JOS_Lab5_Report

姓名：方嘉聪，学号：2200017849

I. Challenge: Add more features to the shell

实现了最后一个Challenge，主要在 `user/sh.c` 中进行修改，具体如下：

1. multiple commands per line `ls; echo hi`：将原有的 `runcmd` 重命名为 `run_single_cmd`，用以执行单步命令（注意需要将最后的 `exit()` 去掉）。新定义 `runcmd` 通过识别 `;` 将命令划分，对每个命令创建一个子进程调用 `run_single_cmd` 以执行。
2. backgrounding commands `ls &/echo &`：在 `run_single_cmd` 中模仿pipe `|`，引入一个新case `&`，并定义一个background变量，来记录是否为后台执行状态。模仿 MacOS Terminal，设置了进入和结束的输出。
3. environment variable expansion `echo $HOME`：定义了全局的结构体和数组来存储环境变量，实现了 `expand_env_vars()` 将 `$varname` 替换为对应的环境变量值。 并在 `run_single_cmd` 将字符串 `s` 进行进一步分析前，调用 `expand_env_vars()` 完成替换。在 `umain()` 中预设了一些环境变量。

```
typedef struct{
        char name[BUFSIZ];
        char value[BUFSIZ];
} EnvVar;

static EnvVar env_vars[MAX_ENV_VARS];
static int env_var_count = 0;
```

4. quoting `echo "a | b"`：在 `SYMBOLS` 宏中添加 `'"'`，在 `_gettoken()` 中定义 `is_quotes` 变量来判断是否为quote字符，在具体实现中进行相应判断即可。
5. command-line history and editing：添加了一个数组用以存储命令的历史，实现了 `add_to_history()` 函数，用以将命令添加到这一数组中。此外重写了 `lib/readline.c/readline()`(命名为 `read_line()`)，支持使用上下箭头实现命令历史的切换。同时修改backspace处理，使得能够将删除的字符从shell显示中删去。此外引入 `cur_pos` 来记录目前的光标位置，用以支持光标的左右移动。

```
char history[MAX_HISTORY][BUFSIZ];
int history_count = 0;
int history_index = 0;
```

> **Warning**
>
> 1. 由于时间仓促，没有来得及实现其他的命令，如 tab 补全和 file create。
> 2. 在 environment variable expansion 中还尝试添加了 `setenv` 命令在 shell 中设置环境变量，但是由于 `runcmd` 通过创建新的子进程来调用 `run_single_cmd`，由于 `env_vars` 不在进程间共享，也没有找到JOS其他可以比较方便修改的方法。
> 3. 在新实现的 `read_line()` 中和真正实用的 Terminal 还有很大距离，可能还有一些的bug（特别是光标移动和中间修改），等有空余的时间再进一步完善。

测试结果见下，`echo` 替换成 `ls` 也能正常运行，只是由于 `ls` 输出比较多，截图跨度会比较大:(

```
$ echo 1
1
$ echo hello; echo world!
hello
world!
$ echo $HOME
/
$ echo "a|b"
a|b
$ echo hello; echo world!
hello
world!
$ echo "a|b"
```

在完成后重新 `make grade`，未影响test通过。

II. Exercises

1. Exercise 1

按照提示在`env_create`中添加类型判断并给予file system I/O 权限即可。
在开始exercise之前，在merge后进行lab4的测试，发现有许多无法通过，是由于lab5 merge时将测试需要检验的一些`cprintf`输出去掉了。。

Question 1

无需任何操作，I/O权限会被存储在 `eflags` register 中，在环境切换过程中会被自动保存和恢复。

2. Exercise 2

按照提示实现 `bc_pgfault` 和 `flush_block`，今儿实现 demand paging，模仿先前实现的 page fault。注意需要先保证地址对齐：`addr = ROUNDDOWN(addr, PGSIZE);`

Question for reader

why do we do this(check the block) *after* reading the block in?
将block读入，再读取块内容并与bitmap进行比对（是否被使用），可以用以判断一致性。

3. Exercise 3

模仿 `free_block` 实现 `alloc_block`即可。

4. Exercise 4

按照提示完成 `file_block_walk` 和 `file_get_block`。

- 前者将对应 `blockno` 所在的entry（包含direct 和 indirect）的地址存储在 `**ppdiskbno` 中返回。
- 后者通过调用 `file_block_walk` 得到entry的地址，注意到该地址中存储着 block 的编号，读取后利用宏 `diskaddr(*pdiskbno)` 得到对应的虚拟地址即可
- 注意：如果`File`结构体内的`indrect`未分配，需要调用 `alloc_block()`

5. Exercise 5&6

- `serve_read`：从`ipc->read.req_fileid`的当前位置（可以通过`OpenFile->Fd *o_fd -> fd_offset`得到）读取至多`ipc->read.req_n` bytes。调用`openfile_lookup`与`file_read`即可
- `serve_write` & `devfile_write`：模仿`serve_read`实现即可。

6. Exercise 7

`sys_env_set_trapframe`：注意权限的设置

```
env->env_tf.tf_eflags |= FL_IF; // interrupts enabled
env->env_tf.tf_eflags &= ~FL_IOPL_MASK; // Clear IOPL
env->env_tf.tf_eflags |= FL_IOPL_0; // Set IOPL to 0
env->env_tf.tf_cs |= 3; // Code protection level 3
env->env_tf.tf_ss |= 3; // Stack protection level 3
```

`syscall()`中需要添加case以支持这一syscall的分配。

7. Exercise 8

- 修改`duppage`，增加一个权限判断即可`if (uvpt[pn] & PTE_SHARE)`
- `copy_shared_pages()`：类似`fork()`函数，遍历所有的页并判断相应权限位即可。
- 最初无法通过`Protection I/O space`的测试，反复排查后发现是由于我在lab4中challenge时自定义了在用户态对`General Protection Fault`的处理，导致不会输出`TRAP`，可以通过注释掉`kern/trap.c/trap_dispatch()`内的下述代码即可通过测试。

```
case T_GPFLT:
        cprintf("TRAP: General Protection Fault at 0x%08x\n", tf->tf_eip);
        gpflt_handler(tf);
        return;
```

8. Exercise 9

这一问很容易，在`kern/trap.c/trap_dispatch()`添加对于case即可。

9. Exercise 10

代码的实现是容易的，模仿重定向`>`的实现即可。

> ⑦ 卡了很久的bug?
>
> 在完成所有的Exercise后，发现无法正常通过shell的相关测试，经过排查之后发现是在`lib/pipe.c/_pipeisclosed()`中`n != nn`，即当前的环境被非预期的抢占。
> 知道目前并不真正明白具体触发的原因，下面是一些猜测:
>
> 1. 先前在代码有一些 debug 用的`cprintf`，也许因此触发了 I/O，导致当前环境被抢占。
> 2. 在上一个lab的challenge中，我实现了`sfork()`对`thisenv`进行了重写，可能导致了未知bug

在处理上述问题之后成功通过了测试。（感动🥹）

III. Result

成功通过了所有测试 150/150

```
make[1]: Leaving directory '/home/ubuntu/6.828/lab'
internal FS tests [fs/test.c]: OK (1.0s)
  fs i/o: OK
  check_bc: OK
  check_super: OK
  check_bitmap: OK
  alloc_block: OK
  file_open: OK
  file_get_block: OK
  file_flush/file_truncate/file rewrite: OK
testfile: OK (1.2s)
  serve_open/file_stat/file_close: OK
  file_read: OK
  file_write: OK
  file_read after file_write: OK
  open: OK
  large file: OK
spawn via spawnhello: OK (0.8s)
Protection I/O space: OK (1.0s)
PTE_SHARE [testpteshare]: OK (1.1s)
PTE_SHARE [testfdsharing]: OK (1.0s)
start the shell [icode]: Timeout! OK (30.6s)
testshell: OK (1.8s)
primespipe: OK (4.7s)
Score: 150/150
ubuntu:lab/ (lab5) $
```