

2024年种子班招新日测题

前言



恭喜大家来到种子班日测考核!

日测的主要内容为，一天的时间内，使用C语言完成一个小项目并制作PPT，在日测结束后进行答辩~

项目——Dian Shell

- 在你的终端界面上，是不是有个小小的提示符，可以输入指令并执行？这个帮助你 and 系统进行交互，执行指令的小程序，就是Shell。
- 日测的项目是，使用C语言完成Dian Shell，一些可以在终端上运行的用户指令和程序，以及部分Shell的特性，如环境变量、重定向、管道命令等）。
- 参考的效果如下图：

```
ubuntu@jyl-desktop:~/seedclass_day_test/nniyilun_kotae$ ./bin/myshell
Dian Shell Login: root
Password:
Login successful.

  _ _ _ _ _
 | _ _ _ \ ( )
 | | | | | _ _ _ _ _
 | | | | | / _ ' _ \
 | | | | | ( _ | | |
 | _ _ _ / | | \ _ _ _
-- Welcome to Dian Shell --
[INFO] [Thu Jun  6 20:22:50 2024] (src/main.c:85) initialize shell and user login successfully
root@/home/ubuntu/seedclass_day_test/nniyilun_kotae>
```

具体时间安排

1. 日测时间：9:00-20:00共11个小时
 - a. 在规定时间内完成项目和答辩的PPT。
 - b. 请及时更新日测进度跟踪表 [📄 日测进度跟踪表](#)，在表内填写你完成各个任务的时间。
 - c. 日测结束前将项目代码打包，和PPT一起作为附件填写在进度跟踪表后。
2. 日测答辩：晚上20:30开始
 - a. 日测结束后，会发布答辩分组名单。

注意事项

1. 项目并非不会通过检查输入输出匹配判断正误，输出格式没有严格的限制，实现要求的功能即可。
2. 只能使用C语言进行代码的编写。
3. 我们提供了代码框架仅作为提示参考，你可以随意改动，或者自己从头开发也是可以的。
4. 我们会从各任务的完成情况、学习态度、思考理解等多方面对大家进行综合评价。
5. 不允许使用C语言标准库中的 `system()` 函数直接执行命令。

框架代码说明

本次测试中源代码全部位于 `src` 文件夹下，并使用GNU make进行编译。`src` 文件夹内目录如下所示：

```
1 src
2 |— main.c
3 |— sh
4 |   |— common.h
5 |   |— logger.c
6 |   └─ ...
7 └─ user
   |— ls.c
   └─ ...
```

- `src/sh` 目录下所有文件和 `src/main.c` 为 `Shell` 本身的代码。
- `src/user` 目录下每一个 `.c` 文件都是独立程序的代码，是在你的 `Shell` 中执行的程序。
- 在 `src` 同级目录下，执行以下编译指令：
 - **make shell**： `src/sh` 目录下的所有 `.c` 文件和 `src/main.c` 一起编译成 `shell` 程序。你可以在 `src/sh` 目录下添加自己需要的新文件一同加入编译。
 - **make user_program**：将 `src/user` 中的每一个 `.c` 文件分别编译成一个单独的可执行文件，该文件与对应的 `.c` 文件有着相同的名称，例如 `ls.c` 将会被编译成可执行文件 `ls`。
 - **make**：同时编译两个部分，等同于一起执行以上两条指令。
 - **make clean**：清除 `bin` 目录下的所有编译生成的可执行文件。
- `bin`：编译生成的可执行文件会在这个目录下

Dian Shell

Level 0 优雅地编写代码

1. **代码编写规范：**代码的可读性与规范性对于一个优秀的程序员而言相当重要，缩进、变量以及函数的命名规则是否优雅是判断一个优秀程序员的过程中不可忽视的因素。希望你能在今天的测试中注重代码规范性的问题。
2. **日志：**对于较复杂的项目来说，如果能输出程序运行中的相关状态信息，对于代码调试工作非常有帮助，这样的信息就叫做日志（log）。在这个阶段，你需要使用c语言实现一个日志模块。

具体要求

1. 每行日志包括日志等级（DEBUG, INFO, WARNING, ERROR, FATAL）、时间、文件名、行号
2. 日志信息支持格式化字符串输出，类似printf()函数一样可以传入可变参数。
3. 调用接口简单。

```
[test.c:4]-[2024-01-23 15:16:34]-[DEBUG] - Hello, world!  
[test.c:5]-[2024-01-23 15:16:34]-[INFO] - Hello, world!  
[test.c:6]-[2024-01-23 15:16:34]-[WARNING] - Hello, world!  
[test.c:7]-[2024-01-23 15:16:34]-[ERROR] - Hello, world!  
[test.c:8]-[2024-01-23 15:16:34]-[FATAL] - Hello, world!
```

Level 1-1 Shell login

传统的 `shell` 程序会记录目前正在使用它的用户，这是为了划分不同用户的操作权限。在这一步中，你需要修改源代码并设计一个进入 `shell` 时的登陆模块，用户需要输入账户名称和密码才能进入 `shell` 中。用户名称和密码你可以自己设定。

具体要求

1. 实现一个登录模块
2. 有一定的**提示性信息**
3. 输入密码时**不能显示出密码明文**
4. **支持多位不同的用户登陆（用户和密码之间一一对应）**
5. 用户和密码放在某个文本文件中即可

```
seedclass_entrance_exam2024 on main ?4 bin/shell  
Dian Shell Login: Dian  
Password:
```

示例输出（密码被隐藏）

Level 1-2 The first program running in Dian shell

当我们在命令行中输入一个程序的名称并按下回车时，`shell` 会为我们执行这个程序。被运行的程序将会作为 `shell` 的子程序。读到这里，你应该开始好奇，`shell` 是如何将读取到的程序作为其子程序运行的呢？请阅读以下内容。

- 1 **Fork and Exec**
- 2 New processes are created by the two related interfaces fork and exec.
- 3 **Fork**
- 4 When you come to metaphorical "fork in the road" you generally have two options to take, and your decision effects your future. Computer programs reach this fork in the road when they hit the `fork()` system call.
- 5 At this point, the operating system will create a new process that is exactly the same as the parent process. This means all the state that was talked about previously is copied, including open files, register state and all memory allocations, which includes the program code.
- 6 The return value from the system call is the only way the process can determine if it was the existing process or a new one. The return value to the parent process will be the Process ID (PID) of the child, whilst the child will get a return value of 0.
- 7 At this point, we say the process has forked and we have the parent-child relationship as described above.
- 8 **Exec**
- 9 Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.
- 10 This is where the `exec` system call comes into play. `exec` will *replace* the contents of the currently running process with the information from a program binary.
- 11 Thus the process the shell follows when launching a new program is to firstly fork, creating a new process, and then `exec` (i.e. load into memory and execute) the program binary it is supposed to run.

在这一步中，你需要实现在 `shell` 程序中运行子程序的任务。可能需要用到以下函数：

- `fork()`：创建子进程。
- `exec`：Linux操作系统中提供了一系列`exec`开头的函数，用于在进程中启动另一个程序。

在你编写的 `shell` 程序中尝试运行我们为你准备的 `helloworld` 程序，`helloworld` 程序对应的源代码位于 `src/user/helloworld.c`。`helloworld` 程序接收一个整形命令行参数作为参数，并将其作为进程的退出值，运行方式为

```
1 0-> ./helloworld
2 [ERROR] expected only one parameter, but got 0
3 exit with 255 0-> ./helloworld 123
```

```
4 hello world!
5 exit with 123 ○->
```

要求：

1. 完成读入和解析命令的功能（注意命令参数的识别哦）
2. 用 `fork` 和 `exec` 执行命令
3. 在 `shell` 程序中成功执行 `helloworld` 程序并在 `shell` 中获取进程的退出值，打印进程的退出值（不能直接在 `helloworld.c` 打印进程的退出值）。
4. 执行完成后，你的shell不应该一同退出。

- 示例：

```
1 ○-> ./helloworld
2 [ERROR] expected only one parameter, but got 0
3 exit with 255 ○-> ./helloworld 123
4 hello world!
5 exit with 123 ○->
```

Level 1-3 cd, pwd & ls

在这个部分，你需要实现 `shell` 中三个常用的指令：`cd`、`pwd` 和 `ls`。

- `cd`： `change directory` 的缩写，用于切换当前的工作目录。
- `pwd`： `present working directory` 的缩写，用于显示 `shell` 当前的工作目录。
- `ls`： `list` 的缩写，用于某个目录下的所有文件。
 - 当 `ls` 后无任何参数时，`ls` 会默认显示当前工作目录下的所有文件。
 - 当 `ls` 后面接有命令行参数时，`ls` 会将每一个命令行参数看作是一个文件目录名称，并分别显示其目录下对应的文件。例如当前目录下有 `LSTM` 和 `dotfiles` 两个文件夹，则 `ls LSTM dotfiles` 的效果如下：

```
1 ○-> ./ls LSTM dotfiles
2 LSTM:
3 model.py train.py
4
5 dotfiles:
6 config icons README.md theme wallpapers
```

要求

1. 在命令提示符前输出当前文件夹的名称（仅输出名称，不输出全部的路径）
2. `cd` 命令，使用该命令能够切换 `shell` 程序的工作目录。
3. `pwd` 命令，显示 `shell` 当前的工作目录。
4. `ls` 命令，支持0个或多个命令行参数，每一个参数分别为一个文件夹名称。
5. （选做）如果 `ls` 遇到软链接文件，请显示出其对应的原文件。
6. 思考一下 `cd` 指令和上面的 `helloworld` 有什么区别。

Level 1-4 jump

管道通信之linux进程间通信常见的方式之一。请你查阅相关管道通信相关的知识，设计一个Linux程序来解决经典的跳楼梯问题，要求使用多个进程和管道来进行计算。

经典跳台阶问题如下：

- 一个人站在楼梯的底部，他可以一次跳一步或两步。求他跳到第 n 级台阶有多少种不同的方法。
- **参照如下状态方程解决问题：** $f(i) = f(i-1) + f(i-2)$ 。这个方程代表着，跳到第 i 个台阶的方法数，是跳到这个台阶的前一个和前两个台阶的方法数之和。易知只有一个台阶和两个台阶的方法数，可以通过不断迭代计算任意台阶的方法数。

这道题要求使用多进程和管道的方式解决这个问题：

- 同时创建三个子进程，前两个进程把数据传递给第三个进程，第三个进程将两个数字相加后，将其传递给后续进程....
- 需要注意的是，每个进程执行时都需要进行一次打印。具体格式如下要求所示。

要求

1. 使用多进程完成跳台阶程序 `jump`
2. `jump` 程序接收一个非负整数 n 作为输入，代表总共有 n 级台阶。

输出格式如下图所示：

```
1 process:[proc_id], layer:[layer_num], method:[method_num]
```

说明：

- `proc_id`：当前进程的进程号
- `layer_num`：当前台阶的层数

- `method_num`: 跳到当前台阶的方法数

示例:

```
1 process:1001, layer:1, method:1
2 process:1002, layer:2, method:2
3 process:1003, layer:3, method:3
4 process:1004, layer:4, method:5
5 process:1005, layer:5, method:8
6 ...
```

提示

- 本小节与前面涉及的所有代码是独立的，你可以在linux系统自带的 `shell` 中运行 `jump` 程序，也可以在1-2实现的基础上在你所编写的 `shell` 中运行 `jump`。

Level 1-5 SIGINT, SIGTSTP与SIGCONT信号传递

你可能已经发现，如果你编写的 `shell` 中使用 `ctrl-c` 尝试终止一个正在运行的程序，你会连同整个 `shell` 一起退出，但这并不符合linux原生shell的规范。那么如何让 `ctrl-c` 只对当前正在运行着的程序起作用呢？为了搞清楚这个问题，我们首先需要了解当我们按下 `ctrl-c` 时发生了什么。

当按下 `ctrl-c` 时，系统会向当前终端中运行着的进程发送一个 `SIGINT` (signal interrupt) 信号，表示该进程需要终止。同理，当按下 `ctrl-z` 时，发送的信号变为 `SIGTSTP` (signal terminal stop)，表示该进程需要暂停，当我们发送 `SIGCONT` (signal continue) 信号时，该暂停着的程序收到了可以继续运行的信号而又重新恢复运行。

在本小节中，你需要实现在 `shell` 中接收 `SIGINT` (`ctrl-c`) 以及 `SIGTSTP` (`ctrl-z`) 信号并将其传递给正在运行着的子程序，同时你还需要实现一个 `fg` (foreground) 命令对已被 `ctrl-z` 暂停的进程发送 `SIGCONT` 信号调度到前台来运行。

我们在 `src/user/signaltest.c` 中为你准备了测试程序。

当你实现了将终止信号传递给运行着的进程时，你可能发现没有办法退出 `shell` 了（除非你新开一个终端并向 `shell` 程序发送 `SIGINT`）。因此在本小节中你还需要实现一个 `exit` 命令用于退出 `shell` 程序

要求

1. 实现在 `shell` 中接收上述三个信号（`SIGINT`、`SIGTSTP`、`SIGCONT`）并将其传递给正在运行着的子程序。
2. 在 `shell` 中运行 `bin/signaltest` 并分别尝试传递上述三个信号。
3. 实现一个 `exit` 命令用于退出 `shell`

4. 实现 `fg` 命令将暂停的进程调度到前台来运行

- 示例：

```
1  o-> ./signaltest
2  ^Cprocess    12926 get sigint  (按下了ctrl-c)
3  exit with -1 o-> ./signaltest
4  ^Zprocess    12942 get sigtstp (按下了ctrl-z)
5  o-> fg
6  process      12942 get sigcont
7  ^Cprocess    12942 get sigint  (按下了ctrl-c)
8
9  exit with -1 o->
```

Level 1-6 Environment variable

如果你不知道环境变量是什么，请先自行查阅shell环境变量的相关知识。

本关卡要求实现shell全局环境变量，并可以添加，删除，和展示shell变量。


要求：

1. 实现 `set` 指令，用 `set NAME=VAR` 的命令形式设置环境变量。
2. 实现 `unset` 指令，用 `unset NAME` 的命令形式删除环境变量。
3. 实现 `env` 指令，用 `env` 的命令形式打印当前所有的环境变量，用 `env NAME` 的形式打印变量名为 `NAME` 的环境变量。
 - 每一行输出格式： `NAME=VAR`
 - 如果名为 `NAME` 的变量不存在，则输出： `NAME=`
 - 实现环境变量解析的功能，即，对于带有 `$NAME` 的命令在执行之前应该先将 `$NAME` 换成相应环境变量值。

- 示例：

```
1  o-> set Dian="Inovative Team"
2  o-> echo $Dian # 执行系统自带的echo 命令
3  "Inovative Team"
```


Level 2

 从现在开始，需要特别注意代码的可扩展性。Level2要求实现的内容并不是割裂开来的，而应该最终实现一些组合命令。可以在原系统的命令行中进行测试

这里给出一个测试命令供参考：在bin目录下

```
echo ../ > tmp.txt ; cat tmp.txt | xargs ls | grep a > out.txt  
; cat out.txt
```

Level 2-1 List commands

实现shell命令中的 `;`，即依次运行被 `;` 分隔的命令。注意，只有在前一个命令退出时才会启动后一个命令。例如：

```
1  o-> echo hello > tmp.txt ; cat tmp.txt  
2  // 将会依次运行echo hello > tmp.txt 与cat tmp.txt两个命令
```

要求

1. 不能直接调用system接口
2. 支持分隔两个以上的命令

Level 2-2 Background commands

实现shell命令中的后台运行命令，即以 `&` 结尾的命令。后台命令表示shell将其放到后台运行，并能够在上一个后台命令退出前运行其他的命令。你可以将 `signaltest` 放到后台运行并尝试运行其他命令。

```
1  o-> ./signaltest &  
2  o-> ls  
3  ...(ls 的输出)
```

要求

1. 为1-5中的 `fg` 接口添加功能：`fg` 可以将后台命令调度到前台来运行，即：

```
1  o-> ./signaltest &
```

- 2 o-> fg
- 3 (卡在./signaltest的死循环中，需要用ctrl-c来结束进程)

2. 支持多个命令同时在后台运行

- a. 每次使用fg时，将最早 **被放在后台运行或者ctrl-z暂停的命令** 调度到前台来，即一次只调度一个后台或暂停中的命令。

Level 2-3 Re-directory commands

在shell中，指令可以添加重定向符号 `>`，`<` 和 `>>`，改变执行命令的默认输入输出方式。

要求

1. 实现shell命令中的输入输出重定向: `>`，`<` 和 `>>`。
2. 当有多个输入或输出时，报错。

Level 2-4 Pipe commands

实现shell命令中的管道命令符号 `|`。管道命令符号表示将前一个命令的标准输出作为后一个命令的标准输入。管道命令的前后两个程序是**同时运行**的，不是一个运行完成后下一个接着运行。

要求

1. 支持两个以上的命令用管道连接

Level 2-5 命令组合

实现可以解析上述四个特性指令的组合形式。例如实现以下命令的解析：

```
1 echo ../ > tmp.txt ; cat tmp.txt | xargs ls | grep a > out.txt ; cat out.txt
```

要求

1. 如果指令同时有 `;` 和 `|`，应该如何运行呢？请参考原系统shell的运行逻辑设计实现。

Level 3-1 shell script

实现在shell中运行脚本，脚本的每一行为Level0-0到Level2-4中可能出现的命令。要求实现的运行脚本的命令为 `dian <脚本名称>`

要求

1. 测试用例

```
1 在script.dian中：
2 echo ../ > tmp.txt
3 cat tmp.txt | xargs ls | grep a > out.txt
4 cat out.txt
5 rm out.txt; rm tmp.txt
6
7 在命令行：
8  o-> dian script.dian
9  (运行结果)
```