shlab实验报告

完成情况

- 代码针对16个测试样例的输出结果与测试程序相同。
- 代码实现了简单的错误输入检查
- 代码支持PATH环境变量
- 代码重定向了`>`

代码详解

parseline函数

```
int parseline(const char *cmdline, char **argv,char **redirect_filename)
{
    ......

    char *redirect_ptr;

    *redirect_filename = NULL;

    ......

    if (*buf == '>' || *buf == '&') { /* Error: Command starts with '>' or '&'
*/
        fprintf(stderr, "Error: Command starts with an invalid character\n");
        return -1; /* Indicate error */
    }

    ......

    redirect_ptr = strchr(buf, '>');
    if(redirect_ptr && *(redirect_ptr-1) == 'h'){/* 避免tsh>  */
        redirect_ptr++;
        redirect_ptr = strchr(redirect_ptr, '>');
    }

    if (redirect_ptr) {
        *redirect_ptr = '\0';
        redirect_ptr++;
        while (*redirect_ptr && (*redirect_ptr == ' '))
            redirect_ptr++;
        *redirect_filename = redirect_ptr;
    }
    ......
}
```

- 检测了输入，当命令以`>`或`&`开始时视为非法输入，如果重定向没有地址也会提醒（在eval中）
- 添加了一个参数redirect_filename，当检测到重定向时把地址存入参数，若无地址则为NULL。

eval函数

```
void eval(char *cmdline)
{
    char *argv[MAXARGS]; /* argv for execve() */
    char buf[MAXLINE]; /* holds modified cmd line */
    int bg; /* should the job run in bg or fg? */
    pid_t pid; /* process id */
    char *redirect_filename = NULL;

    sigset_t mask_child,prev_mask;
    strcpy(buf, cmdline);
    sigemptyset(&mask_child);
    sigaddset(&mask_child,SIGCHLD);

    bg = parseline(buf, argv, &redirect_filename);
    if (argv[0] == NULL)
        return; /* ignore empty lines */

    if(builtin_cmd(argv))
        return;

    sigprocmask(SIG_BLOCK,&mask_child,&prev_mask);
    if(!(pid = Fork())){
        sigprocmask(SIG_UNBLOCK,&mask_child,NULL);
        setpgid(0,0);

        if (redirect_filename != NULL) {
            int fd = open(redirect_filename, O_CREAT | O_WRONLY | O_TRUNC,
S_IRUSR | S_IWUSR);
            if (fd == -1) {
                printf("Unable to open %s for redirection\n",
redirect_filename);
                exit(1);
            }
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }

        if(execvp(argv[0],argv) < 0)
            printf("%s: Command not found\n",argv[0]);
        exit(0);
    }

    addjob(jobs,pid,bg?BG:FG,cmdline);

    if(bg)
        printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
    else
        waitfg(pid);
    sigprocmask(SIG_UNBLOCK,&mask_child,NULL);

    return;
}
```

- 大体上是基于老师PPT上的函数完成的。
- 在子进程处理时屏蔽了SIGCHLD信号以防意外的情况
- 若redirect_filename != NULL，则说明重定向。利用 dup2(fd, STDOUT_FILENO);重定向
- 使用execvp函数执行子程序，使之支持PATH环境变量

builtin_cmd函数

```
int builtin_cmd(char **argv)
{
    int jdg = 1;
    if(!strcmp("quit",argv[0]))
        exit(0);
    else if(!strcmp("jobs",argv[0]))
        listjobs(jobs);
    else if(!strcmp("fg",argv[0]) || !strcmp("bg",argv[0]))
        do_bgfg(argv);
    else
        jdg = 0;

    return jdg;
}
```

- 很简单的函数，对要求的四个内建命令一一处理即可

do_bgfg函数

```
void do_bgfg(char **argv)
{
    char* command = argv[0];
    char* id = argv[1];
    struct job_t* jb;

    if(id == NULL){
        printf("%s command requires PID or %%jobid argument\n",command);
        return;
    }
    else if(id[0] == '%'){
        jb = getjobjid(jobs,atoi(++id));
        if(jb == NULL){
            printf("%s: No such job\n",id);
            return;
        }
    }
    else if(id[0]<'0' || id[0]>'9'){
        printf("%s: argument must be a PID or %%jobid\n",command);
        return;
    }else if((jb = getjobpid(jobs,atoi(id))) == NULL){
        printf("%s: No such process\n",id);
        return;
    }

    if(!strcmp(command,"bg")){
        jb->state = BG;
        kill(-(jb->pid),SIGCONT);
```

```
        printf("[%d] (%d) %s", jb->jid, jb->pid, jb->cmdline);
    }
    else if(!strcmp(command,"fg")){
        jb->state = FG;
        kill(-(jb->pid),SIGCONT);
        waitfg(jb->pid);
    }
    return;
}
```

- 有很多代码是为了处理非法输入，依照测试样例一一写好即可
- 切换状态时只需要把job的state切换一下，然后给进程发信号使其running。最后，如果bg则输出运行信息，如果fg则阻塞主进程

waitfg函数

```
void waitfg(pid_t pid)
{
    sigset_t mask;
    sigemptyset(&mask);
    while (fgpid(jobs))
        sigsuspend(&mask);

    return;
}
```

- 老师的PPT上都有现成代码，拿来改改就行。这里由于waitfg函数仍在eval函数屏蔽SIGCHLD的区域中，所以mask要设置为空以接收SIGCHLD。

sigchld_handler函数

```
void sigchld_handler(int sig)
{
    int old_errno = errno;
    int state;
    pid_t pid;
    sigset_t prev_mask,full_mask;
    sigfillset(&full_mask);
    while((pid = waitpid(-1, &state, WNOHANG | WUNTRACED)) > 0){
        sigprocmask(SIG_BLOCK, &full_mask, &prev_mask);
        if (WIFEXITED(state)) {
            deletejob(jobs, pid);
        }else if(WIFSIGNALED(state)){
            printf("Job [%d] (%d) terminated by signal 2\n", pid2jid(pid), pid);
            deletejob(jobs, pid);
        }
        else if(WIFSTOPPED(state)){
            struct job_t* jb = getjobpid(jobs, pid);
            printf("Job [%d] (%d) stopped by signal 20\n", jb->jid, jb->pid);
            jb->state = ST;
        }
        sigprocmask(SIG_SETMASK, &prev_mask, NULL);
    }
    errno = old_errno;
```

```
    return;
}
```

- （这里的处理PPT上也有）还是利用while（waitpid>0）的方式回收，回收时要屏蔽信号以防意外操作。
- 对回收的子程序要判断其是主动死掉，收到信号死掉还是收到信号停止。后两者要打印信息。

sigint_handler和sigtstp_handler函数

```
void sigint_handler(int sig)
{
    int old_errno = errno;
    pid_t pid;
    if((pid = fgpid(jobs)))
        kill(-pid,sig);
    errno = old_errno;
    return;
}
```

- 这两个函数代码相同，也很简单。就是给指定进程发信号。