lab4-2 exam

考察 `barrier`，即通过阻塞使得一定数量的进程同步，没过。最大的问题在于在`user/lib/ipc.c` 中的两个函数：

```
void barrier_alloc(int n) {
        syscall_barrier_alloc(n); //之前这里啥都没写！！！
}


void barrier_wait(void) {
        syscall_barrier_wait(); //之前这里啥都没写！！！

}
```

现在再来回顾一下涉及系统调用类题的解题流程：

- 功能函数添加流程是比较套路的，具体步骤为：
    1. 在 `user/include/lib.h` 中添加功能函数的声明；
    2. 在 `user/lib/` 的对应文件中添加功能函数的定义，例如上文中的两个函数在 `user/lib/ipc.c` 中；
    3. 在

       `user/lib/syscall_lib.c`

       中添加对应的系统调用，如：
        - `syscall_barrier_alloc(int n)`
        - `syscall_barrier_wait()`
    4. 在 `kern/syscall_all.c` 中实现内核态系统调用，注册中断函数。一个`sys_*`对应一个`SYS_*`。
- 最后反复检查各部分是否填写完善，最后才是各部分依次debug，切勿死磕一个部分，我这一次最大的问题就是在运行结果错误的时候没有把所有流程过一遍，只是简单地认为内核态错了，直到最后5分钟才发现问题，可惜已经来不及了，差了几秒没交上。实在遗憾。

完整代码：

```
/********** user/include/lib.c **********/

void barrier_alloc(int n);

void barrier_wait(void);

/************ user/lib/ipc.c ************/

void barrier_alloc(int n) {
        syscall_barrier_alloc(n);
}
```

```
void barrier_wait(void) {
        syscall_barrier_wait();

}

/******* user/lib/syscall_lib.c *******/

void syscall_barrier_alloc(int n) {
        msyscall(SYS_barrier_alloc, n);
}

void syscall_barrier_wait(void) {
        msyscall(SYS_barrier_wait);
}

/********* kern/syscall_all.c *********/

/* register iterrupt vector*/

void *syscall_table[MAX_SYSNO] = {
    [SYS_barrier_alloc] = sys_barrier_alloc,
    [SYS_barrier_wait] = sys_barrier_wait,
}

/* implement syscall funtion of kernel state*/

//globar variable

int barrier_size = 0;
int cnt = 0;
struct Env* total[128];

void recover() {
        for (int i = 0; i < cnt; ++i) {
                TAILQ_INSERT_TAIL((&env_sched_list), (total[i]), env_sched_link);
                total[i]->env_status = ENV_RUNNABLE;
        }
}

void sys_barrier_alloc(int n) {
        threads_id = curenv->env_id;
        barrier_size = n;
}

void sys_barrier_wait() {
        if (cnt < barrier_size) {
                if (curenv->env_status == ENV_RUNNABLE) {
                        curenv->env_status = ENV_NOT_RUNNABLE;
                        if (!TAILQ_EMPTY(&env_sched_list)) {
                                TAILQ_REMOVE((&env_sched_list), curenv,
env_sched_link);
                        }
                        total[cnt++] = curenv;
                }
        }
```

```
        if (cnt == barrier_size && barrier_size != 0) {
                recover();
                barrier_size = 0;
                cnt = 0;
        }
        if (cnt < barrier_size) {
                schedule(1); //another process
        }
}
```

奇怪的是，并不需注意进程之间的父子关系，这一份代码就没有考虑，仍然可以通过评测，看来父子关系在测试数据中应该得到了保证。