Lab4-Challenge

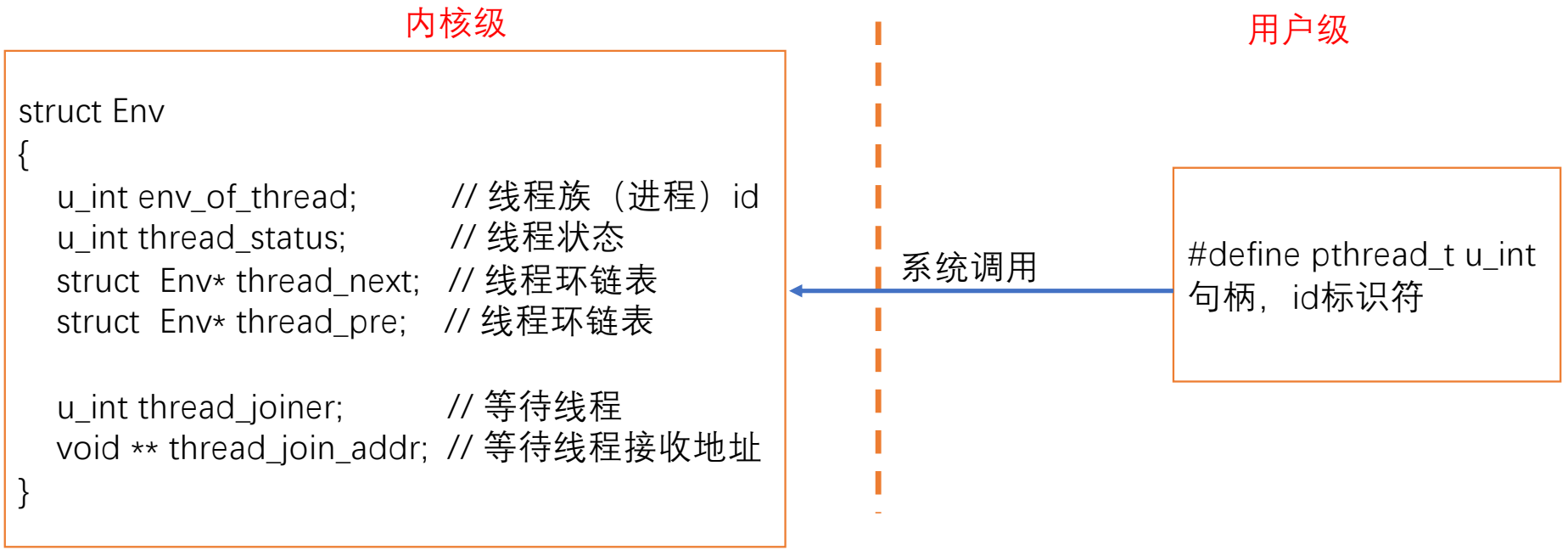
游子诺

Pthread

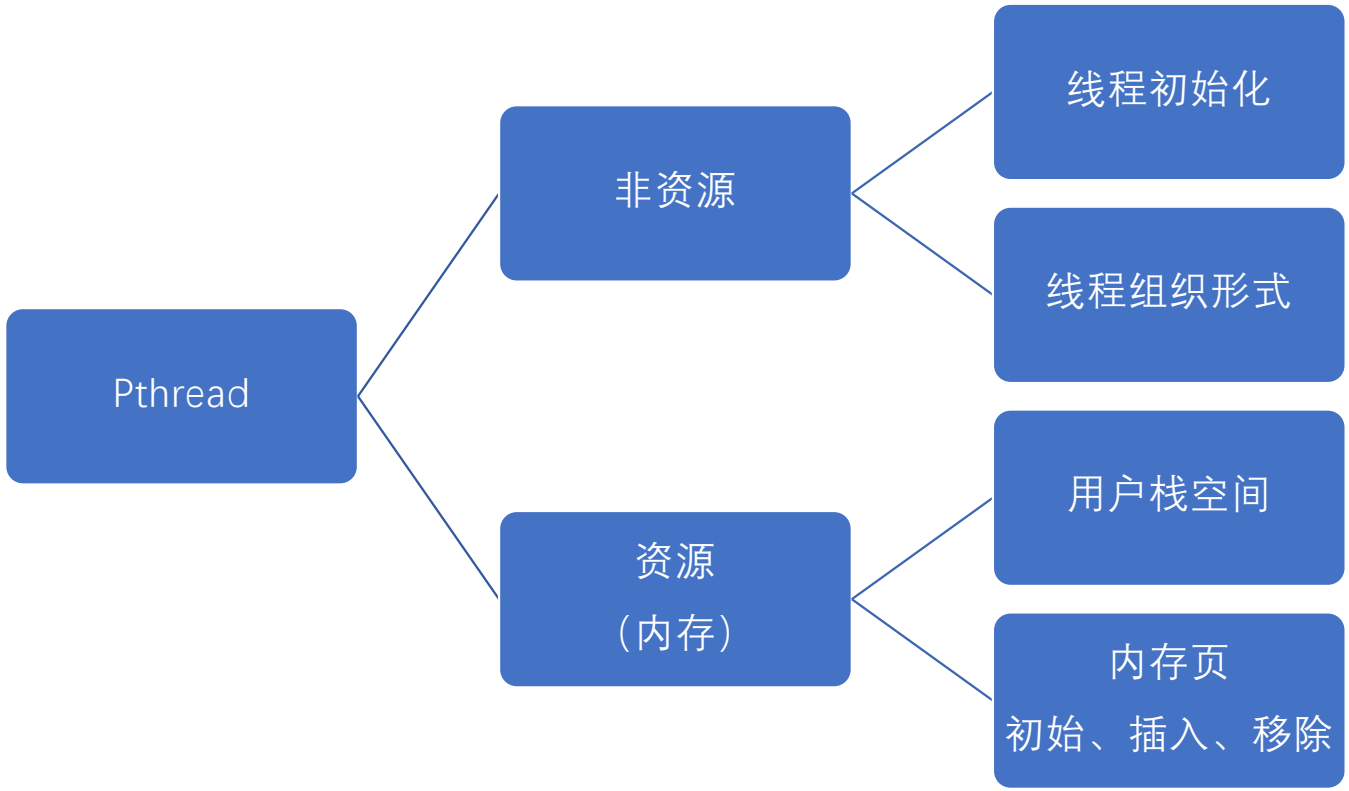
```
int pthread_create(pthread_t* pthread, pthread_attr_t *attr, void*
(start)(void *), void* arg);


void pthread_exit(void* value_ptr);


int pthread_join(pthread_t pthread, void ** retval);


int pthread_cancel(pthread_t pthread);


Int pthread_cancelled(pthread_t pthread);
```

Pthread线程结构

- 基于原有Env结构体实现内核级线程
- 线程用户栈空间限制：16 Kb （可变）



实现要点



非资源部分

线程初始化

入口函数：env -> tf.pc

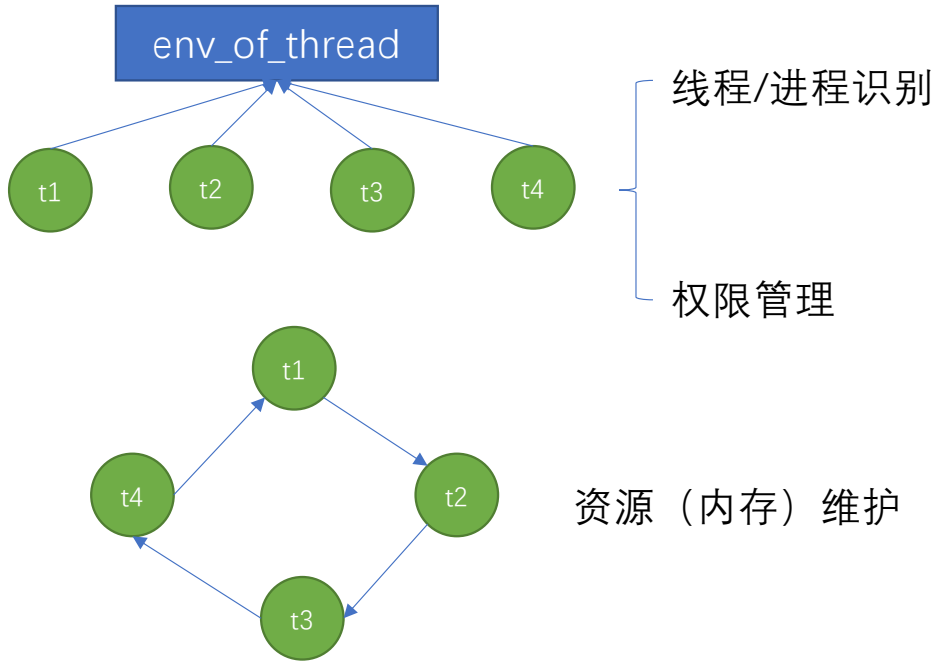
传入值：env -> tf.regs[4]

识别码：

env_of_thread: 进程识别码

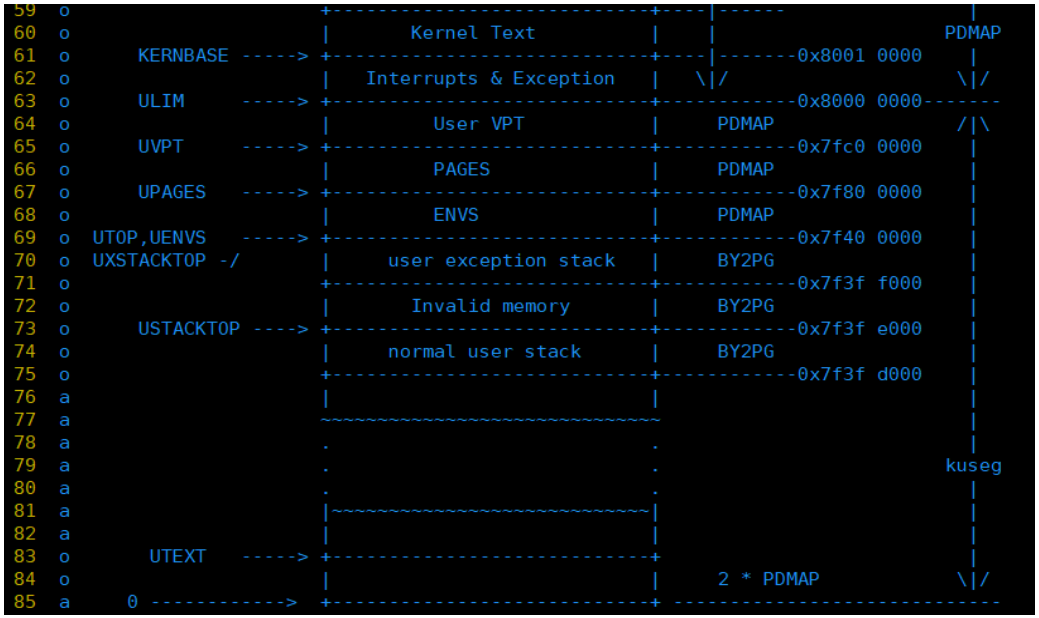
env_id: 线程识别码

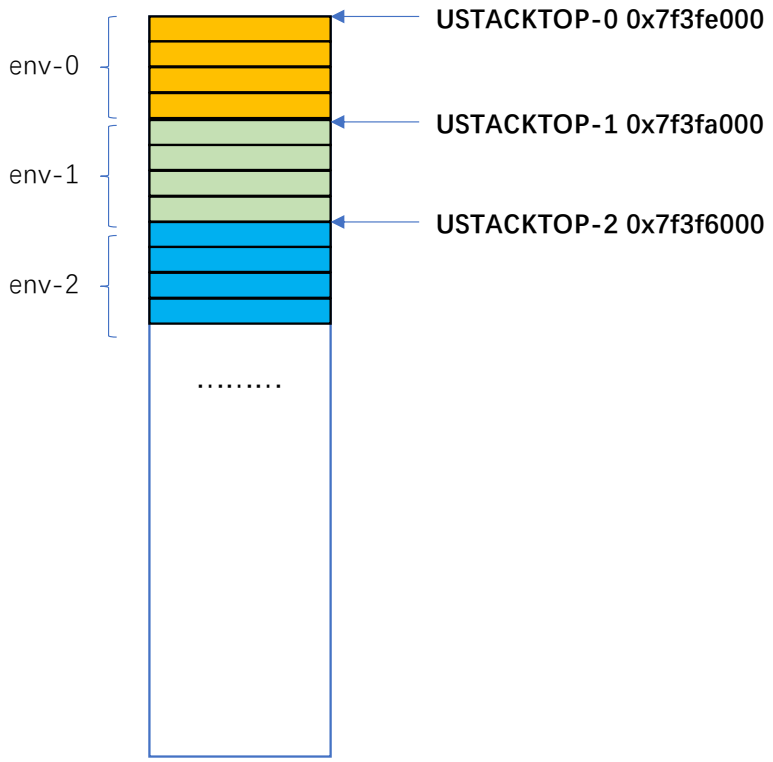
线程组织形式



*主要实现：lib/syscall_all.c: thread_alloc(), thread_exit()

资源部分-用户栈空间





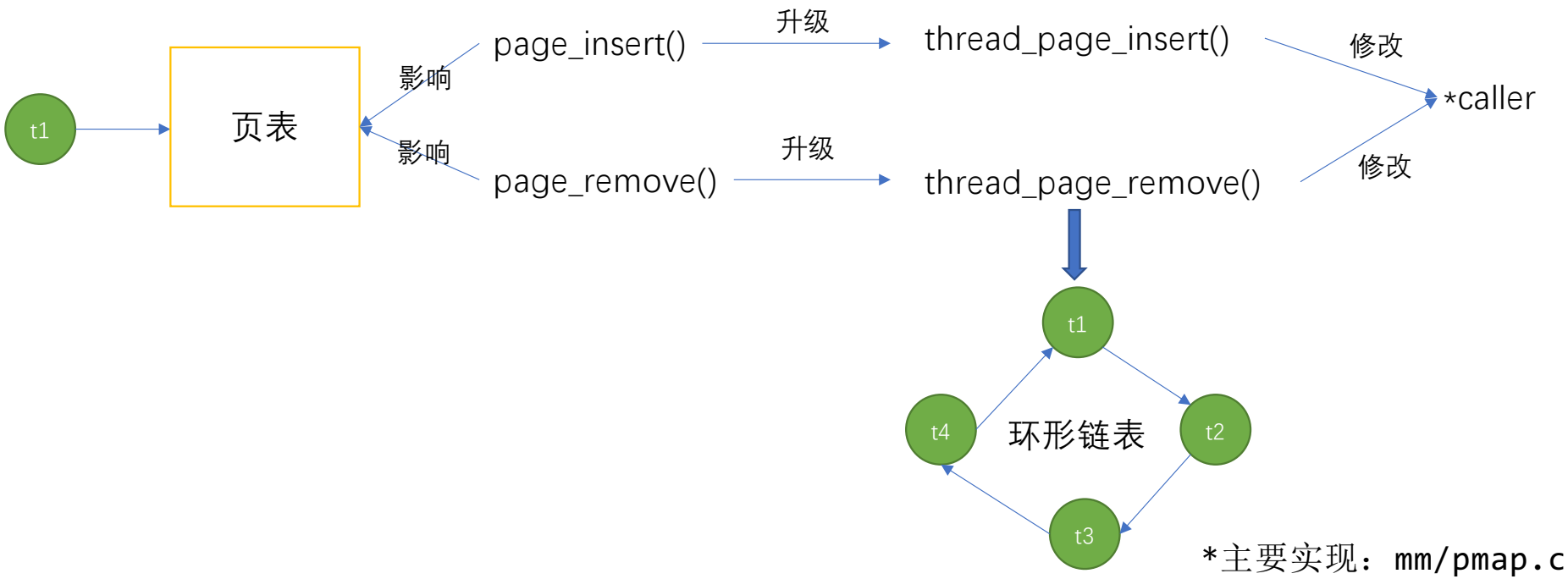
- 线程间全用户空间地址共享（可访问栈内空间）&& 一张虚页只能映射到一个实页
- →
  - Method 1. 所有线程的栈空间需在虚拟内存中。 √
  - Method 2. 通过系统调用得到不处于虚拟地址中的数据。
  - Method X ……

*主要实现：`lib/env.c`

资源部分-内存页的初始、插入、移除

1. 初始化：全用户空间页表复制。（私有栈空间之间需要注意其他线程的读写权限）

2. 插入和移除



*主要实现：mm/pmap.c

Semaphore

```
// non-name semaphore
int sem_init(sem_t * sem, int pshared, int value);
int sem_destroy(sem_t * sem);

// name semaphore
sem_t* sem_open(const char * name, int oflag, mode_t mode, int value);
int sem_close(sem_t sem);
Int sem_unlink(const char* name);

// PV operation
int sem_wait(sem_t * sem);
int sem_trywait(sem_t * sem);
int sem_post(sem_t * sem);
int sem_getvalue(sem_t * sem);
```

Semaphore结构（基础）

user/semaphore.h
user/semaphore.c

```
struct Sem_t
{
    u_int ksem_id;
}
```

系统调用

用户级

内核级

lib/ksemaphore.h
include/ksemaphore.c

```
struct Ksem_t              // kernel semaphore
{
    u_int ksem_value;    // 信号量值
    u_int ksem_id;         // 信号量id
    struct Env* wait_list_head;   // 等待队列
}
```

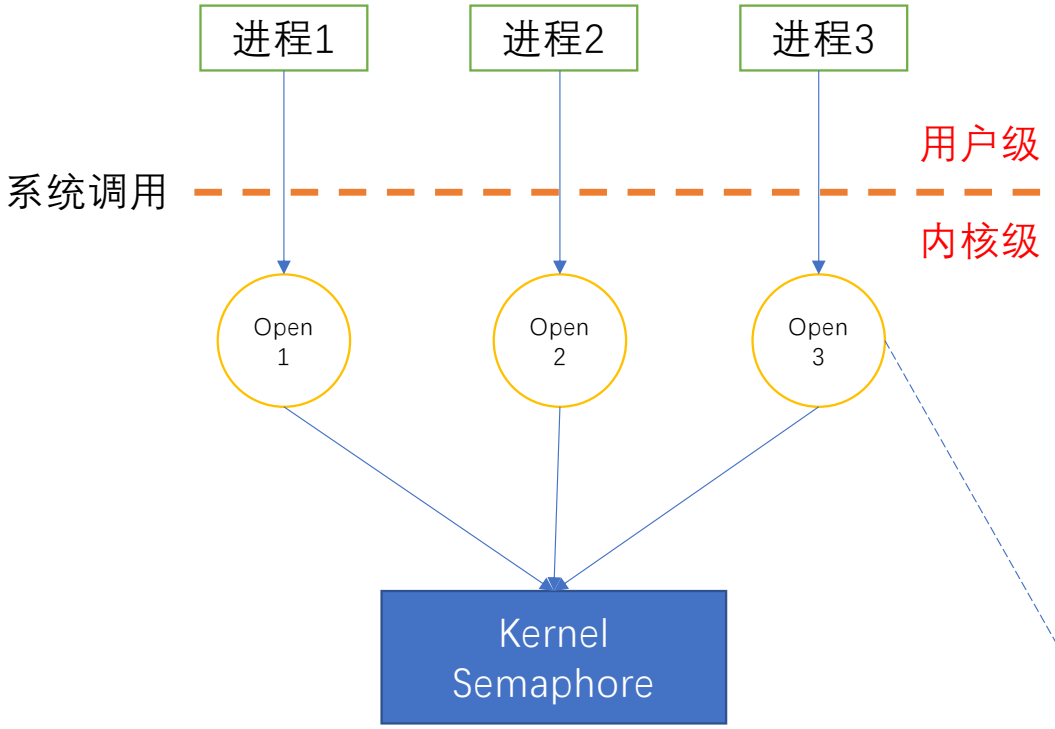
*ksem_id 组织形式类似于env_id

问题：

1. 调度单位必须都能够读取Sem_t，仅限线程之间。（进程？）
2. 权限管理（创毁、开关、操作……）

*主要实现：lib/ksemaphore.c

Semaphore结构（优化）



1. 借助了文件服务进程Open结构体/软链接思路。
2. 实现1：不同进程用户态中Sem_t 操控相同的信号量。
3. 实现2：open结构体专一面向一个进程，有利于结合env_id, mode等信息对信号量操作形成权限保护和与控制。（close、pv操作、getvalue等）

```
struct OpenKsem
{
    struct Ksem* ksem;    // 链接
    Mode_t mode;          // 权限控制（暂未实现）
    int PID;              // open者身份信息（权限控制）
}
```

*主要实现：`lib/ksemaphore.c`

```c
// 信号量类型
#define SEM_FREE 0
#define SEM_NON_NAME 1
#define SEM_NAME 2

// OFLAG
#define O_CREATE 0x0001
#define O_EXCEL  0x0002

// 信号量空闲结构体数量
#define NSEM 64

// 错误返回值
#define E_WRT 1                 //
#define E_NO_FREE_SEM 2         //
#define E_DUP_SEM 3             //
#define E_NOT_FOUND 4           //
#define E_WAIT_FAILED 5         //
#define E_LINKED 6              //
#define E_NO_PERMIT 7           //
#define E_NO_FREE_OPEN 8        //
#define E_WAITING 9             //

```

```c
struct ksem{
    // semaphore type
    int ksem_type;
    int ksem_id;
    int ksem_value;
    struct Env* wait_list_head;             // 线程（进程）等待队列

    // name semaphore - special
    char ksem_name[32];
    int ksem_ref;
    struct ksem * ksem_sched_next;          // sched信号量链表指针
    struct ksem * ksem_sched_pre;
```

库文件(ksemaphore.h)

↑ kernel semaphore结构体（优化）

← 宏定义

ksemaphore.c 实现
←内部(private)方法 →外部(public)方法

```
67  // non-name semaphore
68  // 创建并初始化一个无名信号量
69  int ksem_create(int value, int pshared);
70  int ksem_destroy(int ksem_id);
71
72  // 查找/创建一个有名信号量
73  // 先在链表里通找
74  // 找到 -> 返回 | E_EXCL有效则报错
75  // 找不到 -> 创建 | E_CREATE无效则报错
76  int ksem_open(const char* name, int oflag, Mode_t mode, int value);
77  int ksem_close(int ksem_id);
78  int ksem_unlink(const char* name);
79
80  // PV 操作工具包
81  int ksem_trywait(int ksemid);
82  int ksem_wait(int ksemid);
83  int ksem_post(int ksemid);
84  int ksem_getvalue(int ksemid);
```

```
48  // 初始化
49  void ksem_init();
50
51  // 申请一个信号量结构体
52  int ksem_alloc(struct ksem** ksem, int ksem_type, int ksem_value);
53  // 释放一个信号量结构体
54  void ksem_free(struct ksem* ksem);
55  // 根据ksem_id获取ksem结构体
56  int ksem_get_and_check(struct ksem** ksem, int ksem_id);
57  // 根据name在sched链表中找内核信号量
58  struct ksem* ksem_lookup(const char* name);
59
60  // 申请一个open结构体
61  int ksem_open_alloc(struct OpenKSem** open, struct ksem* ksem, int PID);
62  // 释放一个open结构体
63  void ksem_open_free(struct OpenKSem* open_ksem);
64  // 根据open_id获取open结构体和权限检查
65  int open_get_and_check(struct OpenKSem** open, int open_id);
66
```

谢谢观看，请批评指正！

游子诺