env_free_list 空闲 env_sched_list[0]，env_sched_list[1] 两个链表调度函数
env_runs;执行次数
env_pgdir : 这个变量保存了该进程页目录的内核虚拟地址。
env_cr3 : 这个变量保存了该进程页目录的物理地址。
env_pri 优先级
env_tf : Trapframe 结构体的定义在include/trap.h 中，在发生进程调度，或当陷入内核时，会将当时的进程上下文环境保存在env_tf变量中。

int envid2env(u_int envid, struct Env **penv, int checkperm) 通过一个env 的id获取该id 对应的进程控制块
env_setup_vm 初始化新进程的地址空间
env_ alloc （进程，父节点id）创建进程
load_icode_mapper 加载该段在ELF 文件中的所有内容到内存，如果该段在文件中的内容的大小达不到为填入这段内容新分配的页面大小，那么余下的部分用0来填充。
load_elf 解析对应的elf文件并读取它进入内存
load_icode 加载elf文件最最开始函数入口 load_icode调用load_elf调用load_icode_mapper
env_create_priority 分配进程空间，load_icode，插入env_sched_list[0]
env_create 调用env_create_priority
env_run(struct Env *e)中断目前进程，执行e，恢复现场继续执行原进程
lcontext(e->env_pgdir);设置全局变量mCONTEXT为当前进程页目录地址，这个值将在TLB重填时用到。
env_pop_tf(&(e->env_tf),GET_ENV_ASID(e->env_id));恢复现场、异常返回。

ENVX(envid) 大致意思就是envid相对于envs(开始的地方)的便宜数目（类似a[i]中i对于a）
UTOP 用户所能操纵的地址空间的最大值；ULIM 操作系统分配给用户地址空间的最大值。

typedef u_long Pde;
typedef u_long Pte;
ROUNDDOWN(va,BY2PG);意思应该是该地址页对齐所在小于该地址的最大位置
bcopy((void *)(bin + i), (void *)page2kva(p), size); 前者赋值给后者

/OSLAB/gxemul -E testmips -C R3000 -M 64 gxemul/vmlinux

page2kva(struct Page **pp*) **得到页pp的虚地址**
page2ppn(struct Page pp*) 得到页pp的物理页号*
*page2pa(struct Page **pp*) 得到页pp的实地址
pa2page(u_long *pa*) 得到实地址pa的页
PPN(*va*) 得到实地址va的物理页号
PDX(va) 可以获取虚拟地址 va 的 31-22 位， PTX(va) 可以获取虚拟地址 va 的 21-12 位。

lab3-1

exam

PART1

在今天的实验里我们要求你实现一个简易的 fork 函数（并不包括实际load代码段），通过给定的原始进程块(输入参数struct Env *e)生成一个新的进程控制块，并返回新进程控制块的env_id。同学们需要在 lib/env.c 和 include/env.h 中分别定义和声明 fork 函数，函数接口如下： u_int fork(struct Env *e); 要求如下：

1. 从 env_free_list 中从头申请一个新的进程控制块
2. 新进程控制块的 env_status、env_pgdir、env_cr3、env_pri和原进程控制块保持一致。

3. 为新进程控制块生成对应的 env_id
4. env_parent_id 的值为原进程控制块的 env_id
5. 返回值为新进程的env_id

PART2

void lab3_output(u_int env_id);

本部分要求修改struct Env，在进程控制块中增加字段（具体增加哪些内容请自行组织）组织起进程间的父子、兄弟关系，并按照要求在 lib/env.c 和 include/env.h 中分别定义和声明 lab3_output 函数输出相关内容，详情如下： 函数lab3_output的定义如下：`void lab3_output(u_int env_id);` 要求输出的内容有其父进程的env_id、其第一个子进程的env_id、其前一个兄弟进程的env_id以及其后一个兄弟进程的env_id 所有的子进程都由fork创建，两个进程如果是兄弟，它们的父进程一定相同。 以某进程第一个子进程是指，由该进程作为父进程使用fork创建的第一个子进程。 兄弟进程间的顺序即为这些进程被创建的顺序，前一个兄弟进程为较早被创建的进程 需要在PART1的fork函数中进行对添加字段的修改 输出格式为：printf("%08x %08x %08x %08x\n", a, b, c, d); 其中a, b, c, d分别为父进程的env_id、第一个子进程的env_id、前一个兄弟进程的env_id以及后一个兄弟进程的env_id 如果a, b, c, d中有不存在的参数，则输出0

PART3

在PART2的基础上，在 lib/env.c 和 include/env.h 中分别定义和声明 lab3_get_sum 函数，函数的功能为：给定一个进程的env_id，返回以该进程为根节点的子进程树中进程的数目（包括它本身），具体接口如下： int lab3_get_sum(u_int env_id);

解答

比较简单，但是小坑不断。有的人第一题没有用`env_alloc()`结果忘记给`free_env_list`remove掉分配的`env`

斗胆放上自己的代码。

首先是修改env结构体，这次exam用数组写起来比较简单。为什么我没用链表呢，因为C语言功底太差，，用指针怕错。。。

```
struct Env {
        struct Trapframe env_tf;        // Saved registers
        LIST_ENTRY(Env) env_link;       // Free list
        u_int env_id;                   // Unique environment identifier
        u_int env_parent_id;            // env_id of this env's parent
        u_int env_status;               // Status of the environment
        Pde  *env_pgdir;                // Kernel virtual address of page dir
        u_int env_cr3;
        LIST_ENTRY(Env) env_sched_link;
        u_int env_pri;
        // Lab 4 IPC
        u_int env_ipc_value;            // data value sent to us
        u_int env_ipc_from;             // envid of the sender
        u_int env_ipc_recving;          // env is blocked receiving
        u_int env_ipc_dstva;            // va at which to map received page
        u_int env_ipc_perm;             // perm of page mapping received

        // Lab 4 fault handling
        u_int env_pgfault_handler;      // page fault state
        u_int env_xstacktop;            // top of exception stack

        // Lab 6 scheduler counts
```

```
        u_int env_runs;                 // number of times been env_run'ed
        u_int env_nop;                  // align to avoid mul instruction

        int son_num;            // add on exam
        u_int son_id_arr[1024]; // add on exam

};
```

```
u_int fork(struct Env *e)
{
        struct Env *e_son;
        env_alloc(&e_son, e->env_id);
        e_son->env_status = e->env_status;
        e_son->env_pgdir = e->env_pgdir;
        e_son->env_cr3 = e->env_cr3;
        e_son->env_pri = e->env_pri;

        // ---- father ----
        int son_num = e->son_num;
        e->son_id_arr[son_num] = e_son->env_id;
        e->son_num += 1;

        return e_son->env_id;
}
```

```
void lab3_output(u_int env_id)
{
        struct Env *e_now;
        u_int fa_id = 0;
        u_int first_son_id = 0;
        u_int bro_bf_id = 0; // "bf" means "before"
        u_int bro_af_id = 0; // "af" means "after"

        envid2env(env_id, &e_now, 0);
        // son part
        first_son_id = e_now->son_id_arr[0];

        // parent part
        fa_id = e_now->env_parent_id;
        if (fa_id == 0) { // do not have parent
                // three 0 now
                bro_bf_id = 0;
                bro_af_id = 0;
        } else { // have a parent
                struct Env *e_fa;
                envid2env(fa_id, &e_fa, 0);
                int index = 0;
                for (index = 0; index < 1024; index++) {
                        if (e_fa->son_id_arr[index] == env_id) {
                                break;
                        } else {
                                continue;
                        }
                }
                // index is the env of father now
                if (index > 0) { // have bro bf
```

```
                        bro_bf_id = e_fa->son_id_arr[index - 1];
                }

                // have a bro_af
                if (e_fa->son_num > index + 1) {
                        bro_af_id = e_fa->son_id_arr[index + 1];
                }
        }
        // fa_id, fist_son_id, bro_bf, bro_af
        printf("%08x %08x %08x %08x\n", fa_id, first_son_id, bro_bf_id,
bro_af_id);
}
```

```
int lab3_get_sum(u_int env_id)
{
        struct Env *e_now;
        envid2env(env_id, &e_now, 0);
        int son_num = e_now->son_num;
        // if e_now has no son
        if (son_num == 0) {
                return 1;
        } else {
                // have many sons, recuring
                int ans = 1;
                int i = 0;
                for (i = 0; i < son_num; i++) {
                        struct Env *e_son;
                        u_int son_id = e_now->son_id_arr[i];
                        envid2env(son_id, &e_son, 0); // now got a son
                        ans += lab3_get_sum(son_id);
                }
                return ans;
        }
}
```

Extra

题目

b/env.c 和 include/env.h 中分别定义和声明 lab3_kill 函数，功能为：杀死一个进程。进程树的根节点代表的进程收养（接管）他的孤儿进程。这些孤儿进程将依次排列在根进程的子进程的尾部。测试中保证不会杀死根节点代表的进程。函数接口如下：`void lab3_kill(u_int env_id);`在完成一次删除后，基础测试PART2 lab3_output，输出的结果在新的树结构下仍应正确。 一颗进程树的根节点代表的进程由env_alloc创建，其余节点代表的进程皆由PART1 fork创建。 本部分中"杀死进程"的要**清空进程控制块相关字段**（可以参考`free_env()`，但是不能调用`free_env()`）以及其他相关处部分测试较强，请务必考虑完善。

解答

一堆人以为自己的数据结构出问题了，结果，，是在杀死一个进程后的状态处理上出问题了。

我一push，得11分，呵呵，也跑去de自己结构的bug了，不过已经没时间了。

早知道exam拿60分就跑路了，现在发现原来exam60和100最终得分没差。麻了，战术失误。

看了大佬的博客，发现自己可能是没有free后插回list里，我呵呵了😇。

```
void lab3_kill(u_int env_id)
{
        // get e_now and e_root
        struct Env *e_now;
        struct Env *e_root;
        envid2env(env_id, &e_now, 0);

        u_int root_id = e_now->env_parent_id;
        envid2env(root_id, &e_root, 0);

        // while this "root" have a father
        while (e_root->env_parent_id != 0) {
                root_id = e_root->env_parent_id;
                envid2env(root_id, &e_root, 0);
        }

        // now we get the root
        int now_son_num = e_now->son_num;
        int root_son_num = e_root->son_num;

        int i = 0;

        // remove from father, careful!: root == father
        struct Env *e_fa;
        u_int fa_id = e_now->env_parent_id;
        envid2env(fa_id, &e_fa, 0);
        // get the index
        for (i = 0; i < e_fa->son_num; i++) {
                if (e_fa->son_id_arr[i] == env_id) {
                        break;
                } else {
                        continue;
                }
        }

        // cover
        for (; i < e_fa->son_num-1; i++) {
                e_fa->son_id_arr[i] = e_fa->son_id_arr[i+1];
        }
        e_fa->son_id_arr[i] = 0;
        e_fa->son_num--;


        // attach all sons to root
        for (i = 0; i < now_son_num; i++) {
                u_int now_son_id = e_now->son_id_arr[i];
                e_root->son_id_arr[e_root->son_num] = now_son_id;
                e_root->son_num += 1;
        }

        // free the env
        e_now->son_num = 0;
        for (i = 0; i < e_now->son_num; i++) {
                e_now->son_id_arr[i] = 0;
        }
        e_now->env_status = ENV_FREE;
        e_now->env_cr3 = 0;
        e_now->env_pgdir = 0;
```

```
        // maybe i should add, then i can pass
        LIST_INSERT_HEAD(&env_free_list, e, env_link);

}
```

2019

```
u_int newmkenvid(struct Env *e, int pri)
{
    u_int idx = e -envs;
    return (pri << (1 + LOG2NENV)) | idx;
}

void output_env_info(int envid)
{
    static int cnt_times = 0;
    cnt_times++;
    int pri = envid >> (1 + LOG2NENV);
    printf("no=%d,env_index=%d,env_pri=%d\n", cnt_times, ENVX(envid), pri);
}

void init_envid()
{
    int i;
    struct Env *e;
    for (i = 0; i < NENV; i++)
    {
        e = &envs[i];
        if (e->env_status == ENV_RUNNABLE)
        {
            e->env_id = newmkenvid(e, e->env_pri);
        }
    }
}

int newenvid2env(u_int envid, struct Env **penv, int checkperm)
{
        struct Env *e;
    /* Hint:
 *      *  If envid is zero, return the current environment.*/
        if (envid == 0)
        {
            *penv = curenv;
            return 0;
        }
    /*Step 1: Assign value to e using envid. */
        e = envs + ENVX(envid);

        if (e->env_status == ENV_FREE || e->env_id != envid) {
                *penv = NULL;
                return -E_BAD_ENV;
        }
    /* Hint:
 *      *  Check that the calling environment has legitimate permissions
 *           *  to manipulate the specified environment.
 *                *  If checkperm is set, the specified environment
```

```
 *                     *  must be either the current environment.
 *                        *  or an immediate child of the current
environment.If not, error! */
    /*Step 2: Make a check according to checkperm. */
        if(checkperm)
        {
            if (e != curenv && e->env_parent_id != curenv->env_id)
            {
                *penv = 0;
                return -E_BAD_ENV;
            }
        }

        *penv = e;
        return 0;
}
```