# 内核线程管理

内核线程是一种特殊的进程，内核线程与用户进程的区别

- 内核线程只运行在内核态，而用户进程在用户态和内核态交替运行
- 所有内核线程直接使用共同的ucore内核空间，不需为每个内核线程维护单独的内存空间而用户进程需要维护各自的用户内存空间

线程可看作是共享内存空间的轻量级进程

创建进程控制块->链表管理->调度->分时共享

# get_pid

```
static int
get_pid(void) {
    static_assert(MAX_PID > MAX_PROCESS);
    struct proc_struct *proc;
    list_entry_t *list = &proc_list, *le;
    static int next_safe = MAX_PID, last_pid = MAX_PID;//注意到是static变量
    //越界重新分配，这时之前分配的许多已经被释放
    if (++ last_pid >= MAX_PID) {
        last_pid = 1;
        goto inside;//进去跑循环
    }
    //超过了之前定义的安全区
    if (last_pid >= next_safe) {
    inside:
        next_safe = MAX_PID;
    repeat:
        le = list;
        while ((le = list_next(le)) != list) {
            proc = le2proc(le, list_link);
            if (proc->pid == last_pid) {
            //发现pid已经被使用后的操作很简单，加1即可，后面进行一些个越界判断
                if (++ last_pid >= next_safe) {
                    if (last_pid >= MAX_PID) {
                        last_pid = 1;
                    }
                    next_safe = MAX_PID;
                    goto repeat;
                }
            }
            else if (proc->pid > last_pid && next_safe > proc->pid) {
            //安全区为大于last_pid的最小的那个已经被分配的pid
                next_safe = proc->pid;
            }
        }
    }
```

```
    return last_pid;
}
```

总体来看先按顺序配分1-8091,然后回卷，改变next_safe（下一次小于next_safe的直接返回pid）,找已经被释放的最小的pid返回，next_safe一定程度上减小了搜索的时间消耗，但没有也可以，依次比较就完了

# 2

switch to 用汇编代码写

一方面方便控制寄存器

另一方面 不自动`pushl %ebp movl %esp,%ebp`，感觉像一个黑客

```
.globl switch_to
switch_to: # switch_to(from, to)
# save from's registers
movl 4(%esp), %eax # eax指向from,这涉及到函数调用时，压入参数的顺序，此时esp指向的应该是返回地址
popl 0(%eax) #弹出了返回地址，给到from上下文中的eip
movl %esp, 4(%eax)
……
movl %ebp, 28(%eax)


movl 4(%esp), %eax #之前已经弹出了返回地址，栈顶指针此时指向from，所以+4即为to

movl 28(%eax), %ebp
……
movl 4(%eax), %esp
pushl 0(%eax) #将to上下文中的eip压栈
ret #将栈顶的的eip给到EIP寄存器
```

# [3]

context的eip为forkrets

```
.globl __trapret
__trapret:
//依次弹出
popal

popl %es
popl %ds
//释放中断号和错误码
addl $0x8, %esp
iret
.globl forkrets
forkrets:
```

```
//当前esp指向trap函数的返回地址，+4为参数，即old esp,或者说中断帧
movl 4(%esp), %esp
jmp __trapret
```

上述过程可以很好地与调用trap的过程和trapframe的结构对应