_lab01

Lab01: UNIX Shell with History Feature

22281089 陈可致

关于linux的流程控制方法

- `fork()`
  - 复制当前进程来创建一个新进程作为子进程
  - 子进程返回0, 父进程返回pid
- `exec()`
  - 父进程调用 `fork()` 生成子进程之后, 子进程通过 `exec()` 替换原本的代码, 执行新的程序
- `wait()`
  - 使父进程等待子进程结束, 确保子进程正常执行完成

我对程序的设计

- 程序主体

```
char *args[MAX_LINE / 2 + 1];  // 存放token
static string MeIoN_Input;     // 输入
vector<string> history;        // 历史命令
int MYGO = 1;                  // 用于控制程序结束

while (MYGO == 1) {

    std::cout << "psh> ";      // 命令行输入提示
    flush();

    std::getline(std::cin, MeIoN_Input);

    // do something
}
```

- 判断输入合法情况

```
// too long ?
auto check_length = [&] (const string &input)->bool {
    if (input.length() > MAX_LINE) {
        std::cerr << "Input is too long\n";
        return false;
    }
    return true;
};
// empty
if (MeIoN_Input.empty()) {
```

```
        continue;
    }
    // too many tokens
    if (tokens.size() > MAX_LINE / 2) {
        std::cerr << "Too Many Tokens\n";
        flush();
        continue;
    }
```

- 对特殊命令的处理 (展示 历史检索)

```
if (MeIoN_Input == "history") {
    print_history();
    continue;
} else if (MeIoN_Input == "!!") {
    if (history.empty()) {
        std::cerr << "No Command\n";
        flush();
        continue;
    }
    MeIoN_Input = history.back();
} else if (MeIoN_Input[0] == '!' and MeIoN_Input.length() > 1) {
    const int id = std::stoi(MeIoN_Input.substr(1));
    MeIoN_Input = get_history(id);
}
```

- 对历史命令的展示

```
// 打印历史vector内最后10条记录
auto print_history = [&] ()->void {
    std::cout <<"Show History" << std::endl;
    for (int i = std::max(0, (int)history.size() - Print_Count); i < history.size(); ++i) {
        std::cout << i + 1 << ' ' << history[i] << std::endl;
    }
};
```

- 检索历史信息

```
// 直接利用下标检索
auto get_history = [&] (int id)->string {
    --id;
    if (id > -1 and id < history.size()) {
        return history[id];
    }
    std::cerr << "Get Command Fail\n";
    flush();
    return "";
};
```

- 将命令转化为token

```
// 利用istringstream和getline函数将命令分割为一系列token
auto get_token = [&] (const string &input)->vector<string> {
    static string token;
    std::istringstream _tmp(input);
    vector<string> tokens;

    while (std::getline(_tmp, token, ' ')) {
        tokens.emplace_back(token);
    }

    return tokens;
};
```

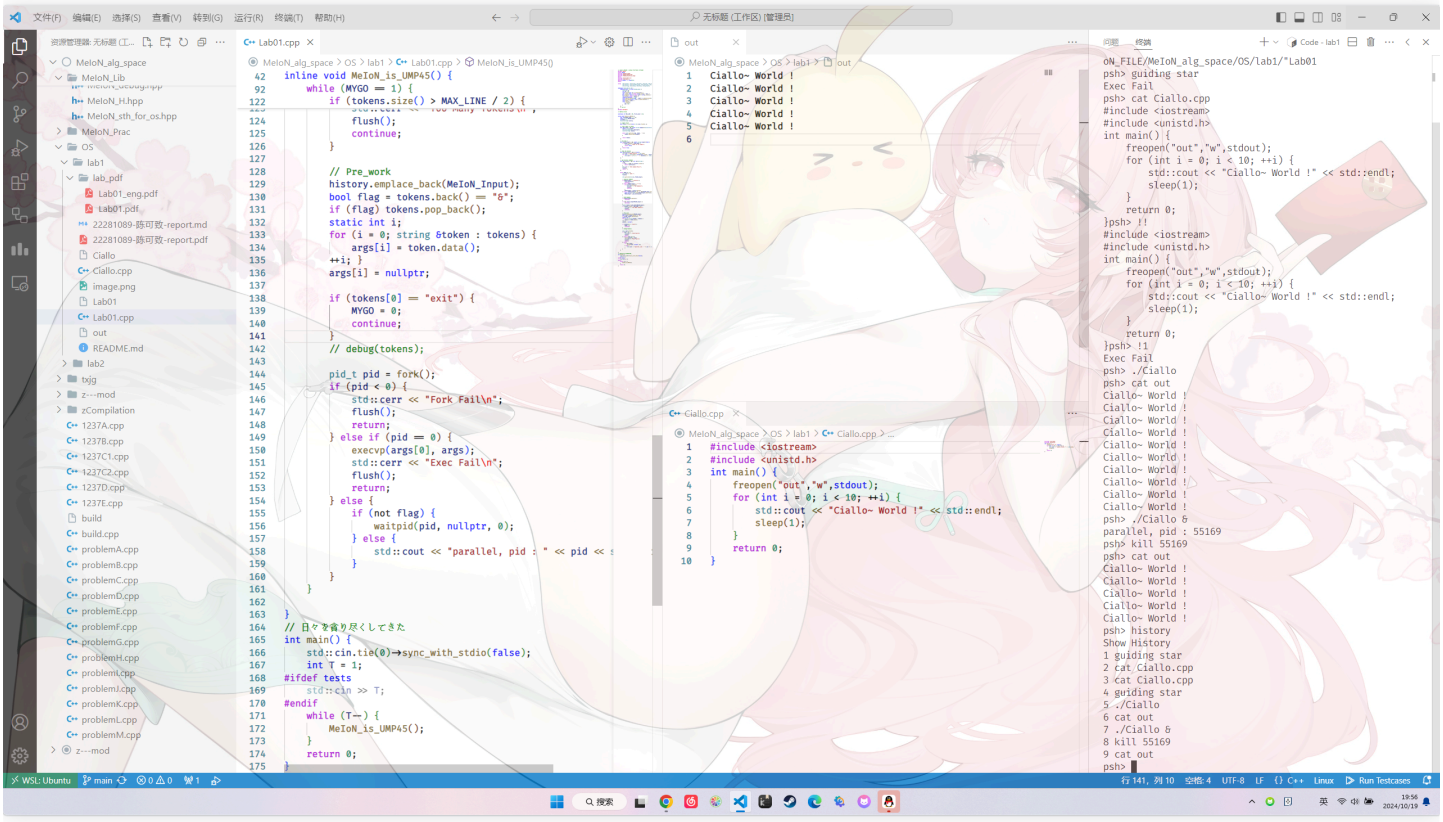
利用标记区分是否并行

```
bool flag = tokens.back() == "&";
if (flag) tokens.pop_back();
```

- 利用 fork() 和 exec() 进行子进程调用

```
pid_t pid = fork();
    if (pid < 0) {
        std::cerr << "Fork Fail\n";
        flush();
        return;
    } else if (pid == 0) {
        execvp(args[0], args);
        std::cerr << "Exec Fail\n";
        flush();
        return;
    } else {
        if (not flag) {
            waitpid(pid, nullptr, 0);
        } else {
            std::cout << "parallel, pid : " << pid << std::endl;
        }
    }
```

实验结果



各项功能正常运行

遇到的问题及我的解决方案

- 安装Linux子系统
  - 根据视频网站教程, 利用wsl安装了Ubuntu
- 配置Linux环境下的vscode开发环境
  - 根据网络资料在Linux上安装了g++, 并从头开始重新进行了vscode的配置, ~~更适合中国大学生体质的vscode~~
- 对进程调用以及并发运行不够了解

- 查阅了 `fork()` 以及 `exec()` 相关资料, 经过尝试实现了实验要求功能

参考资料

- 【Ubuntu + VS Code搭建简单开发环境】 https://www.bilibili.com/video/BV1HL411c7Wp
- https://www.cnblogs.com/love-jelly-pig/p/8471206.html

心得体会

有趣, 学到很多

标签: 作业