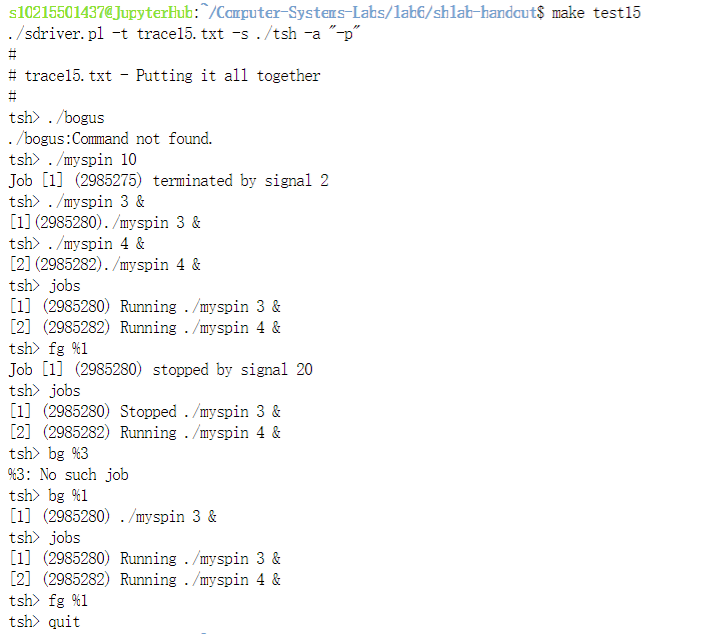
Trace15.txt的运行结果



Eval函数：

首先，根据书上给的内容建立框架：

void eval(char \*cmdline)

{

char \*argv[MAXARGS];/\*Argument list execve() \*/

char buf[MAXLINE];/\*Holds modified command line \*/

int bg;/\*Should the job run in bg or fg? \*/

pid\_t pid;/\*Process id \*/

strcpy(buf, cmdline);

bg = parseline(buf, argv);

if (argv[0] == NULL)

return;/\* Ignore empty lines \*/

if (!builtin\_cmd(argv)) {

if ((pid = Fork()) == 0) {/\* Child runs user job \*/

Execve(argv[0], argv, environ);

}/\* Parent waits for foreground job to terminate \*/

if (!bg) {

int status;

if (waitpid(pid, &status,0) < 0)

unix\_error("waitfg: waitpid error");

}

else

printf("%d %s", pid, cmdline);

}

return;

}

以上均为书本中的原始代码。在shelllab的指导pdf中介绍到，eval中进程在fork之前用sigprocmask阻塞SIGCHLD信号，之后在解除信号阻塞，之后在调用addjob添加孩子到作业列表用sigprocmask阻塞信号，因为子继承继承父进程的阻塞集合，所以子程序必须确保在执行新进程前解除阻塞SIGCHLD信号。父进程需以这种方式阻塞SIGCHLD信号，避免在父进程调用addjob之前，SIGCHLD处理器获取子进程(从而从任务列表中删除)的竞争状态。但是这个代码中存在bug，在子进程结束的时候应该delete job，避免父子竞争。经过改善后代码如下：

void eval(char \*cmdline)

{

char \*argv[MAXARGS];/\*Argument list execve() \*/

char buf[MAXLINE];/\*Holds modified command line \*/

int bg;/\*Should the job run in bg or fg? \*/

pid\_t pid;/\*Process id \*/

sigset\_t mask\_all,mask\_one,prev\_one;

strcpy(buf, cmdline);

bg = parseline(buf, argv);

if (argv[0] == NULL)

return;/\* Ignore empty lines \*/

if (!builtin\_cmd(argv)) {

Sigfillset(&mask\_all);/\* add every signal number to set \*/

Sigemptyset(&mask\_one);/\* create empty set \*/

Sigaddset(&mask\_one, SIGCHLD);/\* add signal number to set \*/

/\* block SIGINT and save previous blocked set \*/

Sigprocmask(SIG\_BLOCK, &mask\_one, &prev\_one); /\* Block SIGCHLD \*/

if ((pid = Fork()) == 0) {/\* Child runs user job \*/

/\* restore previous blocked set,unblocking SIGINT \*/

Sigprocmask(SIG\_SETMASK, &prev\_one, NULL); /\* Unblock SIGCHLD \*/

//Setpgid(0,0);

Execve(argv[0], argv, environ);

}/\* Parent waits for foreground job to terminate \*/

Sigprocmask(SIG\_BLOCK, &mask\_all, NULL); /\* Block SIGCHLD \*/

int st = (bg==0) ? FG : BG;

addjob(jobs,pid,st,cmdline);

Sigprocmask(SIG\_SETMASK, &prev\_one, NULL); /\* Unblock SIGCHLD \*/

if (!bg) {

//由于sigchld\_handler上面被调用，而上面回调用waitpid，因此这里不用调用只需循环等待即可

waitfg(pid);

}

else

printf("[%d] (%d) %s", pid2jid(pid),pid, cmdline);

}

return;

}

函数的思路：

1.初始化各变量以及设置信号阻塞合集

2 解析命令行，得到是否是后台命令，置位state（前后台标志）

3. 判断是否时内置命令，如果是立即执行。否则fork子进程。子进程调用执行具体的命令文件。然后exit父进程执行下一步.

4. 将子进程job加入到job list中。

5. 判断是否是bg,fg调用waifg函数等待前台运行完成，bg打印消息即可

需要注意，在test4的时候需要更改输出的格式。

Do\_bgfg函数

函数思路：

1.判断argv[]是否带%，若为整数则传入pid，若带%则传入jid。接着调用getjobjid函数来获得对应的job结构体，如果返回为空，说明列表中并不存在jid的job，要输出提示。

2.使用strcmp函数判断是bg命令还是fg命令

3.若是bg，使目标进程重新开始工作，设置状态为BG，打印进程信息

4.若是fg，使目标进程重新开始工作，设置状态为FG，等待进程结束

void do\_bgfg(char \*\*argv)

{

int jid;

struct job\_t \*job;

if(argv[1]==NULL){

printf("%s command requires PID or %%jobid argument\n",argv[0]);

//不合规范的输入指令，缺少进程ID或者任务ID

return ;

}

if(argv[1][0]=='%'){//如果是任务id

jid=atoi(&argv[1][1]);

//这里利用了系统atoi函数，传入参数是字符串首地址，返回该字符串所代表的数字

//也可以写成jid=argv[1][1]-'0';

job=getjobjid(jobs,jid);

if(job==NULL){//如果指令为空

printf("%%%d:No such job\n",jid);

return ;

}

}

else if(isdigit(argv[1][0])){//如果是进程id

//利用了系统isdigit函数，用于判断是否为十进制阿拉伯数字0~9，若是则返回1

jid = atoi(argv[1]);

job=getjobjid(jobs,jid);

if(job==NULL){//无法对应找到相应process，如bg 9999

printf("(%d): No such process\n",jid);

return ;

}

}

else {//如果无法满足以上的判断就说明当前terminal中输入的指令既不是进程id也不是任务id

printf("%s: argument must be a PID or %%jobid\n",argv[0]);

return ;

}

if(!strcmp(argv[0],"bg")){//如果是使用bg,作用是将当前任务状态变为后台任务

job->state=BG;

kill(-(job->pid),SIGCONT);//传递一个信号使任务重启然后在后台运行

printf("[%d] (%d) %s", jid, job->pid, job->cmdline);

}

else {//如果不满足以上条件判断则说明是使用fg，将当前任务状态变为前台任务

if(job->state==BG){

//如果是后台程序就转到前台并等待结束

job->state=FG;

waitfg(job->pid);

}

else if(job->state==ST){

//如果是挂起程序就重启并且转到前台，等待结束

job->state=FG;

kill(-(job->pid),SIGCONT);

waitfg(job->pid);

}

/\*前台程序和后台程序的差别在于shell是否等待程序结束，前台程序shell会主动等待然后回收程序，后台程序什么时候返回信号SIGCHLD然后回收掉。\*/

}

return;

}

sigint\_handler函数

函数思路：

1.调用函数fgpid返回前台进程pid

2.如果当前进程pid不为0，那么调用kill函数发送SIGINT信号给前台进程组

void sigint\_handler(int sig)

{

pid\_t pid=fgpid(jobs);

if(pid!=0)

{

kill(-pid,sig);

}

return;

}