Lab3-1

测试说明

考试时间 14:00 ~ 16:00

测试题目分为基础测试和附加测试(选做)两部分

每题单独评分，满分都是 100 分

请注意，Lab 得分为：Lab 基础分值 * (课下成绩 * 0.6 + 课上 exam 成绩 * 0.4) / 100

附加测试加分为：通过（ >= 60 分）课上测试 Extra 题目所给予的加分

lab3-1-exam

Step1: 创建并切换 lab3-1-exam 分支

```
cd ~/学号/
git checkout lab3
git add .
git commit -m "xxxxx"
git checkout -b lab3-1-exam   （注意：有参数-b）
```

Step2: 完成 lab3-1-exam 代码编写

题目背景

当操作系统将程序加载到内存时，为了系统的安全性，操作系统会去监控加载程序的各种信息以确保能正确加载。

你的任务就是完成对加载程序信息的分析。

进程信息分析

请你修改函数 `env_create_priority` ，在执行完函数 `load_icode(e, binary, size)` 后输出加载程序的入口地址、在内存中的大小和内存中的页面数量。

输出格式如下所示：

```
printf("start = %x, memory = %x, pages = %d\n", ...);
```

测试样例

标准输入

`init/init.c` 文件的 `mips_init` 函数修改为

```
void mips_init() {
    mips_detect_memory();

    mips_vm_init();
    page_init();

    env_init();
    ENV_CREATE_PRIORITY(user_A, 2);

    *((volatile char*)(0xB0000010)) = 0;
}
```

标准输出

```
start = 4000b0, memory = cb2, pages = 1
```

本题评测方式

总共会测试5个进程的加载，类似上图的方式，每个进程20分。

Step3: 提交评测

```
cd ~/学号/
git add .
git commit -m "balabala..." （请将balabala改为有意义的信息）
git push origin lab3-1-exam:lab3-1-exam
```

lab3-1-Extra

Step1: 创建lab3-1-Extra分支

通过基础题后，可创建lab1-Extra分支。

请注意：**lab3-1-Extra 和 lab3-1-exam 是独立的，即使没有通过 lab3-1-exam 也可以去尝试完成 lab3-1-Extra 。**

```
cd ~/学号/
git checkout lab3
git add .
git commit -m "xxxxx"
git checkout -b lab3-1-Extra
```

Step2: 完成lab3-1-Extra代码编写

题目背景

操作系统中大量使用 Fork 来创建新的进程，Fork 前后的进程互为父子关系。

在本题中你需要实现简易的 Fork 来确保能有效维护进程之间的关系。

步骤1

在 `env.c` 中创建函数 `fork` ，函数原型如下所示：

```
struct Env* fork(struct Env* e, int pri);
```

表示由进程 `e` 创建出一个新的子进程， `pri` 表示子进程的优先级，如果 `pri > 0` 则表示新创建子进程的优先级等于 `pri` ，否则子进程直接继承父进程的优先级。

在创建新进程时你**不需要**为新进程准备页表（不需要 `env_setup_vm` ），只需要申请对应的进程控制块即可。注意要对父进程编号、进程状态（设置为可运行）、本进程编号、优先级这四个域进行赋值。

步骤2

在 `env.c` 中创建函数 `getParentCount` ，函数原型如下所示：

```
int getParentCount(struct Env* e);
```

表示计算进程 `e` 有多少个父进程，这里的父进程包括直接的父进程和间接的父进程（父进程的父进程们）。

若进程状态不合法则需要直接返回 -1。

步骤3

在 `env.c` 中创建函数 `getChildCount` ，函数原型如下所示：

```
int getChildCount(struct Env* e);
```

表示计算进程 `e` 有多少个子进程，这里的子进程包括直接的子进程和间接的子进程（子进程的子进程们）。

若进程状态不合法则需要直接返回 -1。

提示

1. 注意在 `env.h` 添加这些函数的声明。
2. 进程状态不合法当且仅当进程不为可运行状态。
3. `fork` 函数可以参考 `env_alloc` 的创建，在本题中你不需要管所有有关内存的申请和释放，只需要关注于进程本身即可。
4. 注意步骤2和3中**间接**的这一词。
5. 本题中实现是一个简易的 `fork` ，如果希望父子进程可以跑起来需要做更多的处理，感兴趣的同学们可以去查询 `Linux` 中 `fork` 是如何实现的。
6. 在评测时我们不关心你的 `env_id` 是否正确，而会关注于进程控制块 `e` 之间的关系是否正确。

关于数据

1. 对于前 60% 的测试数据，仅包含 `fork` 和 `getParentCount` 函数。
2. 对于所有测试数据，进程的总数量不超过 64 个。

测试样例

标准输入

`init/init.c` 文件的 `env_fork_test` 函数和 `mips_init` 函数修改为

```
void env_fork_test() {
    struct Env *root;

    if (env_alloc(&root, 0) < 0) {
        printf("First env alloc error!\n");
      return;
    }

   struct Env* child = fork(root, -1);

   if (child->env_parent_id != root->env_id) {
        printf("Env parent id error\n");
      return;
    }

   printf("Accept!\n");
}

void mips_init()
{
    mips_detect_memory();

    mips_vm_init();
    page_init();

    env_init();

    env_fork_test();

    *((volatile char*)(0xB0000010)) = 0;
}
```

标准输出

```
Accept!
```

Step3: 提交评测

```
cd ~/学号/
git add .
git commit -m "balabala..." （请将balabala改为有意义的信息）
git push origin lab3-1-Extra:lab3-1-Extra
```