奔跑吧 LINUX社区

第4季

BenOS操作系统相关的知识

保密文件，仅供奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

本节课主要内容

- 本章主要内容
  - 如何创建进程
  - 实现一个简易的调度器
  - 让进程运行在用户态
  - 系统调用

技术手册：

1. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA， Document Version 20191213

本节课主要讲解书上第17章内容

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

Part 1：创建进程

任务要求：创建一个内核进程，然后一直打印计数

```
kernel_thread: 0
kernel_thread: 1
kernel_thread: 2
kernel_thread: 3
kernel_thread: 4
kernel_thread: 5
kernel_thread: 6
kernel_thread: 7
kernel_thread: 8
kernel_thread: 9
kernel_thread: 10
kernel_thread: 11
kernel_thread: 12
kernel_thread: 13
kernel_thread: 14
kernel_thread: 15
kernel_thread: 16
```

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员学习之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

进程与程序

进程 = 程序 + 执行

- 程序：完成特定任务的一系列指令集合或者指的是一个可执行文件，包含可运行的一堆CPU指令和相应的数据等信息，它不具有生命力。
- 进程：有生命的个体，它不仅仅包含代码段数据段等信息，还有很多运行时需要的资源。
- 进程是操作系统分配内存、CPU时间片等资源的基本单位。

保密文件，已提供购买了奔跑吧Linux社区第4季视频课程
《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

进程控制块

- 使用task_struct数据结构描述一个进程控制块（Process Control Block，PCB）

- cpu_context用来表示进程切换时的硬件上下文。
- state表示进程的状态。
- count用来表示进程调度用的时间片。
- priority用来表示进程的优先级。
- pid用来表示进程的ID。

```
struct task_struct {
    struct cpu_context cpu_context;
    enum task_state state;
    enum task_flags flags;
    long count;
    int priority;
    int pid;
};
```

include/sched.h

```
enum task_state {
    TASK_RUNNING = 0,
    TASK_INTERRUPTIBLE = 1,
    TASK_UNINTERRUPTIBLE = 2,
    TASK_ZOMBIE = 3,
    TASK_STOPPED = 4,
};
```

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

0号进程

- BenOS的启动流程： 上电→MySBI固件→BenOS汇编入口→kernel_main()函数。
- 从进程的角度来看，init进程可以看成系统的“0号进程”。

- 使用INIT_TASK宏来静态初始化0号进程的进程控制块

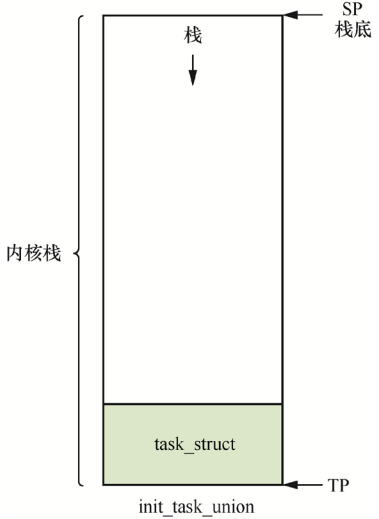
```
/*0号进程即init进程*/
#define INIT_TASK(task) \
{                       \
    .state = 0,     \
    .priority = 1,   \
    .flags = PF_KTHREAD,   \
    .pid = 0,     \
}
```

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

0号进程的内核栈

- 对于0号进程，把内核栈放到.data.init_task段里。
- 定义了一个内核栈的框架，内核栈的底部用来存储task_struct

```
/*
 *task_struct数据结构存储在栈顶（位于栈的底部）
 */
union task_union {
    struct task_struct task;
    unsigned long stack[THREAD_SIZE/sizeof(long)];
};
```



- 把task_union编译、链接到.data.init_task段。

```
/*把0号进程的内核栈编译、链接到.data.init_task段*/
#define __init_task_data __attribute__((__section__(".data.init_task")))

/*0号进程为init进程*/
union task_union init_task_union __init_task_data = {INIT_TASK(task)};
```

- 链接文件linker.ld中新增一个名为.data.init_task的段

```
SECTIONS
{
    ...
    . = ALIGN(PAGE_SIZE);
    _data = .;
    .data : {
        *(.data)
        . = ALIGN(PAGE_SIZE);
        *(.data.init_task)
    }
    ...
}
```

union task_union **init_task_union** __attribute__((__section__(".data.init_task"))) = {INIT_TASK(task)};

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

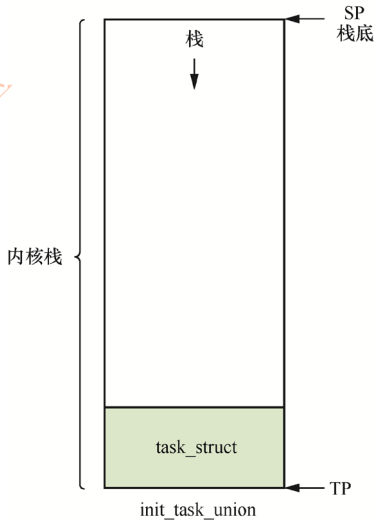
获取task_struct数据结构

- 设置TP指向init_task_union

```
1  .globl _start
2  _start:
3      ...
4      /* 设置栈: init_task_union + THREAD_SIZE*/
5      la sp, init_task_union
6      li t0, THREAD_SIZE
7      add sp, sp, t0
8      la tp, init_task_union
```

- 获取task_struct数据结构

```
static struct task_struct *get_current(void)
{
  register struct task_struct *tp __asm__("tp");
  return tp;
}

#define current get_current()
```



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览使用
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

do_fork实现

```
int do_fork(unsigned long clone_flags, unsigned long fn, unsigned long arg)
```

- do_fork()函数新建一个进程，其流程如下：
  （1）新建一个task_struct数据结构，设置好内核栈
  （2）为新进程分配PID。
  （3）设置进程的上下文。

保密文件，仅供购买了奔跑吧Linux社区第一季视频课程《RISC-V体系结构编程与实战》的学员浏览使用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

进程上下文切换

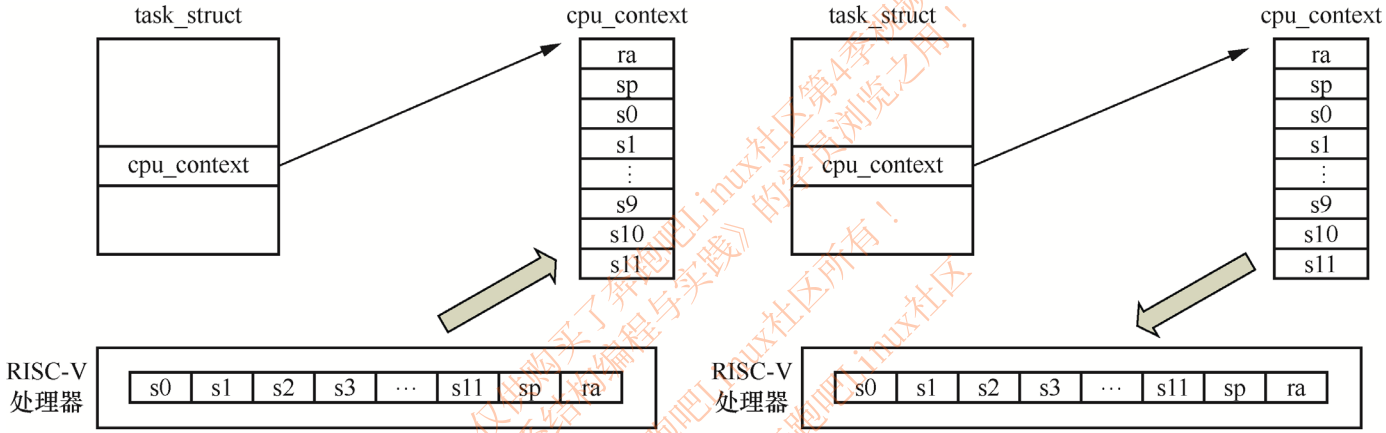
- cpu_switch_to()函数，它用于保存prev进程的上下文，并且恢复next进程的上下文。

```
void cpu_switch_to(struct task_struct *prev, struct task_struct *next);
```

- 保存的上下文包括s0 ~ s11寄存器、sp寄存器以及ra寄存器的值，把它们保存到next进程的task_struct->cpu_context中
- 从next进程的task_struct->cpu_context中恢复处理器中这些寄存器的值。
- cpu_context数据结构用来把进程上下文的相关信息保存到与CPU相关的通用寄存器中。

```
<benos/include/asm/processor.h>

/*切换进程时需要保存的上下文*/
struct cpu_context {
    unsigned long ra;
    unsigned long sp;  /*栈指针*/

    /*函数调用过程中必须要保存的通用寄存器s0 ~ s11的值*/
    unsigned long s[12];
};
```

仅供购买《RISC-V体系结构编程与实践》的学员浏览使用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

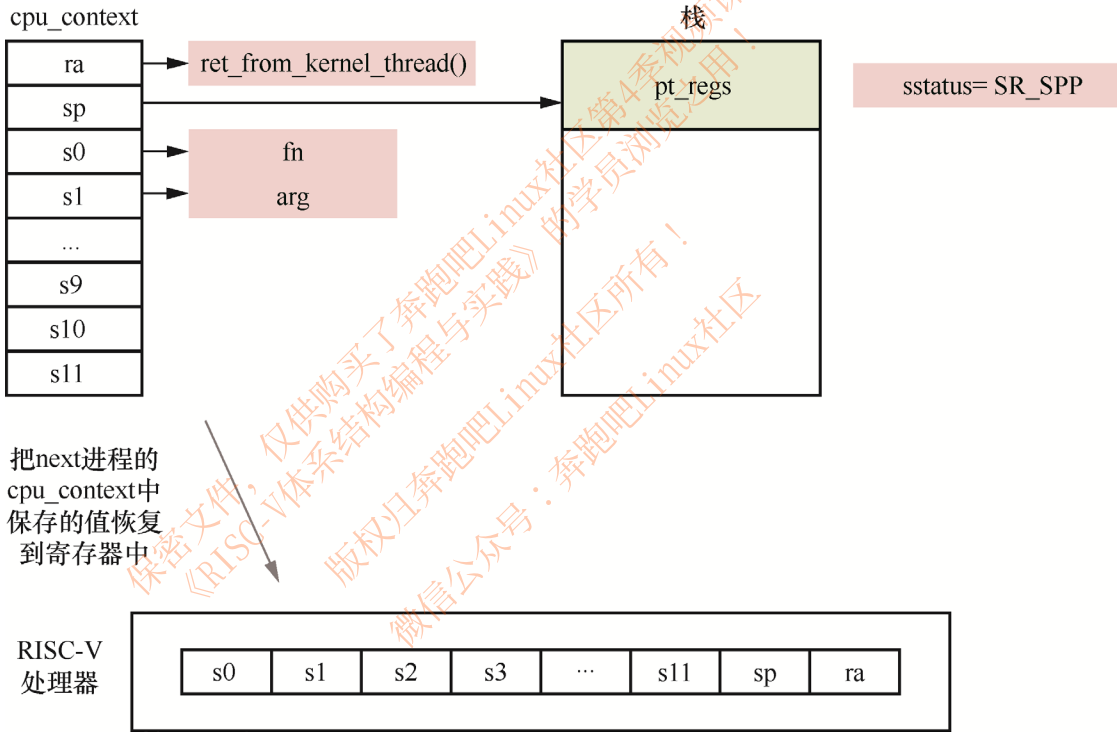


（a）把寄存器的值保存到prev进程的cpu_context中

（b）把next进程存储的上下文恢复到CPU中

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

新进程的第一次执行



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览使用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

Part 2：实现一个简单的调度器

任务：创建两个内核线程，这两个内核线程只能在内核空间中运行，线程A输出“12345”，线程B输出“abcde”，要求调度器能合理调度这两个内核线程，二者交替运行，而系统的0号进程不参与调度。

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的用户浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

实验目的

- 了解 进程调度的概念，发展历史
- 了解进程调度的本质，灵魂拷问：
  - 调度的时机是什么？
  - 如何合理和高效地选择下一个进程？
  - 如何切换到下一个进程？
  - 当调度器切换到next进程来运行时，那next进程执行的第一条指令是什么？
  - 下一个进程如何返回上一次暂停的地方？
- 通过实验和单步调试的方式，深入理解 进程调度

保密文件，仅供购买《奔跑吧Linux社区第4季视频课程 《RISC-V体系结构编程实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

基本概念

- 就绪队列：存储即将要参与调度的候选进程。
- 就绪队列可以是链表也可以是红黑树

- 调度策略：
  - ✓ O(n)调度器
  - ✓ 经典多级调度算法Multi-level Feedback Queue
  - ✓ Linux 2.6内核的O(1)调度算法
  - ✓ CFS调度器

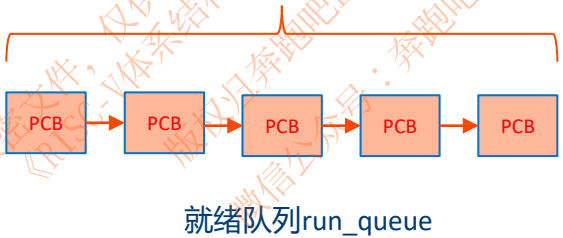
- 调度类：操作系统为了支持多种不同的调度策略，实现一个统一的抽象框架

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

简易的调度器

- 实现一个简易的调度器，类似Linux 0.11里的调度器

- 它遍历就绪队列中所有的进程，然后找出剩余时间片最大的那个进程并以它作为next进程。
- 如果就绪队列里所有进程的时间片都用完了，为所有进程的时间片重新赋值。



保密文件，仅供购买了奔跑吧Linux系列视频课程《RISC-V体系结构编程与实践》的学员个人使用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

自愿调度与抢占调度

- 自愿调度就是进程主动调用schedule()函数来放弃CPU的控制权。
- 抢占调度是指在中断处理返回之后，检查是否可以抢占当前进程的运行权。

保密文件，仅供购买了奔跑吧Linux社区第4期视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

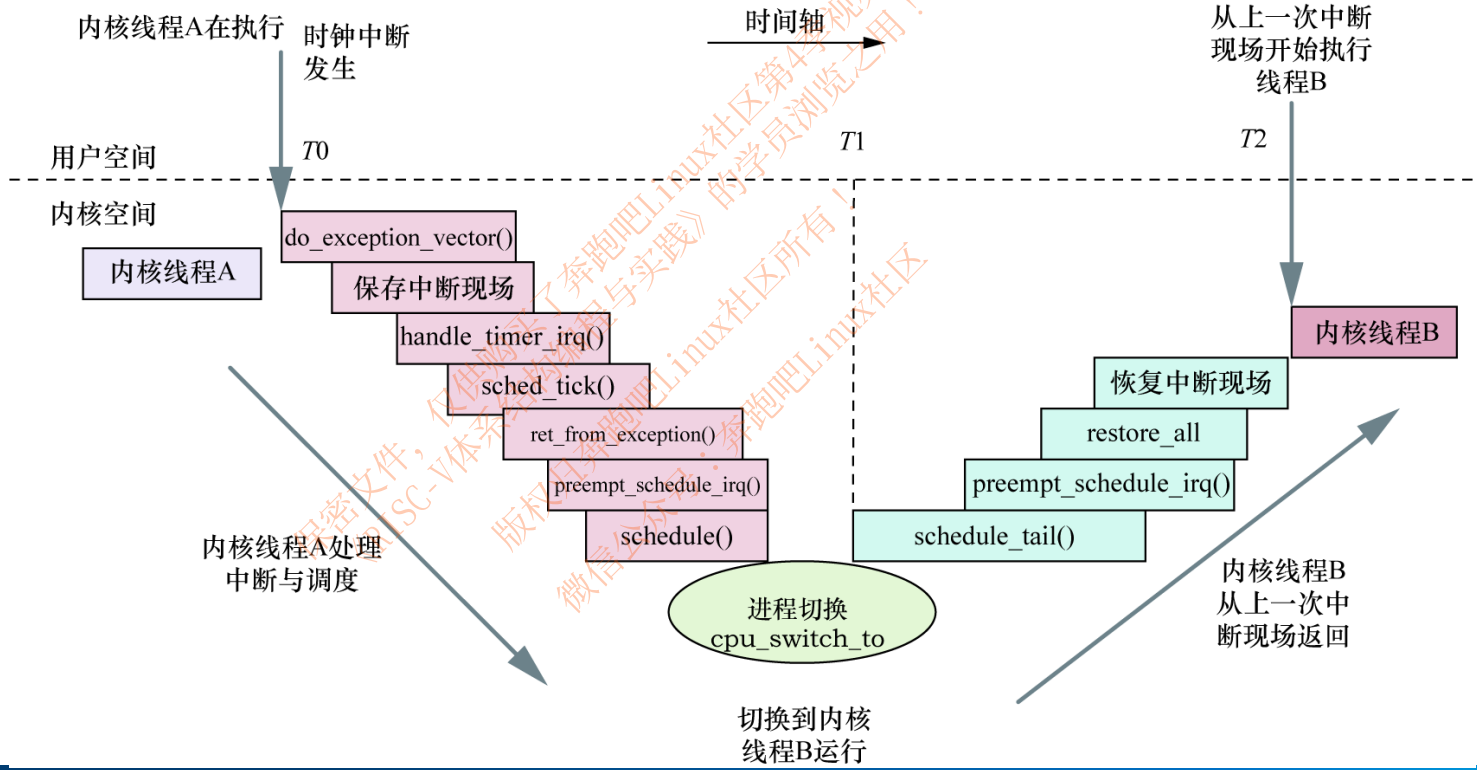
关于调度的思考

- 调度相关的几个核心问题：
  - 调度的时机是什么？
  - 如何合理和高效地选择下一个进程？
  - 如何切换到下一个进程？
  - 当调度器切换到next进程来运行时，那next进程执行的第一条指令是什么？
  - 下一个进程如何返回上一次暂停的地方？

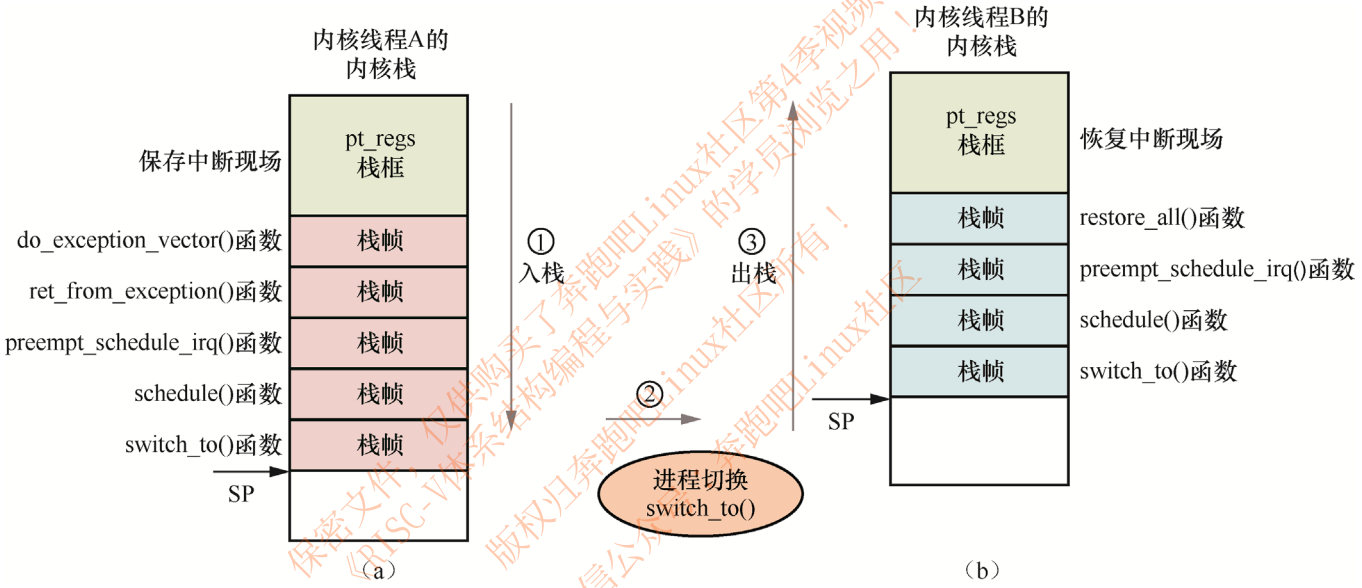
保密文件，仅供购买了《RISC-V体系结构编程与实战》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区
奔跑吧Linux社区第4季视频课程

案例分析

假设系统中有两个内核线程A和B，在不考虑自愿调度和系统调用的情况下，请描述这两个内核线程是如何相互切换并运行的。



版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

Part 3：让进程运行在用户模式

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程
《RISC-V体系结构编程与实战》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

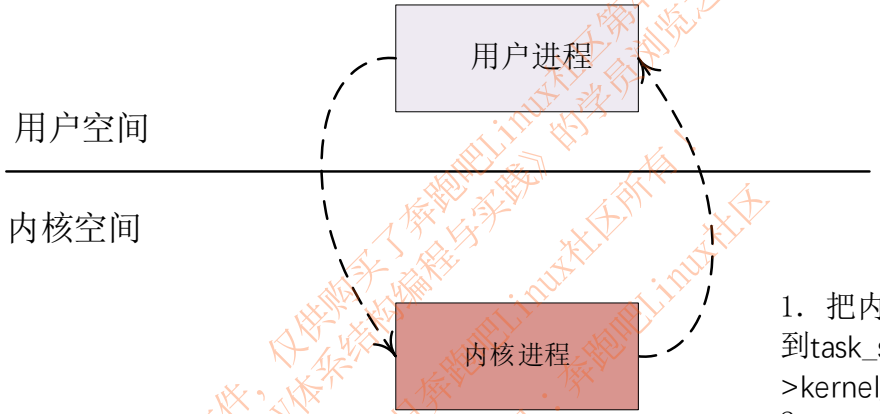
让进程运行在用户模式

- RISC-V体系结构中，所有的处理器模式共用一个SP，需要妥善处理SP问题
- BenOS在task_struct数据结构中新增两个字段，用来保存内核模式的SP和用户模式的SP

```
/*进程控制块*/
struct task_struct {
    ...
    unsigned long kernel_sp;
    unsigned long user_sp;
    ...
};
```

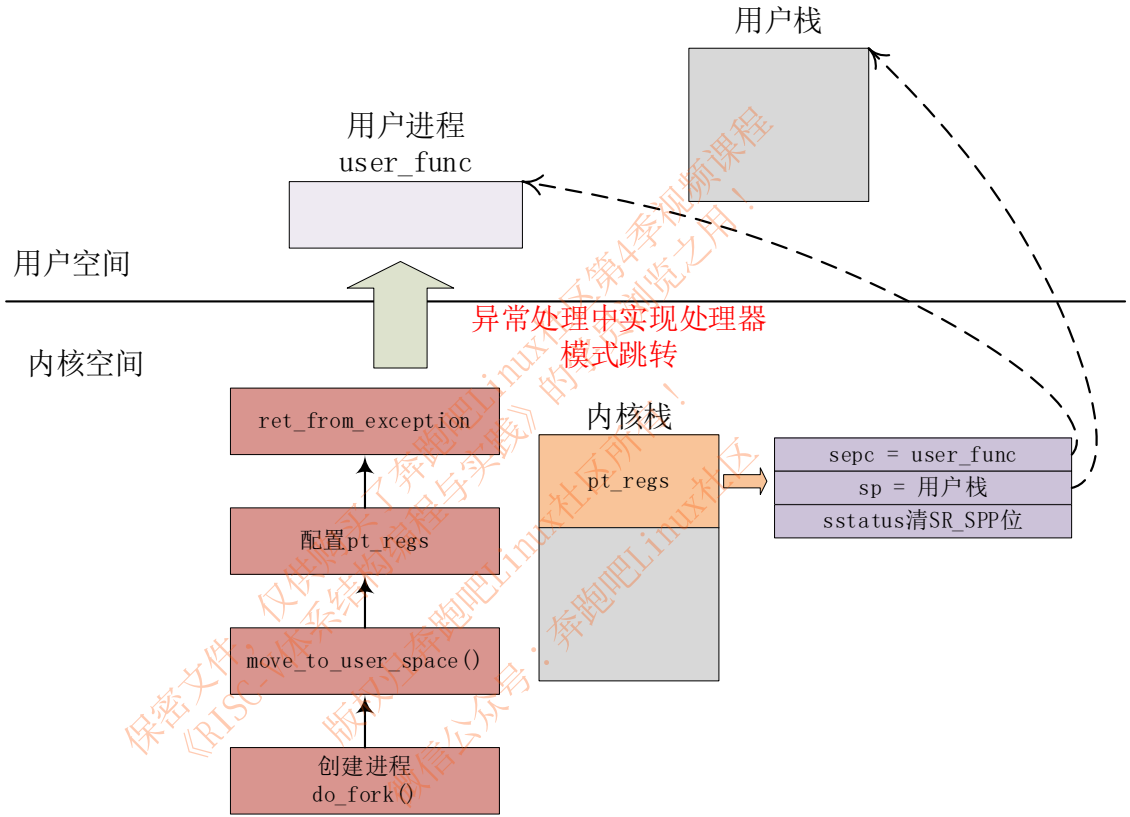
保密文件，仅限购买《奔跑吧Linux社区第4季 视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

1.读sscratch寄存器到tp寄存器
2. 用户态的sp保存到task_struct->user_sp
3. task_struct->kernel_sp中加载正确的内核态sp



1. 把内核栈sp保存到task_struct->kernel_sp
2. task_struct的指针保存到sscratch寄存器

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区



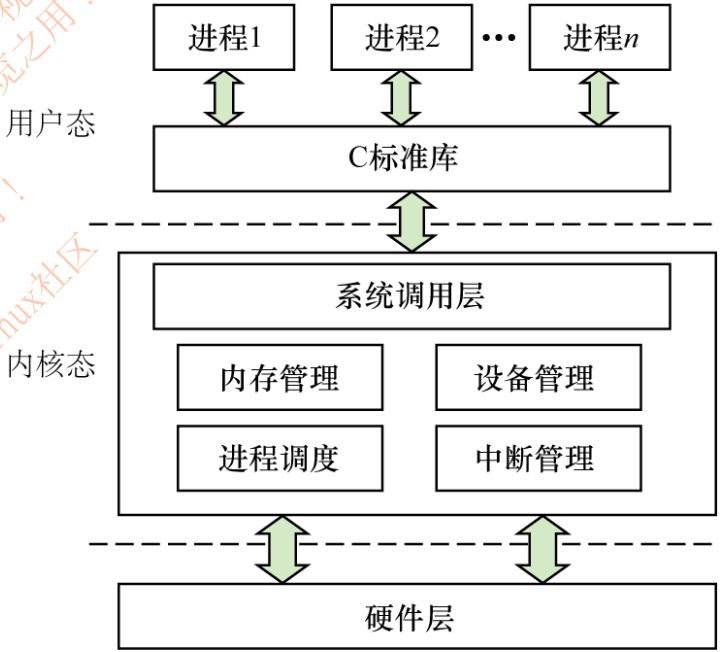
问题1：进程要切换到处理器用户模式运行，需要做哪些准备？
问题2：进程要切换到处理器用户模式运行，什么时候切？

Part 4：系统调用

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程
《RISC-V体系结构编程与实战》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

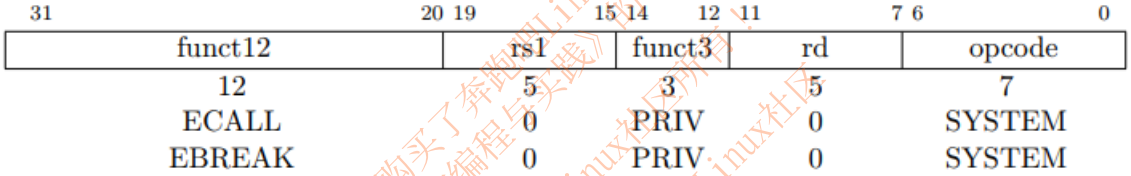
系统调用层

- 系统调用层：内核地址空间和用户地址空间之间的中间层
  - 为用户地址空间中的程序提供硬件抽象接口。
  - 保证系统稳定和安全。
  - 可移植性。



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

RISC-V中的ECALL指令



31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

系统调用过程

- 操作系统为每个系统调用赋予了一个系统调用号

```
unsigned long open(const char *filename, int
flags)
{
    return syscall(__NR_open, filename, flags);
}
```

```
#define __NR_open 0
#define __NR_close 1
#define __NR_read 2
#define __NR_write 3
#define __NR_clone 4
#define __NR_malloc 5
#define __NR_syscalls 6
```

```
1   .global syscall
2   syscall:
3       move    t0, a0
4       move    a0, a1
5       move    a1, a2
6       move    a2, a3
7       move    a3, a4
8       move    a4, a5
9       move    a5, a6
10      move    a6, a7
11      move    a7, t0
12      ecall
13      ret
```

ecall指令陷入内核态

_NR_syscalls个
系统调用

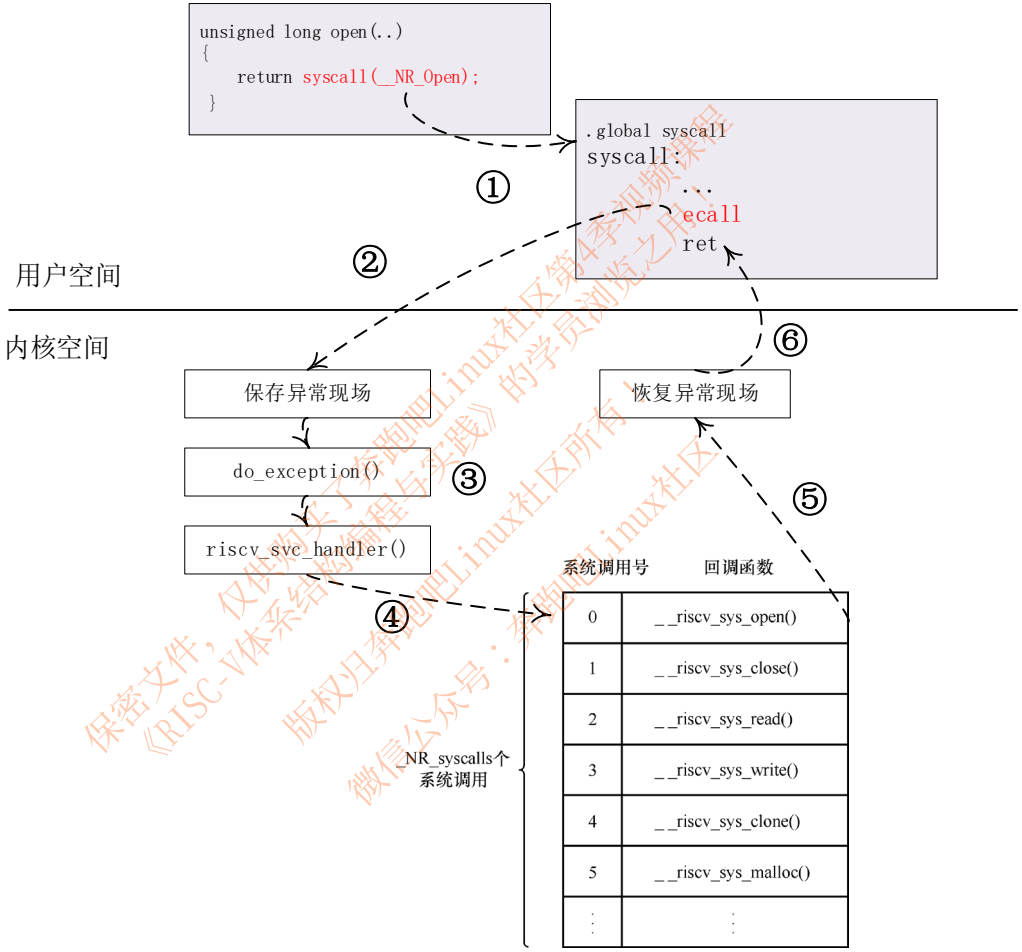
系统调用号	回调函数
0	__riscv_sys_open()
1	__riscv_sys_close()
2	__riscv_sys_read()
3	__riscv_sys_write()
4	__riscv_sys_clone()
5	__riscv_sys_malloc()
⋮	⋮

用户态

内核态

保密文件，仅供奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

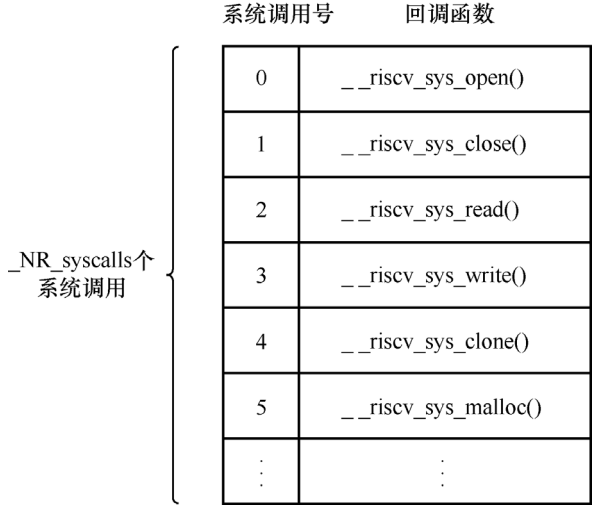
系统调用过程



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程
《RISC-V体系结构编程与实践》的学员浏览之用
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

系统调用表

- 操作系统内部维护了一个系统调用表。在BenOS中我们使用syscall_table[]数组实现这个表。
- 每个表项包含一个函数指针，由于系统调用号是固定的，只需要查表就能找到系统调用号对应的回调函数。




```
1   #define __SYSCALL(nr, sym) [nr] = (syscall_fn_t)__riscv_##sym,
2
3   /*
4    *创建一个系统调用表
5    *每个表项包括一个函数指针syscall_fn_t
6    */
7   const syscall_fn_t syscall_table[__NR_syscalls] = {
8       __SYSCALL(__NR_open, sys_open)
9       __SYSCALL(__NR_close, sys_close)
10      __SYSCALL(__NR_read, sys_read)
11      __SYSCALL(__NR_write, sys_write)
12      __SYSCALL(__NR_clone, sys_clone)
13      __SYSCALL(__NR_malloc, sys_malloc)
14  };
```

```
long __riscv_sys_open(struct pt_regs *regs)
{
    return sys_open((const char *)regs->a0,
            regs->a1);
}
```


《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

Part 5：实现clone系统调用

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程
《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有
微信公众号：奔跑吧Linux社区

clone系统调用

- clone系统调用常常用于创建用户线程。

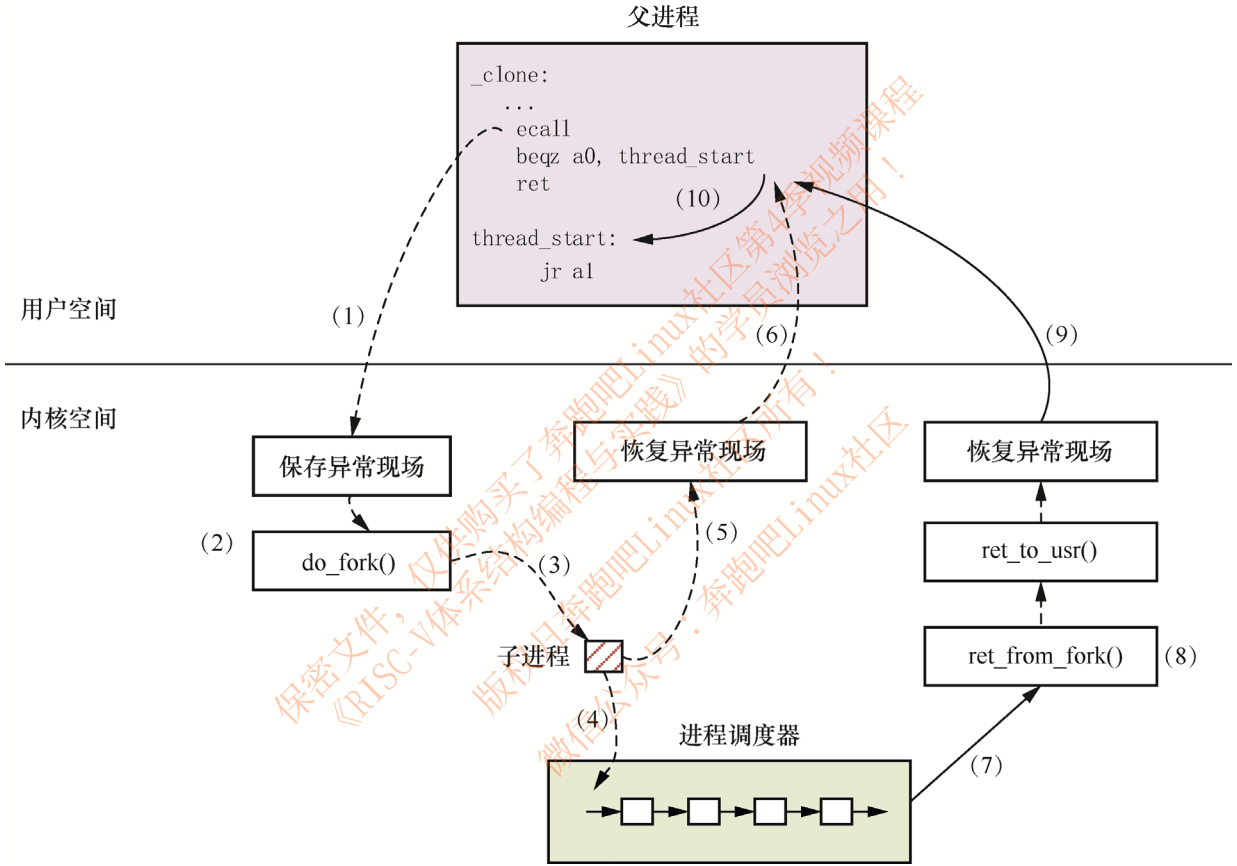
```
int clone(int (*fn)(void *arg), void *child_stack,
    int flags, void *arg)
{

 return __clone(fn, child_stack, flags, arg);
}
```

保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区

<benos/usr/syscall.S>

```
1       .global __clone
2       __clone:
3         /*把fn和arg保存到child_stack的底部*/
4         addi a1, a1, -16
5         sd a0, (a1)
6         sd a3, 8(a1)
7
8         /*调用 syscall*/
9         move a0, a2
10
11         li  a7, __NR_clone
12         ecall
13         beqz a0, thread_start
14         ret
15
16      .align 2
17      thread_start:
18         /*从child_stack取出fn和arg*/
19         ld a1, (sp)
20         ld a0, 8(sp)
21
22         /*调用clone的回调函数fn()*/
23         jalr   a1
24
25         ret
```

版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区



保密文件，仅供购买了奔跑吧Linux社区第4季视频课程《RISC-V体系结构编程与实践》的学员浏览之用！
版权归奔跑吧Linux社区所有！
微信公众号：奔跑吧Linux社区