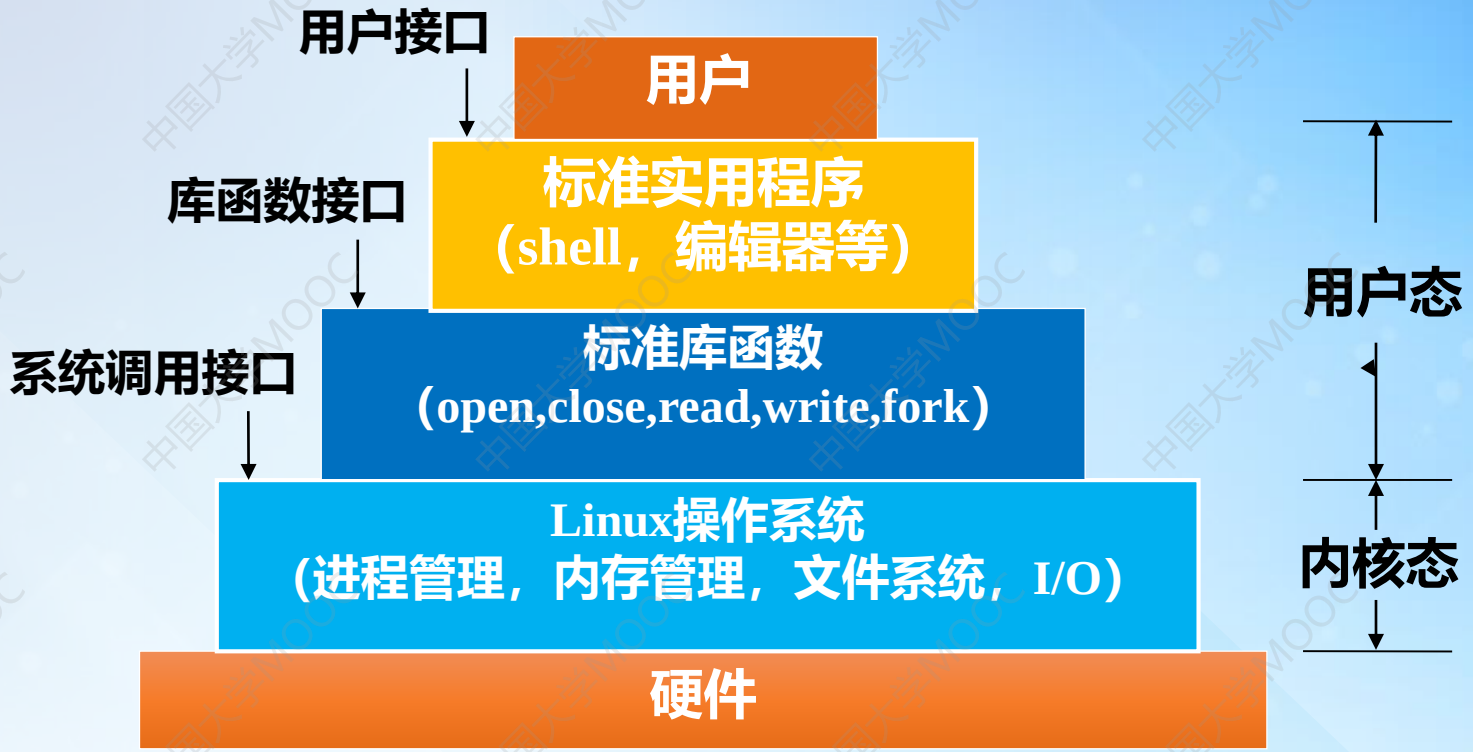
# 操作系统及Linux内核

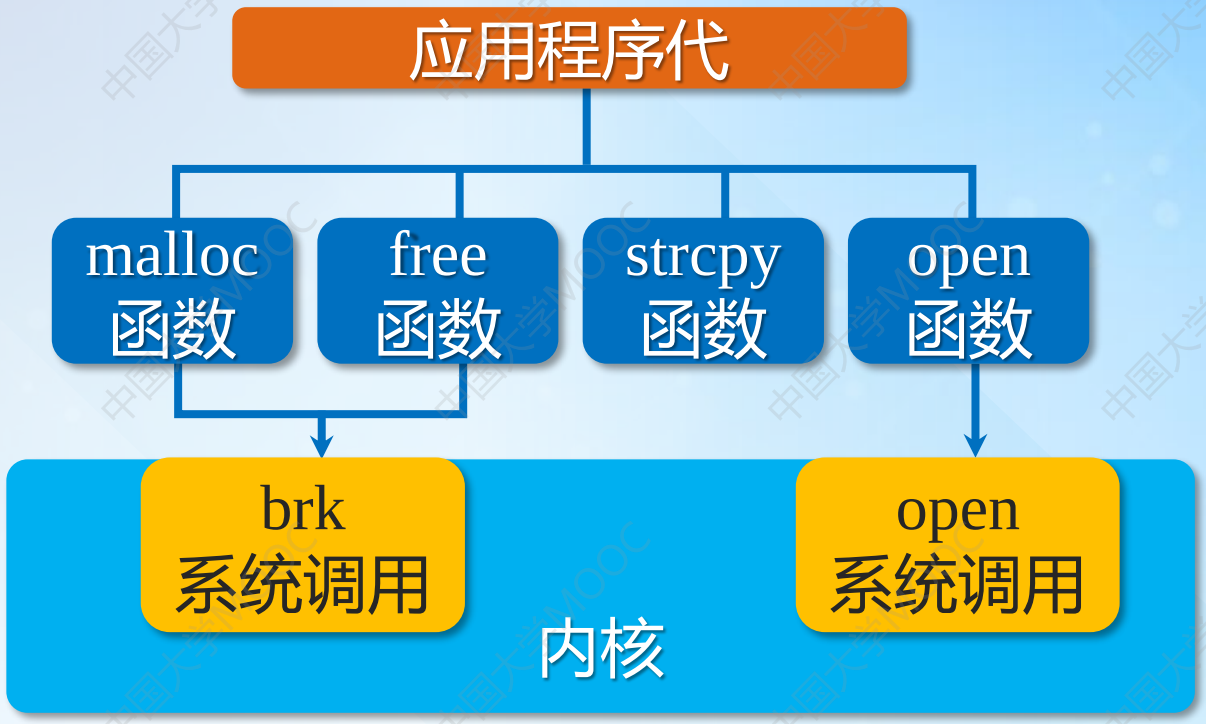
西安邮电大学

# 操作系统接口

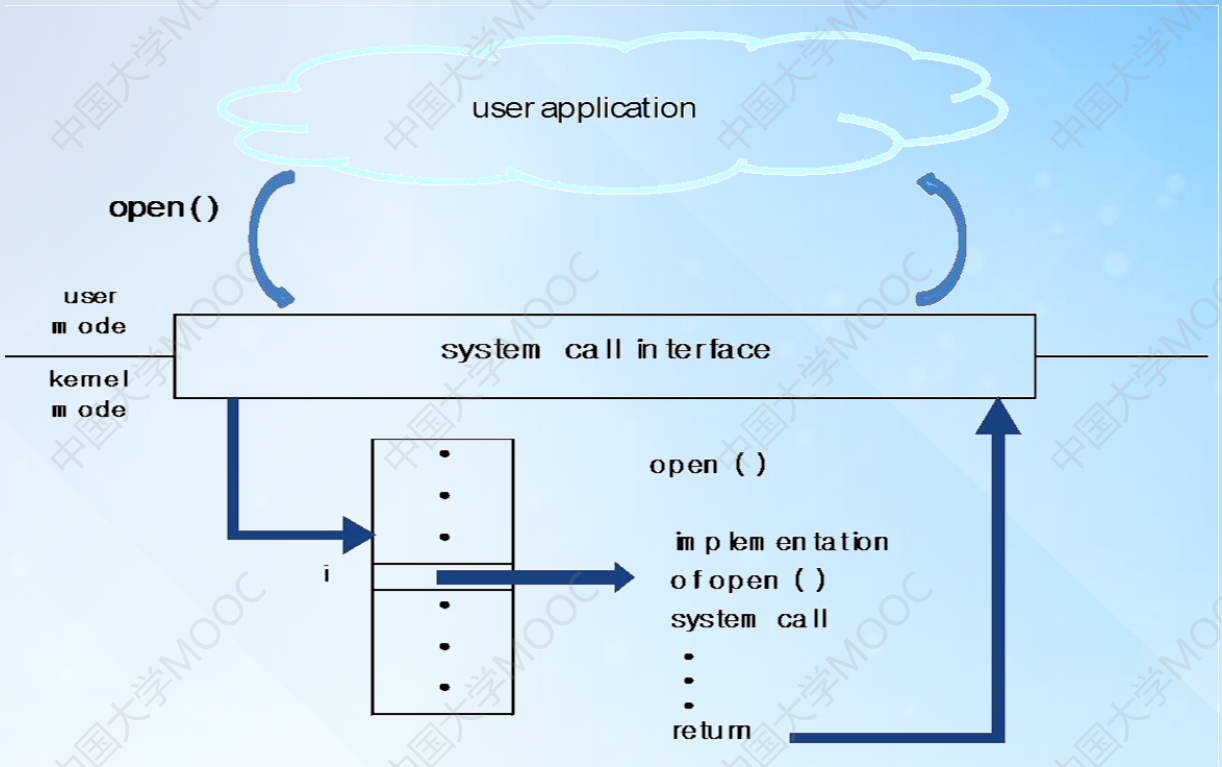
# Linux 各种接口


用户接口
用户
库函数接口
标准实用程序
（shell，编辑器等）
系统调用接口
标准库函数
（open,close,read,write,fork）
Linux操作系统
（进程管理，内存管理，文件系统，I/O）
硬件
用户态
内核态

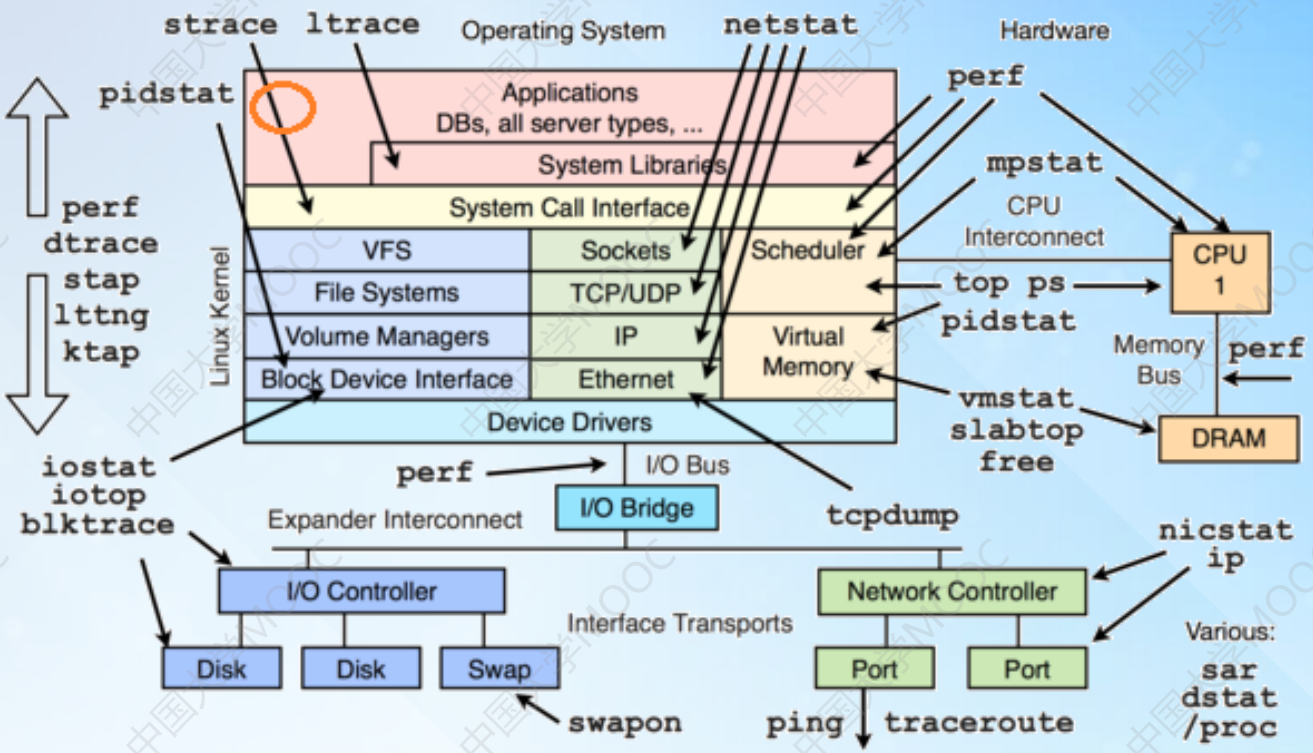
# 系统调用与API


应用程序代
malloc
函数
free
函数
strcpy
函数
open
函数
brk
系统调用
open
系统调用
内核

# 系统调用 - 内核的出口


user application
open()
user mode
kernel mode
system call interface
open ( )
implementation of open ( ) system call
return
i

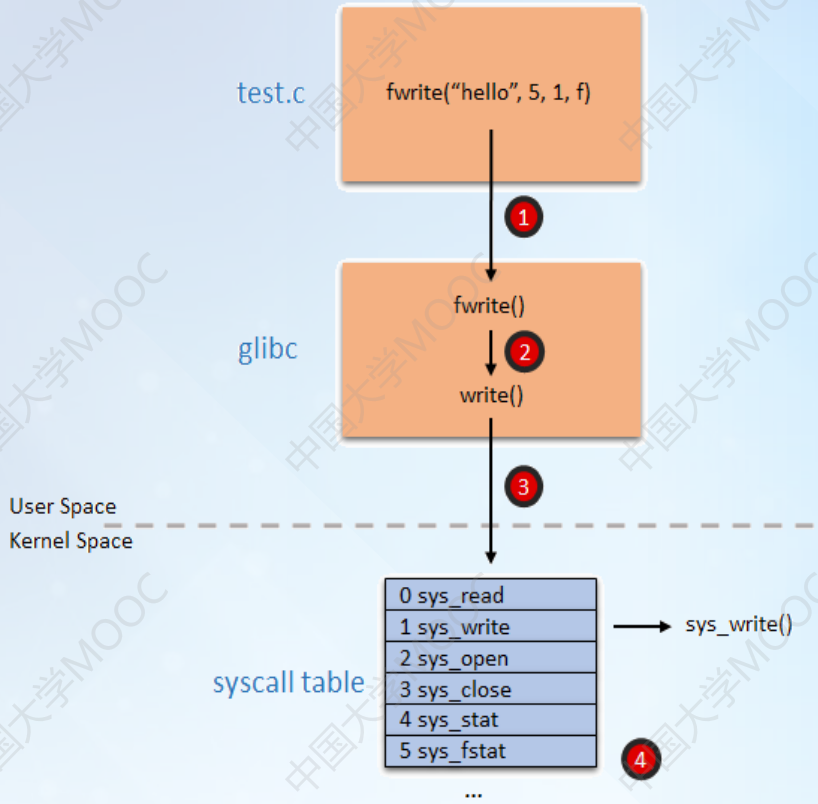
# 跟踪进程所调用的系统调用



# strace命令跟踪系统调用样例

```
[clj@VM_0_7_centos hds]$ strace -c ./father_child
Before fork ...
I am father.
The pid of parent is: 32369
the pid of parent's child is: 32373
After fork, program exiting...
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------
 31.91    0.000045          45         1           clone
 20.57    0.000029           4         8           mmap
 15.60    0.000022           4         5           write
 12.06    0.000017           4         4           mprotect
  5.67    0.000008           4         2           open
  4.26    0.000006           2         3           fstat
  4.26    0.000006           6         1           munmap
  2.13    0.000003           2         2           close
  1.42    0.000002           2         1           read
  1.42    0.000002           2         1           getpid
  0.71    0.000001           1         1           arch_prctl
  0.00    0.000000           0         1           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------
100.00    0.000141                    32         1 total
```

# 从用户态函数到系统调用




比如在程序中调用fwrite函数，而fwrite函数在glibc库中调用系统调用write(),然后从用户态陷入内核态，查找系统调用表，对应的系统调用服务例程为sys_write

# 系统调用基本概念

## 系统调用号

①用来唯一的标识每个系统；

②作为系统调用表的下标，当用户空间的进程执行一个系统调用时，该系统调用号就被用来指明到底要执行哪个进程。

## 系统调用表

作用是用来把系统调用号和相应的服务例程关联起来。

该表存放在sys_call_table数组中：

# 系统调用基本概念

```
ENTRY(sys_call_table)
        .long sys_restart_syscall       /* 0 - old "setup()" system
                                           call，used for restarting */
        .long sys_exit
        .long sys_fork
        .long sys_read
        .long sys_write
        .long sys_open          /* 5 */
……
```

# 部分系统调用列表

| %eax | Name | Source | %ebx | %ecx | %edx | %esx | %edi |
|---|---|---|---|---|---|---|---|
| 1 | sys_exit | kernel/exit.c | int | | | | |
| 3 | sys_read | fs/read_write.c | unsigned int | char | size_t | | |
| 4 | sys_write | fs/read_write.c | unsigned int | const char | size_t | | |
| 15 | sys_chmod | fs/open.c | const char | Mode_t | | | |
| 20 | Sys_getpid | kernel/timer.c | void | | | | |
| 21 | sys_mount | fs/namespace.c | char_user* | char_user* | char_user* | unsigned long | vold_user* |
| 88 | sys_reboot | kernel/sys.c | int | int | unsigned int | vold_user* | |

System Call Number

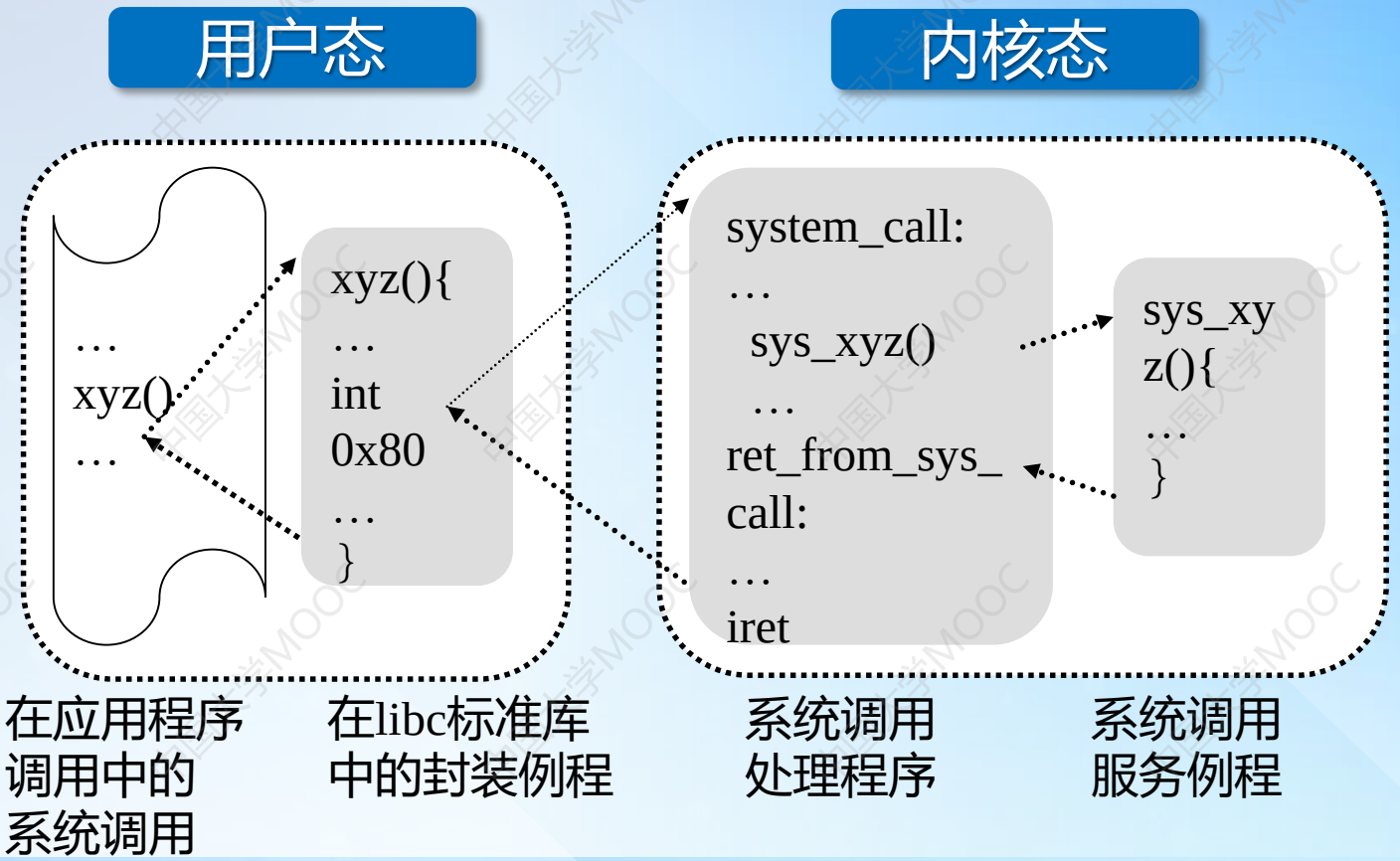
System Call Number

First parameter

Second parameter

Third parameter

# 调用一个系统调用的过程



# 系统调用处理程序及服务例程

当用户态的进程调用一个系统调用时，CPU切换到内核态并开始执行一个内核函数

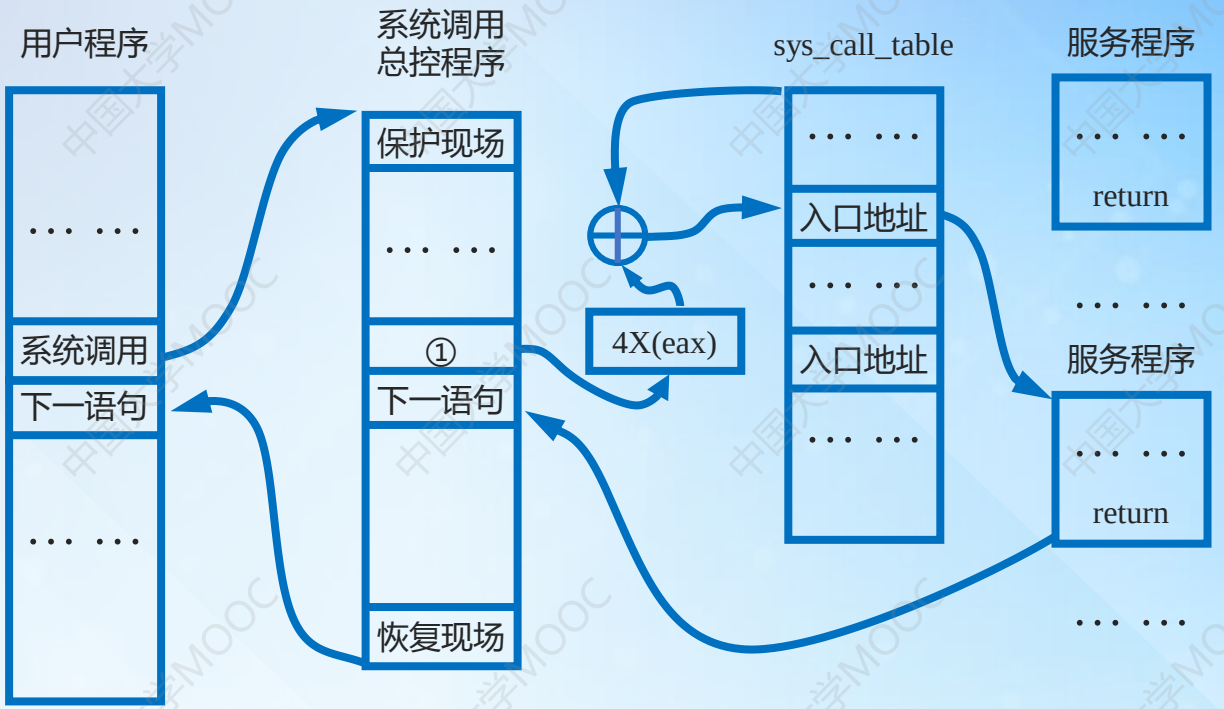
**系统调用处理程序执行下列操作：**

1. 在内核栈保存大多数寄存器的内容
2. 调用所谓系统调用服务例程的相应的C函数来处理系统调用
3. 通过ret_from_sys_call( )函数从系统调用返回

查看源代码：

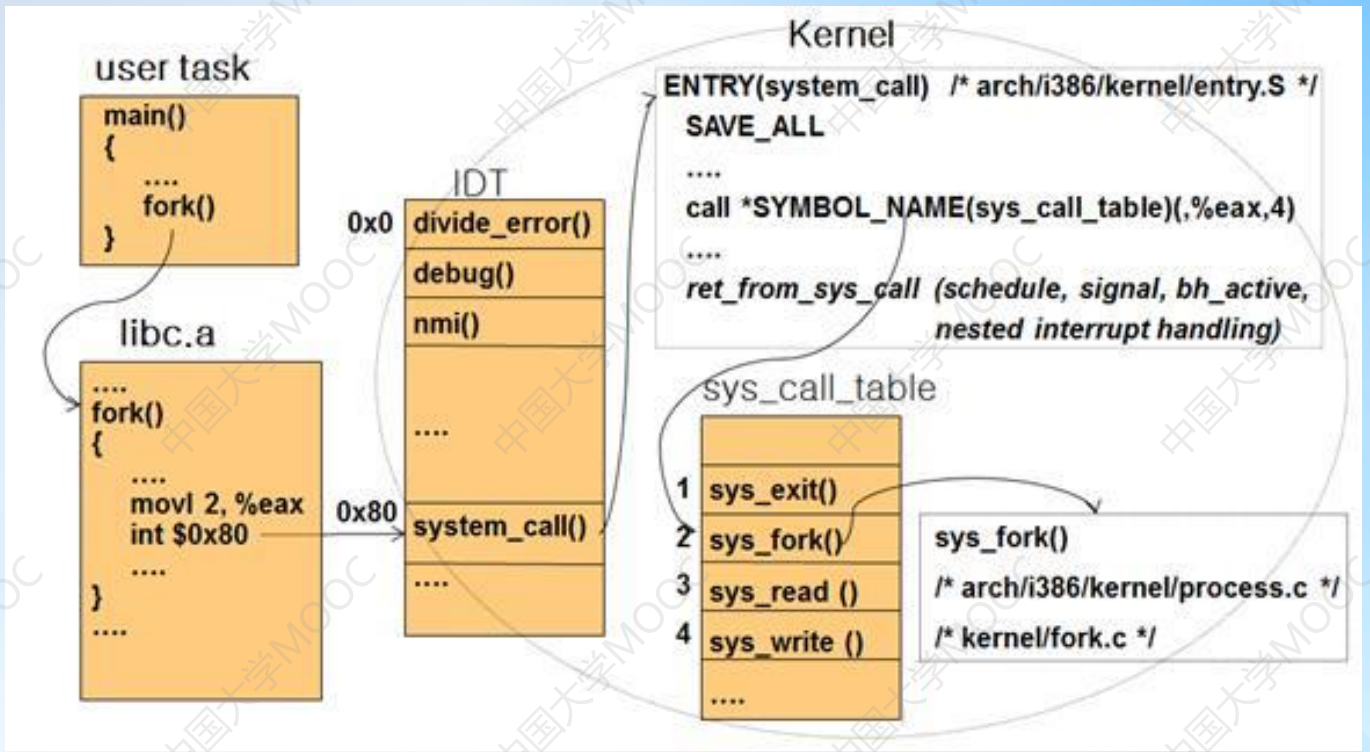
/linux-3.x.x/arch/x86/kernel/entry_32(64).S

# 系统调用处理过程

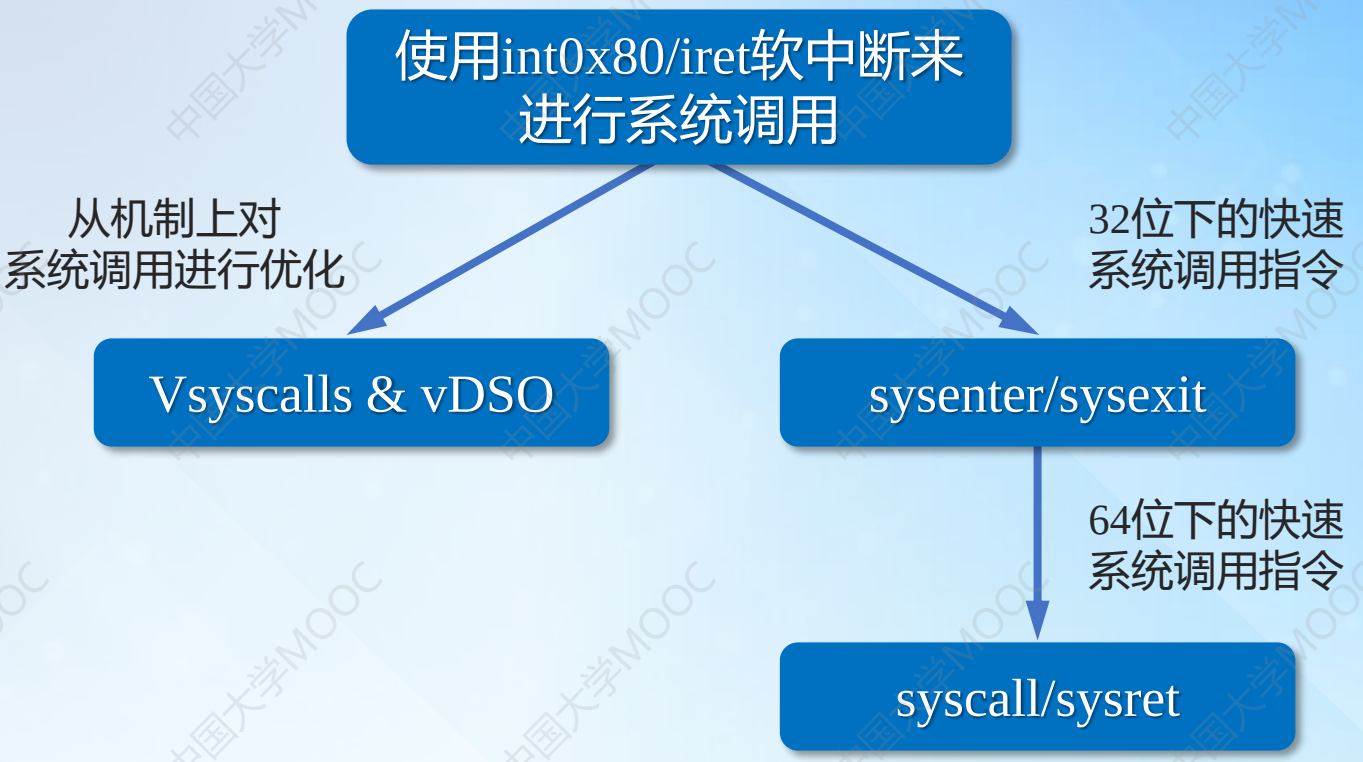



注①：此处语句为：call*SYMBOL_NAME(sys_call_table)(

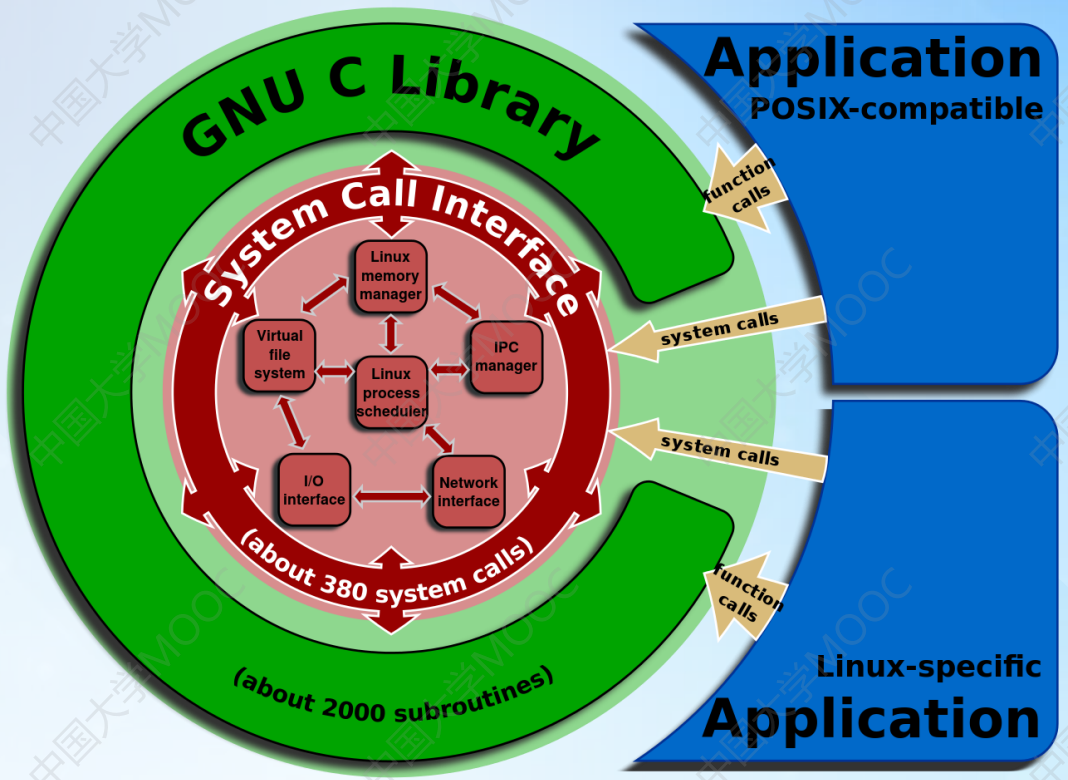
# 从用户态跟踪一个系统调用到内核



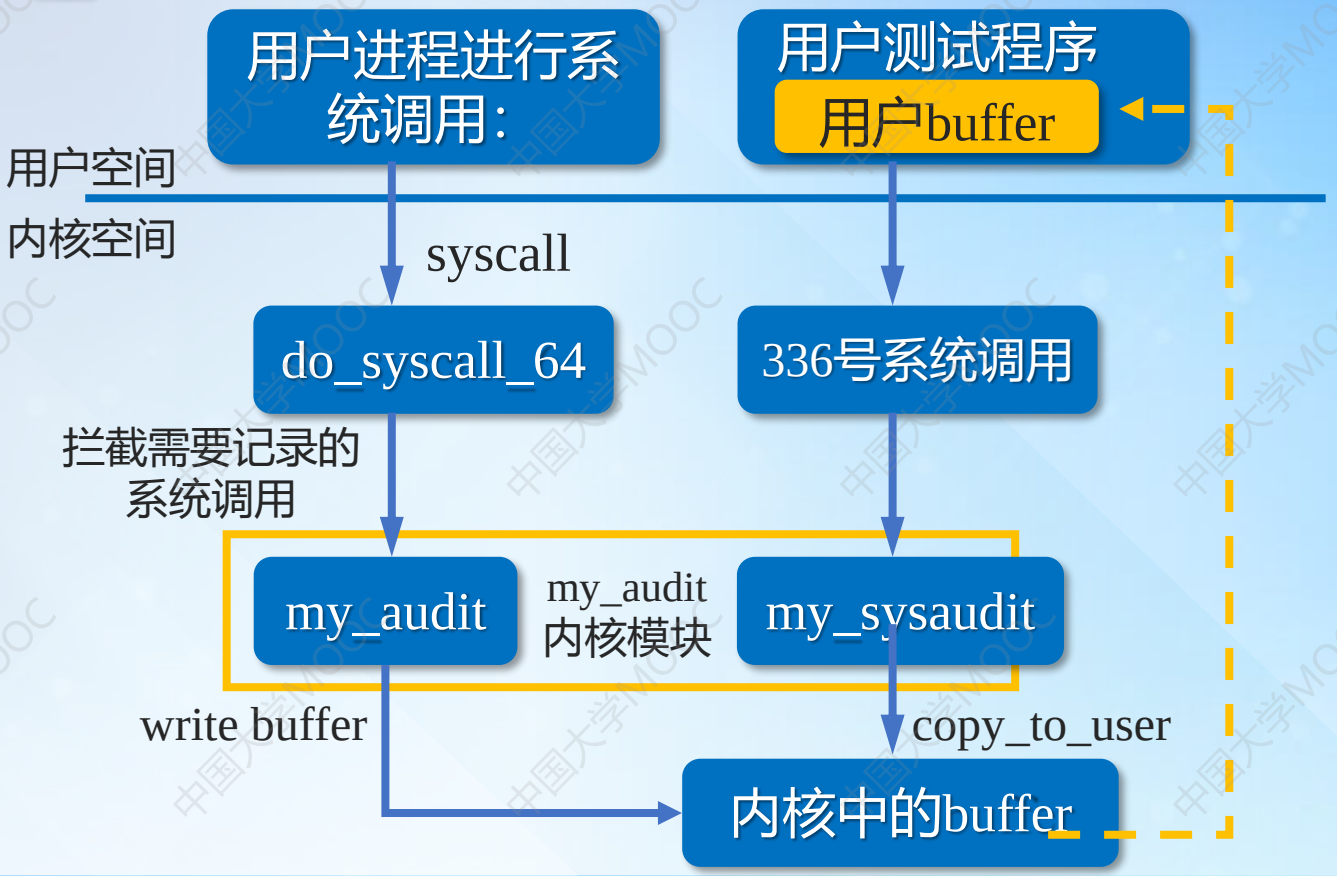
# 系统调用机制的优化


使用int0x80/iret软中断来进行系统调用
从机制上对系统调用进行优化
32位下的快速系统调用指令
Vsyscalls & vDSO
sysenter/sysexit
64位下的快速系统调用指令
syscall/sysret

# GNU C库与系统调用是什么关系?



# 系统调用实例-日志收集系统



# 小结

**Linux 系统调用**

- 1. Linux三种接口
  - 用户接口
  - 库函数接口
  - 系统调用接口
- 2.追踪系统调用
- 3.处理系统调用
- 4.优化系统调用
- 5. 系统调用实例