ICS II Shell Lab 报告

中国人民大学 sheriyuo

摘 要

RUC 2023-2024 计算机系统基础 II Shell Lab 的思路与实现。

1 基本函数

1.1 错误处理函数

对所有用到的函数进行了错误处理封装，通过 `unix_error` 来输出错误信息。

```
/* Error detecting function */
pid_t Fork(void);
void Sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
void Sigemptyset(sigset_t *set);
void Sigfillset(sigset_t *set);
void Sigaddset(sigset_t *set, int signum);
void Setpgid(pid_t pid, pid_t pgid);
void Kill(pid_t pid, int sig);
void Execvpe(const char *filename, char *const argv[], char *const envp[]);
void Pipe(int fds[2]);
```

1.2 eval

`eval` 函数通过提供的 `parseline` 来获取 `bg` 状态与参数列表，并使用 `builtin_cmd` 来判断是否立即执行。如果是内置命令，则直接在 `builtin_cmd` 中调用 `do_bgfg` 来执行，否则回到 `eval` 函数。

在 `Fork` 前先阻断 `SIGCHLD`，子进程中结束阻断，`Setpgid(0,0)`，然后执行命令并退出；父进程中，先阻断所有信号并 `addjob`，随后恢复阻断 `SIGCHLD`，若状态为 `BG` 则输出信息，否则 `waitfg(pid)`。

更多的补充见附加分部分。

1.3 builtin_cmd

对 argv[0] 进行判断，如果为内置指令则执行相关函数（do_bgfg 或 listjobs），并设返回值为 1 以跳过 eval；否则返回 0。

1.4 do_bgfg

判断 argv[0] 来获取状态，然后分类讨论 argv[1]（NULL、数字或 %），以此区分 pid 与 %jobid。获取到 job 后，对其 state 进行修改，对其进程组（-(job->pid)）发送 SIGCONT 信号。

错误提示信息依照 tshref 实现，trace14.txt 的运行结果如下：

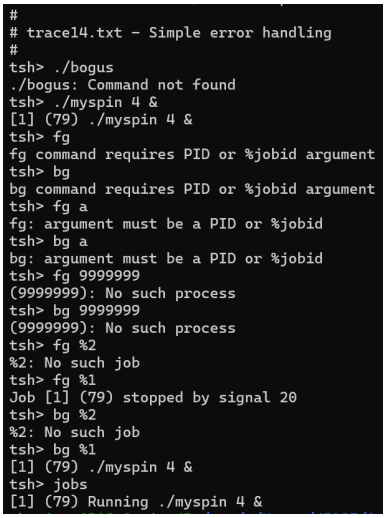

图 1.4.1: trace14.txt

1.5 waitfg

由于 sigsuspend 的原子性，以此来实现等同 sleep 的阻断功能。

```
void waitfg(pid_t pid) {
    sigset_t mask;
    Sigemptyset(&mask);
    while (fgpid(jobs) > 0) {
        sigsuspend(&mask);
    }
    return;
}
```

1.6 sigchld_handler

采用 `pid = waitpid(-1, &status, WNOHANG | WUNTRACED)` 来获取 pid，先阻断所有信号，然后对获取的 status 进行判断。

`WIFEXITED(&status)` 或 `WIFSIGNALED(&status)` 为真的情况删除该 pid 对应的 job，而 `WIFSTOPPED(&status)` 为真的情况更新其状态为 ST，然后结束阻断，最后将 errno 改为初始的记录值。

1.7 sigint_handler 与 sigtstp_handler

两者实现几乎完全一致，不同之处仅在于 kill 传输的信号。

阻断部分同 sigchld_handler，对 `fgpid(jobs)` 所在的进程组传输相应的信号即可。

2 附加分

2.1 支持 PATH 环境变量

由于 execve 本身并不支持对 envp 进行搜索，于是将所有的 execve 替换为集成了该功能的 execvpe。

在 main 函数开头进行环境变量的添加 `setenv("PATH", "/bin:/usr/bin", 0)`，传参使用 environ 即可。

2.2 重定向 >

遍历 argv 参数列表，若匹配到 ">"，取其下一个参数作为输出文件 output，采用 `open(output, O_CREAT | O_WRONLY | O_TRUNC, 0644)` 打开文件输出流，并用 dup2 函数将其重定向至子进程标准输出流 STDOUT_FILENO。随后正常执行子进程，以 `exit(0)` 结束。

笔者仿照 bash 的错误信息，对可能出现的错误输入进行了处理：

```
tsh> echo hello! > 1.txt
tsh> cat 1.txt
hello!
tsh> echo hello! >
-tsh: syntax error near unexpected token `newline'
tsh>
```

图 2.2.1: 错误输入处理

2.3 管道 |

由于笔者不想对 parseline 的解析进行优化，于是对 eval 原有的框架进行了更改，仅注释掉了 parseline 最后判断 "&" 并 argc-- 的部分，将其放在了 eval 中。

由于无法直接以 | 为分隔符进行解析，笔者采用了 goto loop 的结构，读取到 | 就停止当前 argv 的解析，并将其赋值为 _argv 来进行原先的 eval 过程，如果此后下一个 argv 非空，则用 goto 来继续解析过程。

在解析前，用 calloc 创建进程列表，如果本次 Fork 含有管道，采用 pipe(fds) 创建管道，只需要将进程的输出绑定到写端 fds[0]，将下一个进程的读入绑定到读端 fds[1]，就实现了一次进程间的管道通信。采用 prev_fd 在父进程中记录前一个管道的读端，以此来传递给下一个子进程。

同时，第一个进程的读入绑定到标准输入流 STDIN_FILENO，最后一个进程的输出绑定到标准输出流 STDOUT_FILENO。

最后，对 zombie 进程进行判断处理。

```
tsh> | |
-tsh: syntax error near unexpected token `|'
tsh> echo hello | echo \|
\|
tsh>
```

图 2.3.1: 管道

值得一提的是，这里的 echo 无法实现转义字符（同 tshref），而 echo | 会被识别为错误输入。

2.4 混合支持

混合支持的实现重点在于同时适配管道与重定向，代码中以 is_pipe 和 is_wr 分别表示有管道和重定向的情况，对两种情况的重定向分别作了不同的处理，并对最后一个管道的重定向输出采用覆盖 STDOUT_FILENO 的方法进行实现。

```
tsh> cat tsh.c | grep "dup2" | sort | uniq > 2.txt
tsh> cat 2.txt
                        dup2(fds[1], STDOUT_FILENO);
                    dup2(wr_fd, STDOUT_FILENO);
                dup2(prev_fd, STDIN_FILENO);
                dup2(wr_fd, STDOUT_FILENO);
    dup2(1, 2);
tsh>
```

图 2.4.1: 混合支持

3 附录

3.1 对比测试结果

以 `trace15.txt` 为测试用例：

```
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (131) terminated by signal 2
tsh> ./myspin 3 &
[1] (133) ./myspin 3 &
tsh> ./myspin 4 &
[2] (135) ./myspin 4 &
tsh> jobs
[1] (133) Running ./myspin 3 &
[2] (135) Running ./myspin 4 &
tsh> fg %1
Job [1] (133) stopped by signal 20
tsh> jobs
[1] (133) Stopped ./myspin 3 &
[2] (135) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (133) ./myspin 3 &
tsh> jobs
[1] (133) Running ./myspin 3 &
[2] (135) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
./sdriver.pl -t trace15.txt -s ./tshref -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (149) terminated by signal 2
tsh> ./myspin 3 &
[1] (151) ./myspin 3 &
tsh> ./myspin 4 &
[2] (153) ./myspin 4 &
tsh> jobs
[1] (151) Running ./myspin 3 &
[2] (153) Running ./myspin 4 &
tsh> fg %1
Job [1] (151) stopped by signal 20
tsh> jobs
[1] (151) Stopped ./myspin 3 &
[2] (153) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (151) ./myspin 3 &
tsh> jobs
[1] (151) Running ./myspin 3 &
[2] (153) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

图 3.1.1: 对比测试结果