Lab6 挑战性任务

在 Lab6 挑战性任务中，**Easy 部分和 Normal 部分为必做，还需要从两个 Challenge 部分中任选一个完成。**

在 Lab6 基础部分的实验中我们实现了一个简单的 shell。在本次挑战性任务，我们将通过实现一些难度层次递进的小组件或附加功能，来丰富我们的 shell，从而加深对整个 MOS Lab6 的理解，让我们的 MOS 更加完整。

> 前置要求：实现 Lab5 的 `fsformat` 中"文件夹生成"的相关代码。

Easy 部分

(1) 实现后台运行

在一个命令后增加 `&` 符号，使得 shell 不需要等待此命令执行完毕后再继续执行，而是将此命令置于后台执行，此时可在 shell 继续输入新命令。

> 提示：目前的 shell 在等待输入时，是在内核态忙等，需要对此进行修改才能实现后台运行。

(2) 实现一行多命令

用 `;` 分开同一行内的两条命令。

> 提示：我们保留 symbol 里已经预留有 `;` 和 `&` 字符。

(3) 实现引号支持

实现引号支持后，shell 可以处理如： `echo.b "xxx | xxx"` 这样的指令。即 shell 在解析时，会将双引号内的内容看作**一个**字符串。

(4) 实现如下命令

- `tree`
- `mkdir`
- `touch`

Normal 部分：历史命令功能

任务背景

在 Linux 下我们输入的 shell 命令都会被保存起来，并可以通过 Up / Down 键回溯指令，这为我们的 shell 操作带来了极大的方便。

任务目标

实现保存在 shell 输入的指令，并可以通过 `history.b` 命令输出所有的历史指令以及通过上下键回溯指令。

任务要求

第一部分：要求我们将在 shell 中输入的每步指令，在解析前/后保存进一个专用文件（如 `.history`）中，每行一条指令。

第二部分：通过编写一个用户态程序 `history.b` 文件并写入磁盘中，使得每次调用 `history.b`时，能够将文件（`.history`）的内容全部输出。

第三部分：键入上下键时，切换历史命令（与 Linux 的上下键行为一致）。

注意：

- 禁止使用局部变量或全局变量的形式实现保存历史指令。（这意味着不能用堆栈区保存历史指令）
- 禁止在烧录 `fs.img` 时烧录一个 `.history` 文件。这意味着你需要在第一次写入时，创建一个 `.history` 文件，并在随后每次输入时在 `.history` 文件末尾写入。

需要关注的核心文件：

- `sh.c`：需要在命令执行前后把命令行写进文件
- `history.c`：一个简单的用户态程序
- 其他自行设计的数据结构

Challenge 部分：exec

任务背景

在 Lab6 的实验中，我们使用 `spawn` 函数实现了生成新进程（将目标程序加载为新的进程），这帮助我们实现了 shell。在 `spawn` 中，我们将目标程序的相关信息加载给新进程，此时目标程序所在的新进程和原进程的 `envid` 是不同的。

另一种运行目标程序的方法是调用 `exec` 函数族，在 Linux 中有 6 个以 `exec` 开头的函数，请同学们查找资料，理解 `exec` 系列函数。

`exec` 函数族并没有创建新的进程，它所做的是将原进程的代码段、数据段、堆栈段等用目标程序代替，但进程的 `envid` 并没有被替换。

任务目标

1. 为了简化问题，我们仅要求实现 `exec` 函数族的简单版本。

   即我们要求实现函数 `int exec(char *path, char **argv)`。其中 `path` 为目标程序的路径，`argv` 为参数数组。该函数将目标程序加载到本进程，并开始执行目标程序。

   若 `exec` 执行成功，则不会返回，否则返回 `-1`。

2. 将 shell 改写为通过 `exec` 加载目标程序。

   修改前的 shell 在接收到命令时，会先 fork 出**子 shell**，然后**子 shell** 调用 **spawn** 加载目标程序。**修改后**我们希望**子 shell** 可以通过 **fork** 与 **exec** 的配合来加载目标程序并完成 shell 的基本功能。

参考与提示

- 在执行 `exec` 时不能“左脚踩右脚”，需要借助**其它服务进程**或**内核态**才能帮助我们将原进程替换。我们推荐两种方式来实现 `exec`：
  - 方式一：借助**其它服务进程**。类似于文件系统进程，创建一个管理进程 `system` 进程，将 `exec` 的参数信息通过 **IPC** 的方式发给 `system` 进程，由 `system` 进程辅助原进程完成 `exec` 操作，即由 `system` 进程将代码段和数据段等加载到原进程的地址空间。

- 方式二：借助**内核态**。`exec` 内部通过**系统调用**的方式将参数传递给内核态，由内核态辅助原进程完成 `exec` 操作，即由内核态将代码段和数据段等加载到原进程的地址空间。在这种方式下，你需要解决的难点是如何在内核态下访问文件系统服务。

Challenge 部分：实现 shell 环境变量

任务目标

1. 支持 `declare [-xr] [NAME [=VALUE]]` 命令，其中：
   - `-x` 表示将变量 `NAME` 导出为环境变量，否则为局部变量。
     - 环境变量对子 shell 可见，也就是说在 shell 中输入 `sh.b` 启动一个子 shell 后，可以读取 `NAME` 的值。
     - 局部变量对子 shell **不可见**，也就是说在 shell 中输入 `sh.b` 启动一个子 shell 后，没有该局部变量。
   - `-r` 表示将变量 `NAME` 设为只读。只读变量不能被 `declare` 重新定义或被 `unset` 删除。
   - 如果没有 `[-xr]` 及 `[NAME [=VALUE]]` 部分，则输出当前 shell 的所有变量，包括局部变量和环境变量。
2. 支持 `unset NAME` 命令，若 `NAME` 不是只读变量，则删除 `NAME` 。
3. 支持并在执行诸如 `echo.b $variable` 指令时能显示正确的值。

参考与提示

- `declare` 命令的详细信息可以在 Linux 系统中输入 `declare --help` 查看。
- `unset` 命令的详细信息可以在 Linux 系统中输入 `unset --help` 查看。
- 本部分开放性很强，可以参考 Linux 系统的环境变量。建议用内建指令实现 `declare`、 `unset` 指令的简易版本。

提交要求

- 在 Lab6 挑战性任务中，**Easy 部分和 Normal 部分为必做，还需要从两个 Challenge 部分中任选一个完成。**实现上述任务后，请自行设计展示（需要在申优答辩时展示效果并陈述实现流程）。
- 请自行建立 `lab6-challenge` 分支，在该分支完成代码后，push 到个人的远程仓库。代码内需要包含对于功能的详细测试程序，测试程序本身及运行测试程序得到的运行结果应具有足够的可读性。

```
git checkout lab6
git add .
git commit -m "xxxxx"
git checkout -b lab6-challenge
# 完成代码
git push origin lab6-challenge:lab6-challenge
```

- 实验报告请提交至 SPOC 系统，在撰写**实验报告**和准备**申优答辩**时，请包含以下内容：
  - 对于任务的实现思路，并配合关键代码进行说明。
  - 对于功能的详细**测试程序**，以及运行测试程序得到的运行结果。
  - 完成挑战性任务过程中**遇到的问题及解决方案**。