实验日志 2.9

1.编程实现 eval()的后台作业管理功能并使用 trace04 验证

为了区别前台作业和后台作业，前台运行要等待子进程的终止而阻塞父进程操作，而在后台进程运行时，可以在父进程中输入命令执行其他操作。代码如下：

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    char buf[MAXLINE];
    int bg;
    sigset_t mask;
    pid_t pid;
    strcpy(buf, cmdline);
    bg = parseline(buf, argv); //读取命令
    if (argv[0] == NULL)
        return;
    if (!builtin_cmd(argv)) //如果是内置命令那么结束，否则创建新的子进程
    {
        sigemptyset(&mask);
        sigaddset(&mask, SIGCHLD); //在派生子进程之前阻塞SIGCHLD信号
        sigprocmask(SIG_BLOCK, &mask, NULL);
        // if ((pid = fork()) == 0) //子进程
        {
            sigprocmask(SIG_UNBLOCK, &mask, NULL); //解除阻塞
            setpgid(0, 0);
            //在子进程中通过execve寻找判断是否找到可执行文件
            if (execve(argv[0], argv, environ) < 0)
            {
                printf("%s: Command not found.\n", argv[0]); //找不到
                exit(0);
            }
        }
        if (!bg) //前台作业
        {
            if (addjob(jobs, pid, FG, cmdline) == 0)
                app_error("create job failed!");
            sigprocmask(SIG_UNBLOCK, &mask, NULL); //解除阻塞
            waitfg(pig);                           //等待该进程结束
        }
        else //后台作业
        {
            if (addjob(jobs, pid, BG, cmdline) == 0)
                add_error("create job failed!");
            sigprocmask(SIG_UNBLOCK, &mask, NULL);
            printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
        }
    }
    return;
}
```

测试结果：首先打印三行字符串，然后在后台运行myspin函数，让系统休眠一秒，以&符号终止代表在后台运行，验证的结果如下：

```
makelh@lh-VirtualBox:~/shlab-handout$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
[2] (2874) ./myspin 1 &
tsh> ./myspin 1 &
lh@lh-VirtualBox:~/shlab-handout$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (2880) ./myspin 1 &
```

测试结果和标准结果一模一样，证明成功。

2.学习 trace 测试文件符号(空格、&、#等)、命令、用户程序 myspin 含义

文件符号：

空格：用来分隔命令和参数或者参数与参数；

&：一个命令以&结尾，shell 应该在后台运行它，否则在前台运行； #：注释，会被解释器忽略，但是会将其打印在终端显示出来

%：表示接下来使 JID（工作号）

命令：

命令有系统命令（用户自己写的c语言代码实现的命令）和内置（内建）命令（linux系统自动支持的命令）：

tsh应该支持的内置命令有如下几条：

1）quit（结束进程）

2）jobs（显示放到后台的作业）

3）bg（重新启动某个job并在后台运行）

4）fg（重新启动某个job并在前台运行）

系统命令实际上就是所写的用户程序myint、myspin、mysplit、mystop，下面进行解释：

myint程序为一个c语言代码，用法是myint ，用途是测试我们的shell，实际上实现的是睡眠函数的功能，让程序睡眠n秒，运行结束后不会自动退出，并会检测系统错误；

mysplit程序为一个c语言代码，用法是mysplit ，用途是测试我们的shell，实际上实现的是睡眠函数的功能，程序睡眠n秒，但它是创建一个子进程进行睡眠，然后父进程等待子进程正常睡眠n秒后，继续运行；

mystop程序为一个c语言代码，用法是mystop ，用途是测试我们的shell，实际上实现的是让进程暂定n秒，并发送信号。

myspin 的含义：

myspin程序为一个c语言实现的可执行程序，用法是myspin ，用途是测试我们的shell，实际上实现的是休眠函数的功能，使程序休眠n秒，在休眠结束后就自动退出，不检测系统错误。

3.编程实现 jobs 内建命令，使用 trace05 验证

想要实现内建命令需要从builtin_cmd函数入手，只需在原有的 builtin_cmd 函数中添加一个判断函数，如果参数是 jobs，则执行 listjobs 函数（打印所有的作业）代码如下：

```
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0],"quit"))
    {
        exit(0);
    }
    if(!strcmp(argv[0],"jobs"))
    {
        listjobs(jobs);
        return 1;
    }
    return 0;
}
```

测试结果：先打印三行注释，然后在后台运行 myspin 函数，计算机沉睡两秒，然后在后台运行 myspin 函数，计算机沉睡三秒，然后展示所有的后台作业。

可以看到尽管有部分参数是有区别的（这是因为机器本身），其余都是相同的，所以证明结果正确。

4.了解作业、前台、后台、进程组的概念

作业：

Shell 分前后台来控制的不是进程而是作业（Job）或者进程组（Process Group）。一个前台作业可以由多个进程组成，一个后台也可以由多个进程组成， Shell 可以运行一个前台作业和任意多个后台作业，这称为作业控制。

&：使作业在后台运行

ctrl+z：暂停作业

Jobs：查看当前的作业

bg：使作业在后台运行

ctrl+c：杀死前台作业

前台：

前台进程就是用户使用的有控制终端的进程。前台的程序和用户交互，需要较高的响应速度，优先级别稍微高一点。

后台：

后台进程也叫守护进程（Daemon），是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。守护的意思就是不受终端控制。Linux 的大多数服务器就是用守护进程实现的。同时，守护进程完成许多系统任务。后台程序基本上不和用户交互，优先级别稍微低一点。

进程组：

进程组就是一些进程的组合。这些进程并不是孤立的，他们彼此之间或者存在父子、兄弟关系，或者在功能上有相近的联系。

进程必定属于一个进程组，也只能属于一个进程组。 一个进程组中可以包含多个进程。 进程组的生命周期从被创建开始，到其内所有进程终止或离开该组。

作业与进程组的区别：

通常，进程组与同一作业相关联。如果进程组中的某个进程又fork出子进程，子进程也属于同一进程组，但是Shell并不知道子进程的存在，也不会调用wait等待它结束。换句话说，原来的进程组是Shell的作业，而这个子进程不是，这是作业和进程组在概念上的区别。