Lab3-extra

测试说明

- 本次测试题没有额外分支，需要自己创建。
- 本次测试分为基础测试和进阶测试两个部分。
- 基础测试和进阶测试是独立的两道题目，进阶测试不需要使用到基础测试的代码。
- **在开始动手写代码前，请仔细阅读每个部分的题目和每个部分后的注意事项！**

完成与提交实验的方式

1. 使用 git branch 检查并**确保自己在lab3分支**下
2. 将本地改动提交，避免分支切换造成混乱(add与commit操作)
3. git checkout -b **lab3-extra # 新建并切换到 lab3-extra** 分支下
4. **在lab3-extra分支下完成实验内容**
5. git add --all 或 git add -A
6. git commit -m 'balabala'
7. git push origin lab3-extra:lab3-extra

Step 1: lab3-extra基础测试

- 题目要求：
- STAGE1: 在MOS操作系统中进程的 id 是根据下图的方式生成的，既保证了每个进程 id 的唯一性，
- 同时也可以通过进程的 envid 找到该进程对应的 Env 结构体（envid2env 函数的作用）。

```
u_int mkenvid(struct Env *e)
{
        static u_long next_env_id = 0;

    /*Hint: lower bits of envid hold e's position in the envs array. */
        u_int idx = e - envs;

    /*Hint:  high bits of envid hold an increasing number. */
        return (++next_env_id << (1 + LOG2NENV)) | idx;
}
```

现在请同学们自行设计一种生成进程的 envid 的方法，使得生成的 id 满足下面要求：

1. 任何两个不同的进程 id 不能相同，即保证进程 id 的唯一性
2. 通过进程的 id 能够找到该 id 对应的 Env 结构体在 envs 数组中的索引位置
3. 进程的 id 中包含该进程的优先级信息（假定进程的优先级包括 0~15，共16种）

Step 1: lab3-extra基础测试

- 题目要求：

- 同学们一共需要在 lib/env.c 中编写两个函数并在 include/env.h 中声明这两个函数，函数的接口如下：

  1. u_int newmkenvid(struct Env *e, int pri)
  2. void output_env_info(int envid)

其中 newmkenvid 代替了原来的 mkenvid 函数的作用，用来生成进程的 envid，并且需要满足上述 4 个条件。

output_env_info用于输出一个envid代表的信息，格式为“no=%d,env_index=%d,env_pri=%d\n”。

no代表是第几次调用这个函数，从1开始计数。

env_index代表该进程控制块在envs中的偏移。

env_pri代表该进程的优先级。

Step 1: lab3-extra基础测试

- 题目要求：
- STAGE2：同学们需要在 lib/env.c 和 include/env.h 中定义、声明 init_envid 函数和newenvid2env函数。
- 编写 init_envid 函数，要求遍历 Envs 数组，对所有状态为 ENV_RUNNABLE 的进程利用 newmkenvid 得到该进程的 envid，并将该值存到这个进程控制块的 env_id 域里。
- newenvid2env 则是根据输入的 envid 解析出对应的进程控制块。这部分函数只用在课下实现的 envid2env 基础上修改解析进程块在数组中的索引方式即可。
- 函数接口如下：

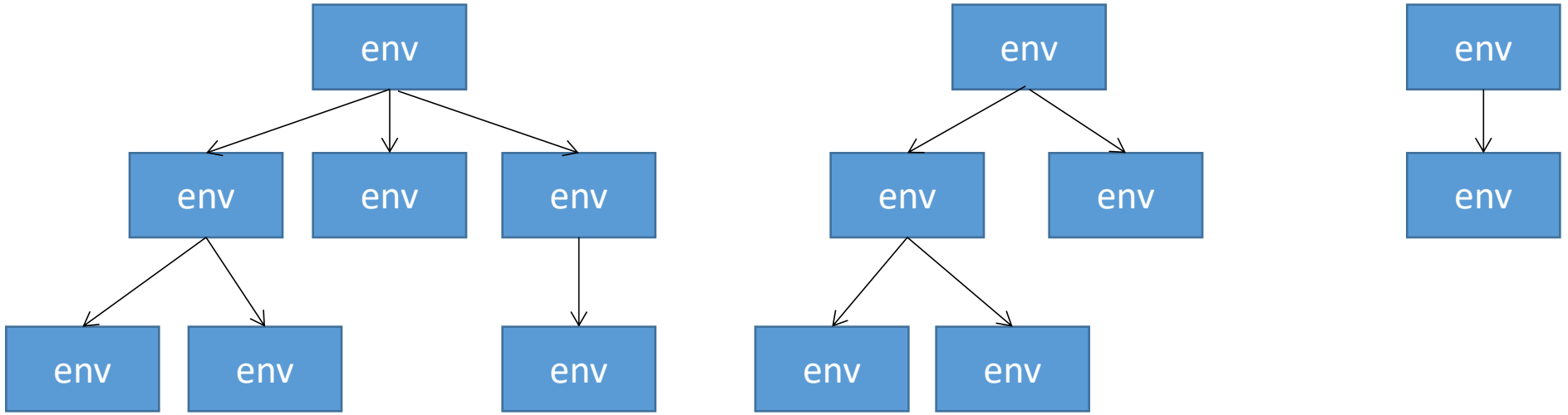
```
void init_envid();
int newenvid2env(u_int envid, struct Env **penv, int checkperm)
```

Step 1: lab3-extra基础测试

- 题目要求：
- 测试有以下 4 个部分：
- 1. 课下强测 15 分
- 2. newmkenvid 和 output_env_info 20 分
- 3. init_envid 和 newenvid2env 25 分
- 注意：无论完成哪一部分的实验，都需要在 lib/env.c 和 include/env.h 中定义、声明以上 4 个函数否则将无法编译成功。
- 注意：output_env_info输出进程的优先级信息pri的时候，不可以直接输出该进程控制块的env_pri，需要通过envid来获取该进程的pri信息。

Step 2：lab3-extra进阶测试

- 题目要求：
- 在MOS操作系统中，所有的进程依据父子关系形成了一棵棵树，如下图所示：



Step 2: lab3-extra进阶测试

- 题目要求：

- 现在我们要实现两个函数
    1. int check_same_root(u_int envid1, u_int envid2)
    2. void kill_all(u_int envid)
- 对于 check_same_root 函数，传入两个进程的 envid，根据以下要求返回结果：
    1. 如果传入的两个进程至少有一个 env_status 为 ENV_NOT_RUNNABLE，则返回 -1
    2. 如果传入的两个进程不在同一棵进程树上且都不为 ENV_NOT_RUNNABLE，则返回 0
    3. 如果传入的两个进程在同一棵进程树上且都不为 ENV_NOT_RUNNABLE，则返回 1
- 对于 kill_all 函数，根据以下要求进行操作：

    1. 如果 envid 所在进程树上有至少一个进程的 env_status 为 ENV_NOT_RUNNABLE 则打印“something is wrong!\n”，不对进程块做任何操作。

    2. 如果 envid 所在进程树上的进程都不为 ENV_NOT_RUNNABLE，则将该进程树上所有进程状态改为 ENV_NOT_RUNNABLE。

注意这两个函数都会被多次调用，因此要注意**初始化**

Step 2：lab3-extra进阶测试

- 题目要求：
- 分数占比：
- check_same_root 10 分
- kill_all 30 分：
  - 第一个测试点:10分
  - 第二个测试点：20分
- 注意：

1. kill_all 传入的 envid 不一定是进程树的根节点。
2. 需要操作的进程不是以该 envid 为根的子树，而是 envid 所在的整个进程树。
3. 完成该部分实验，需要在 lib/env.c 和 include/env.h 中定义、声明以上2个函数，否则将无法编译成功。
4. 我们在测试kill_all函数时会限定时间，所以你需要尽可能使用时间复杂度低的实现方法。

Good Luck!