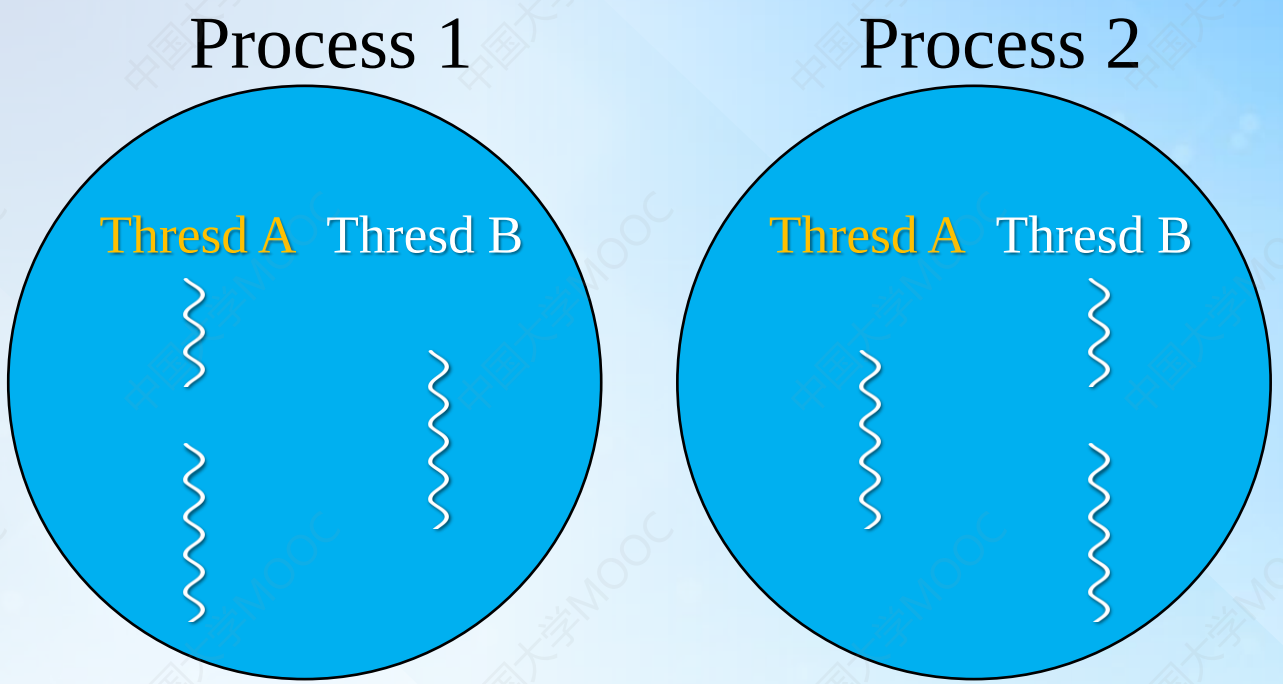
操作系统及Linux内核

西安邮电大学

Linux中进程和线程的创建

进程和线程



进程和线程

代码

数据

进程空间

打开文件

寄存器 栈 线程1

寄存器 栈 线程2

寄存器 栈 线程3

寄存器 栈 线程4

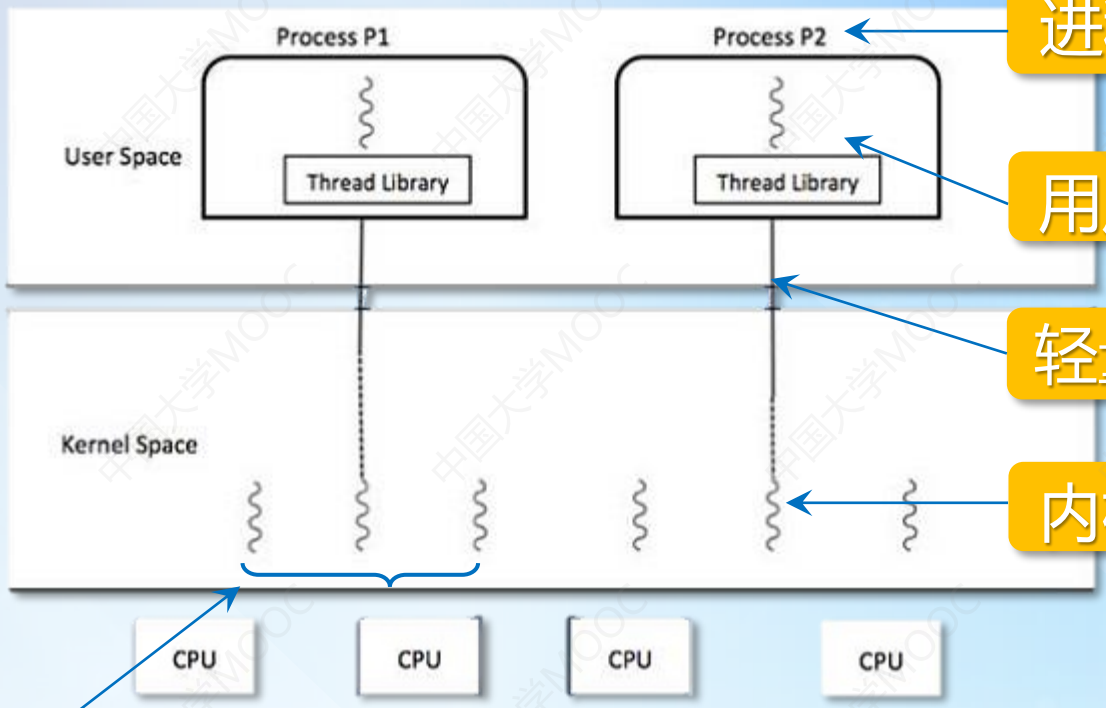
Linux中进程和线程标识符

进程标识符

轻量级进程标识符
（等于内核线程标识符）

```
[clj@cloudhhu ~]$ ps -LA
  PID   LWP TTY          TIME CMD
    1     1 ?        01:14:15 systemd
    2     2 ?        00:00:01 kthreadd
    3     3 ?        00:06:59 ksoftirqd/0
    5     5 ?        00:00:00 kworker/0:0H
    7     7 ?        00:00:00 migration/0
 4171  4362 ?        00:00:00 gmain
 8581  8581 ?        00:00:00 httpd
 8581  8584 ?        00:00:00 httpd
```

Linux中进程和线程标识符



进程pid

用户级线程tid

轻量级进程LWP

内核级线程tid

tgid，线程组leader的id，等于进程的PID

task_struct结构的统一性与多样性

task_struct

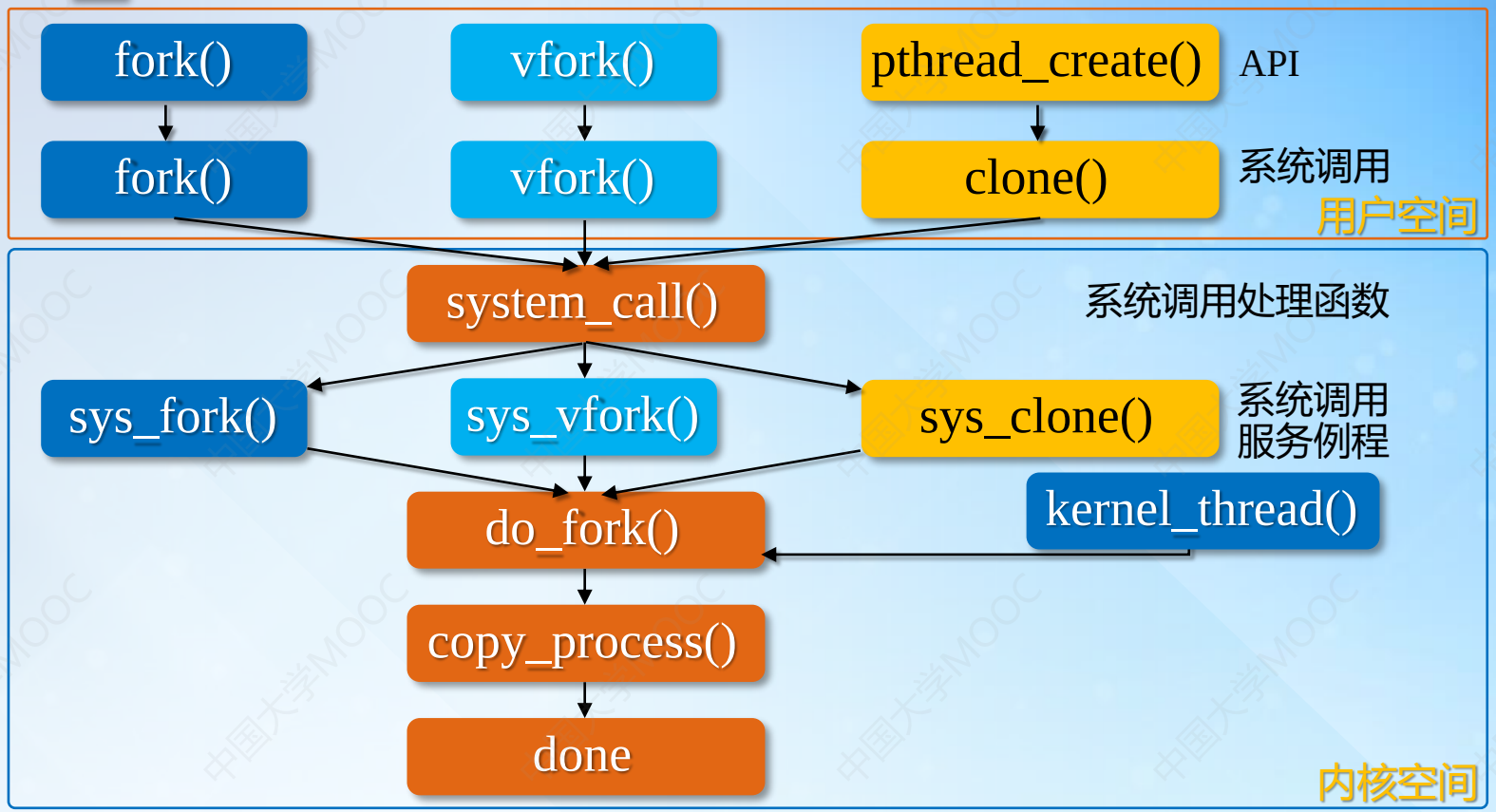
进程

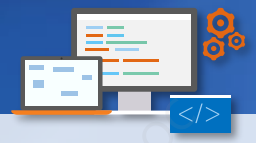
线程

内核线程

内核中最终都通过do_fork()分别创建它们

进程和线程的API实现

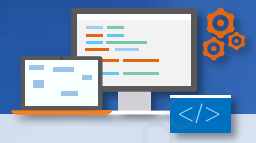




do_fork()

do_fork()在内核中的原型：

```
long do_fork(unsigned long clone_flags,
             unsigned long stack_start,
             unsigned long stack_size,
              int __user *parent_tidptr,
              int __user *child_tidptr)
```

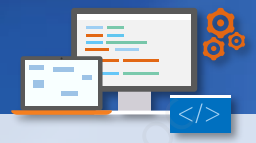


三个系统调用如何调用do_fork

```
int sys_fork(struct pt_regs regs)
{
        return do_fork(SIGCHLD, regs.esp, &regs, 0, NULL, NULL);
}
```

```
int sys_vfork(struct pt_regs regs)
{
        return do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, regs.esp, &regs, 0);
}
```

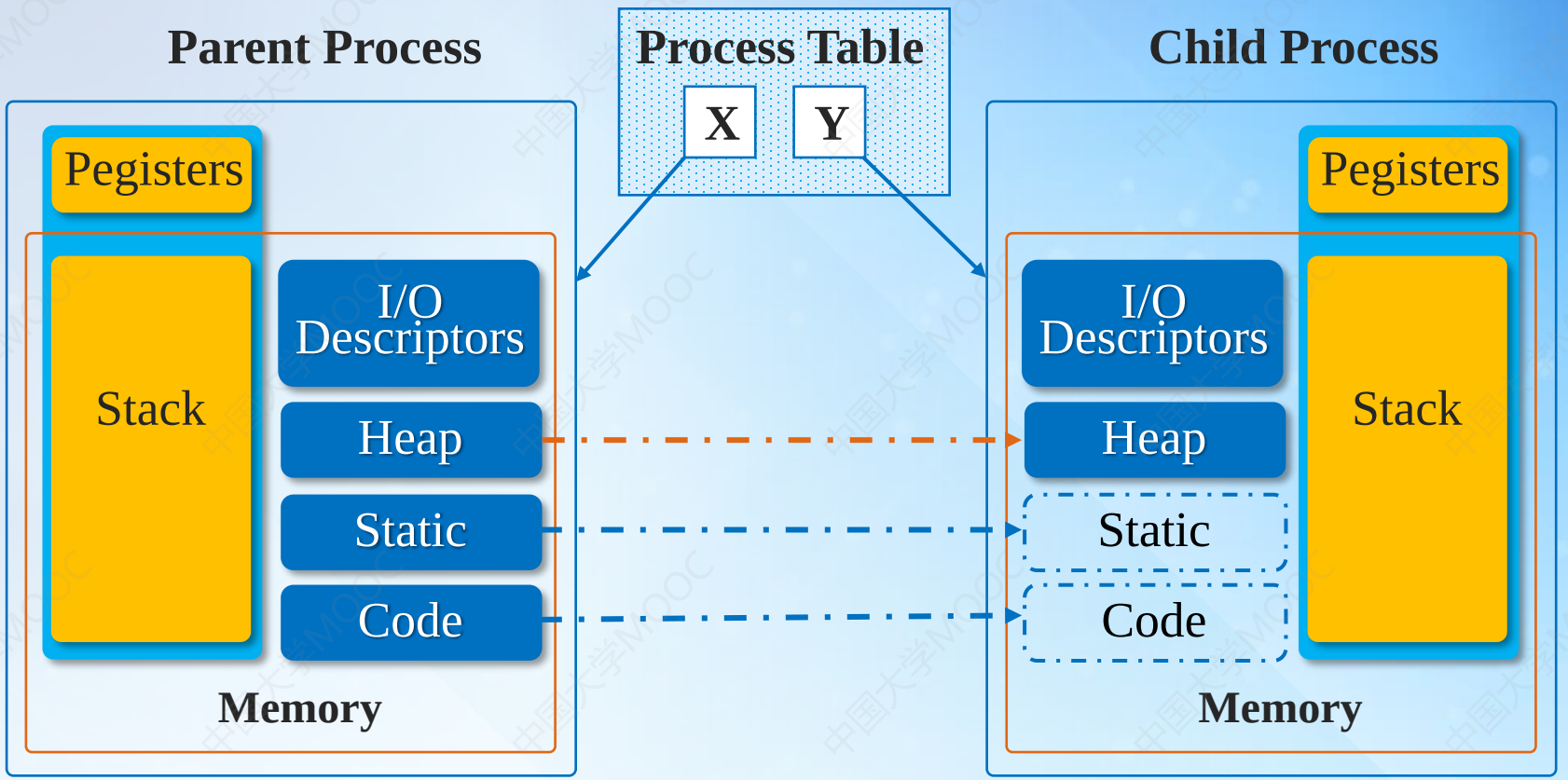
```
long sys_clone(unsigned long, unsigned long, int __user *,
                int __user *, unsigned long);
{
        return _do_fork(clone_flags, newsp, 0, parent_tidptr, child_tidptr, tls);
}
```



fork的实现

```
 int sys_fork(struct pt_regs regs)
{
        return do_fork(SIGCHLD, regs.esp, &regs, 0, NULL, NULL);
}
```

fork的实现


Parent Process
Process Table
X
Y
Child Process
Pegisters
Stack
I/O Descriptors
Heap
Static
Code
Memory
Pegisters
Stack
I/O Descriptors
Heap
Static
Code
Memory

vfork的实现

```
 int sys_vfork(struct pt_regs regs)
{
      return do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, regs.esp, &regs, 0);
}
```

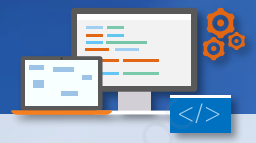
- 第一个标志（CLONE_VFORK），儿子优先，老爸等着。于是父进程就去睡觉，等子进程结束才能醒来。
- 第二个标志（CLONE_VM）儿子干脆与父亲待在一个进程的地址空间中，对，就是共享父进程的内存地址空间（父进程的页表项除外）。

clone的实现

```
long sys_clone(unsigned long, unsigned long, int __user *,
                int __user *, unsigned long);
{
        return _do_fork(clone_flags, newsp, 0, parent_tidptr, child_tidptr, tls);
}
```

四个参数:

- CLONE_VM
- CLONE_FS
- CLONE_FILES
- CLONE_SIGHAND



内核线程的创建

```
pid_t kernel_thread(int (*fn)(void *), void *arg, unsigned long flags)
{
        return do_fork(flags|CLONE_VM|CLONE_UNTRACED,
(unsigned long)fn,
            (unsigned long)arg, NULL, NULL);
}
```

- 早期内核中创建内核线程是通过kernel_thread()函数的，目前内核中调用 kthread_create()或者kthread_run创建内核线程，其本质也是向do_fork()提供特定的flags标志而创建的。

task_struct带来的统一性

task_struct

进程

线程

内核线程

do_fork()代码流程

- do_fork
  - copy_process()
  - 确定PID
  - 如果设置了CLONE_STOPPED，阻塞子进程
  - wake_up_new_task()
  - 如果CLONE_VFORK标志被设置，阻塞父进程

copy_process()代码流程图

- copy_process
  - 检查标志合法性
  - dup_task_struct()
  - 检查资源限制
  - 初始化task_stuct中的字段
  - sched_fork()
  - 复制或共享父进程的各个资源
    - copy_semundo()
    - copy_files()
    - copy_fs()
    - copy_sighand()
    - copy_singal()
    - copy_mm()
    - copy_namespace()
    - copy_thread()
  - 设置PID，进程关系等等

父子进程的资源共享-copy_XXX()

复制或共享父进程的各个资源

- copy_semundo()
- copy_files()
- copy_fs()
- copy_sighand()
- copy_singal()
- copy_mm()
- copy_namespace()
- copy_thread()

小结