哈爾濱工業大學

实验报告

实 验（七）

题　　目 TinyShell

微壳

专　　业 计算机类

学　　号 1190200526

班　　级 1903002

学　　生 沈城有

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.6.4

计算机科学与技术学院

目 录

第 1 章 实验基本信息 ..... - 4 -
1.1 实验目的 ..... - 4 -
1.2 实验环境与工具 ..... - 4 -
1.2.1 硬件环境 ..... - 4 -
1.2.2 软件环境 ..... - 4 -
1.2.3 开发工具 ..... - 4 -
1.3 实验预习 ..... - 4 -
第 2 章 实验预习 ..... - 5 -
2.1 进程的概念、创建和回收方法（5 分） ..... - 5 -
2.2 信号的机制、种类（5 分） ..... - 5 -
2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分） ..... - 7 -
2.4 什么是 shell，功能和处理流程（5 分） ..... - 8 -
第 3 章 TinyShell 的设计与实现 ..... - 9 -
3.1.1 void eval(char *cmdline)函数（10 分） ..... - 9 -
3.1.2 int builtin_cmd(char **argv)函数（5 分） ..... - 9 -
3.1.3 void do_bgfg(char **argv) 函数（5 分） ..... - 10 -
3.1.4 void waitfg(pid_t pid) 函数（5 分） ..... - 10 -
3.1.5 void sigchld_handler(int sig) 函数（10 分） ..... - 10 -
第 4 章 TinyShell 测试 ..... - 12 -
4.1 测试方法 ..... - 12 -
4.2 测试结果评价 ..... - 12 -
4.3 自测试结果 ..... - 12 -
4.3.1 测试用例 trace01.txt ..... - 12 -
4.3.2 测试用例 trace02.txt ..... - 12 -
4.3.3 测试用例 trace03.txt ..... - 13 -
4.3.4 测试用例 trace04.txt ..... - 13 -
4.3.5 测试用例 trace05.txt ..... - 13 -
4.3.6 测试用例 trace06.txt ..... - 13 -
4.3.7 测试用例 trace07.txt ..... - 14 -
4.3.8 测试用例 trace08.txt ..... - 14 -
4.3.9 测试用例 trace09.txt ..... - 14 -
4.3.10 测试用例 trace10.txt ..... - 15 -
4.3.11 测试用例 trace11.txt ..... - 15 -
4.3.12 测试用例 trace12.txt ..... - 15 -
4.3.13 测试用例 trace13.txt ..... - 16 -
4.3.14 测试用例 trace14.txt ..... - 16 -
4.3.15 测试用例 trace15.txt ..... - 17 -
第 5 章 评测得分 ..... - 18 -
第 6 章 总结 ..... - 19 -

5.1 请总结本次实验的收获 ..... - 19 -
5.2 请给出对本次实验内容的建议 ..... - 19 -
参考文献 ..... - 20 -

第 1 章 实验基本信息

1.1 实验目的

理解现代计算机系统进程与并发的基本知识；
掌握 linux 异常控制流和信号机制的基本原理和相关系统函数；
掌握 shell 的基本原理和实现方法；
深入理解 Linux 信号响应可能导致的并发冲突及解决方法；
培养 Linux 下的软件系统开发与测试能力。

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

Visual Studio Code；gcc 等

1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）。

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

了解进程、作业、信号的基本概念和原理；
了解 shell 的基本原理；
熟知进程创建、回收的方法和相关系统函数；
熟知信号机制和信号处理相关的系统函数。

第 2 章 实验预习

总分 20 分

2.1 进程的概念、创建和回收方法（5 分）

（1）进程的概念

进程是计算机中的程序关于某数据集合上的一次运行活动，是操作系统进行资源分配和调度的基本单位。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。

（2）进程的创建

Linux 操作系统中，创建进程使用 fork()函数，调用 fork()的进程称为父进程，新产生的进程称为子进程。fork 系统调用从内核返回两次：一次返回到父进程（子进程 PID），另一次返回到新产生的子进程（0）。

新创建的子进程几乎但不完全与父进程相同，子进程得到与父进程虚拟地址空间、打开文件描述符等相同的但又独立的一份副本。二者最大的区别是不同的 PID。

（3）进程的回收方法

当进程终止时，它仍然消耗系统资源，被称为“僵尸”(zombie)进程。父进程通过 wait（或 waitpid）等函数等待子进程结束并回收子进程，假如父进程未回收子进程就退出，init 进程会负责回收所有孤儿进程。

2.2 信号的机制、种类（5 分）

（1）信号的机制

信号就是一条小消息，它通知进程系统中发生了一个某种类型的事件。

信号常从内核发送到（有时是在另一个进程或进程本身的请求下）一个进程。

信号类型是用小整数 ID 来标识的，其中唯一的信息是它的 ID 和它的到达。

内核通过更新目的进程上下文中的某个状态，发送（递送）一个信号给目的进程；当目的进程被内核强迫以某种方式对信号的发送做出反应时，它就接收了信号。

进程反应的方式：

- 忽略这个信号；
- 终止进程；
- 通过执行一个称为信号处理程序(signal handler)的用户层函数捕获这个信

号，执行一些任务。

（2）信号的种类

课程所讲的信号都是不可靠信号，如下表：

编号	信号名称	缺省动作	说明
1	SIGHUP	终止	终端线被挂断
2	SIGINT	终止	键盘产生的中断(Ctrl-C)
3	SIGQUIT	终止	键盘产生的退出
4	SIGILL	终止	非法指令
5	SIGTRAP	终止并转储内存	跟踪陷阱
6	SIGABRT / SIGIOT	终止并转储内存	来自 abort 函数的终止信号
7	SIGBUS / SIGEMT	终止	总线异常/EMT 指令
8	SIGFPE	终止并转储内存	浮点异常
9	SIGKILL	终止	杀死程序
10	SIGUSR1	终止	用户定义信号 1
11	SIGSEGV	终止并转储内存	非法内存地址引用
12	SIGUSR2	终止	用户定义信号 2
13	SIGPIPE	终止	向某个没有读取的管道中写入数据
14	SIGALRM	终止	时钟中断(闹钟)
15	SIGTERM	终止	软件终止信号
16	SIGSTKFLT	终止	协处理器栈错误
17	SIGCHLD	忽略	子进程停止或终止
18	SIGCONT	忽略	如进程停止则继续运行
19	SIGSTOP	停止直至下一个 18	不是来自终端的停止信号
20	SIGSTP	停止直至下一个 18	来自终端的停止信号
21	SIGTTIN	停止直至下一个 18	后台进程从终端读
22	SIGTTOU	停止直至下一个 18	后台进程向终端写
23	SIGURG	忽略	socket 紧急情况
24	SIGXCPU	终止	CPU 时间限制被打破
25	SIGXFSZ	终止	文件大小限制被打破
26	SIGVTALRM	终止	虚拟定时器期满
27	SIGPROF	终止	剖析定时器期满
28	SIGWINCH	忽略	窗口尺寸调整
29	SIGIO/SIGPOLL	终止	在某个描述符上可执行 I/O 操作
30	SIGPWR	终止	电源故障

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

（1）信号的发送方法

a) 进程可以通过 kill 等函数向包括它本身在内的其他进程发送一个信号，如果程序没有发送这个信号的权限，对 kill 函数的调用就将失败，而失败的常见原因是目标进程由另一个用户所拥有；

b) 从键盘发送信号输入 ctrl-c/ctrl-z 会导致内核发送一个 SIGINT/SIGTSTP 信号到前台进程组中的每个作业，SIGINT 默认情况是终止前台作业 SIGTSTP 默认情况是停止（挂起）前台作业；

c) 用 /bin/kill 程序发送信号

/bin/kill 程序可以向另外的进程或进程组发送任意的信号

例：/bin/kill -9 24818 发送信号 9(SIGKILL)给进程 24818

/bin/kill -9 -24817 发送信号 SIGKILL 至进程组 24817 中的每个进程（负的 PID 会导致信号被发送到进程组 PID 中的每个进程）

（2）信号的阻塞方法

- 隐式阻塞机制：内核默认阻塞与当前正在处理信号类型相同的待处理信号。例：一个 SIGINT 信号处理程序不能被另一个 SIGINT 信号中断，此时另一个 SIGINT 信号被阻塞。
- 显式阻塞进制：应用程序可以调用 sigprocmask 函数和它的辅助函数，明确地阻塞和解除阻塞选定的信号。

  辅助函数：

  sigempty(set)：初始化 set 为空集合。

  sigfill(set)：把每个信号都添加到 set 中。

  sigadd(set)：把指定的信号 signum 添加到 set 中。

  sigdel(set)：从 set 中删除指定的信号 signum。

（3）信号处理程序的设置方法

可以使用 signal 函数修改和信号 signum 相关联的默认行为：

handler_t *signal(int signum, handler_t *handler)

handler 的不同取值:

1.SIG_IGN: 忽略类型为 signum 的信号；

2.SIG_DFL: 类型为 signum 的信号行为恢复为默认行为；

3.用户定义的函数的地址，这个函数即为信号处理程序，只要进程接收到类型为 signum 的信号就会调用信号处理程序。

由于 signal 的语义在不同版本的 Unix 系统中可能存在差异，我们需要一个可

移植的信号处理函数设置方法，Posix 标准定义了 sigaction 函数，它允许用户在设置信号处理时明确指定他们想要的信号处理语义。

2.4 什么是 shell，功能和处理流程（5 分）

（1）shell 的定义

shell 是一个交互型应用级程序，代表用户运行其他程序。它是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。

（2）shell 的功能

shell 是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如此，shell 有自己的编程语言用于对命令的编辑，它允许用户编写由 shell 命令组成的程序。shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 shell 程序与其他应用程序具有同样的效果。

（3）shell 的处理流程

shell 首先检查命令是否是内部命令，若不是再检查是否是一个应用程序（这里的应用程序可以是 Linux 本身的实用程序，如 ls 和 rm，也可以是购买的商业程序，如 xv，或者是自由软件，如 emacs）。然后 shell 在搜索路径里寻找这些应用程序（搜索路径就是一个能找到可执行程序的目录列表）。如果键入的命令不是一个内部命令并且在路径里没有找到这个可执行文件，将会显示一条错误信息。如果能够成功找到命令，该内部命令或应用程序将被分解为系统调用并传给 Linux 内核。

简化处理流程：

1) 从终端读入输入的命令；
2) 将输入字符串切分获得所有的参数；
3) 如果是内置命令则立即执行；
4) 若不是则调用相应的程序执行；
5) shell 应该接受键盘输入信号，并对这些信号进行相应处理。

第 3 章 TinyShell 的设计与实现

总分 45 分

3.1 设计

3.1.1 void eval(char *cmdline)函数（10 分）

函数功能：解析和解释用户输入的命令行。
参　　数：char *cmdline 指向用户输入的命令行字符串的指针
处理流程：
(1) 判断用户输入的是否是 tsh 的 build-in 命令(quit, jobs, bg, fg)；
(2) 若是则直接执行，函数结束，若不是继续进行第(3)步；
(3) 调用 fork 函数创建一个子进程并在子进程的上下文中执行(execve)作业，同时父进程将作业添加到作业列表(job list)；
(4) 若在前台运行，函数需等待子进程作业结束再返回；若在后台运行，函数输出子进程相关信息并直接退出。

要点分析：
(1) 为消除 SIGCHLD、SIGINT 及 SIGSTP 信号的到达与作业列表添加作业操作之间的竞争，避免添加已终止作业至作业列表的情况发生，在添加作业至列表这一操作执行前，需要先将这几个信号暂时阻塞。直至作业添加完成后解除；
(2) 子进程会拥有与父进程相同但独立的阻塞信号集，为避免阻塞信号对子进程执行程序造成影响，子进程在调用 execve 加载并运行程序前需解除对这些信号的阻塞；
(3) 每个子进程必须有与独立的、不同的进程组 ID 以避免其在后台运行时受到用户键入 ctrl-c/ctrl-z（内核发送 SIGINT/SIGSTP 至相关进程）的影响。

3.1.2 int builtin_cmd(char **argv)函数（5 分）

函数功能：判断用户输入的命令是否为 build-in 命令(quit, jobs, bg, fg)，执行用户键入的 build-in 命令并返回 1 表示输入是 build-in 命令或返回 0 表示输入不是 build-in 命令。
参　　数：char **argv 指向命令行参数字符串地址数组的指针
处理流程：
　　可分为以下四种情况来处理：
(1) 输入命令为 quit，则以状态 0 退出 tsh；
(2) 输入命令为 jobs，则调用提供的帮助函数 listjobs()列出当前所有作业，随后返回 1 表示这是一条内置命令；
(3) 输入命令为 fg/bg 类，则调用 do_fgbg()函数来处理，随后返回 1 表示这是一条内置命令；
(4) 输入命令不是以上任何一种，则返回 0 表示这不是一条内置命令。

要点分析：
(1) 利用返回值传递判断结果，避免不必要的操作；
(2) 内置命令较少，逐个比较判断即可。

3.1.3 void do_bgfg(char **argv) 函数（5 分）

函数功能：执行 tsh 内置的 fg/bg 命令。
参　　数：char **argv 指向命令行参数字符串地址数组的指针
处理流程：
(1) 检查命令是否含有参数，若无参数，则为无效命令，输出提示信息后返回；
(2) 若有参数，则检查参数的类型，参数类型不合法则输出提示信息并返回；
(3) 根据参数类型调用不同的函数在作业列表中查找是否存在这一作业项，若不存在，输出提示信息并返回；
(4) 若存在，无论是 bg 命令还是 fg 命令，均要先发送 SIGCONT 信号至所选作业；
(5) 若为 bg 命令，则输出作业相关信息并返回；
(6) 若为 fg 命令，则调用 waitfg()函数等待此前台作业结束再返回。
要点分析：
(1) 检查参数合法性并使用帮助函数查找对应作业；
(2) 执行 fg/bg 操作时，注意修改作业列表中此作业的状态；
(3) 是否等待作业结束是前台作业与后台作业的主要区别；
(4) 向作业进程发送 SIGCONT 信号使其在暂停后继续运行。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：暂时阻断 tsh 的常规运行直至前台作业不再是指定 PID 的进程。
参　　数：pid_t pid 前台作业进程的 PID
处理流程：按照实验建议，函数使用帮助函数 fgpid()获取当前前台作业的 PID，进行 busy loop 每次循环 sleep(1)，直至前台作业 PID 不再等于参数 pid（说明前台作业终止或停止）。
要点分析：
(1) 调用帮助函数 fgpid()获取当前前台作业的 PID；
(2) 使用 while 作为 busy loop，每次循环 sleep(1)。

3.1.5 void sigchld_handler(int sig) 函数（10 分）

函数功能：捕获 SIGCHLD 信号，回收所有僵死进程后立即返回（不等待其他正在运行的子进程终止）。
参　　数：sig 信号（SIGCHID）
处理流程：

首先暂存之前的 errno 以便函数结束前恢复。

随后使用 while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) 循环，检查已终止或被停止的进程，输出相关提示信息并回收僵死进程。

主要分为以下三种情况：

1) 子进程为正常终止的进程：阻塞所有信号，从作业列表中删除此作业，然后解除阻塞；
2) 子进程为被停止的进程：阻塞所有信号，模仿 tshref 程序输出提示信息，在作业列表中更新此作业的状态（停止 ST），然后解除阻塞；
3) 子进程为由于信号终止的进程：阻塞所有信号，模仿 tshref 输出提示信息，从作业列表中删除此作业，然后解除阻塞。

处理完所有已终止或被停止的子进程后，恢复 errno 并返回。

要点分析：

(1) 不能使用 while 来回收进程，因为后台还可能有正在运行的进程，这样做的话会使 waitpid 等待后台进程结束，违背了 tsh 的要求；
(2) 使用 WNOHANG | WUNTRACED 选项调用 waitpid 以检查已终止和被停止的进程，并且实现立即返回；
(3) 需要考虑提示信息的输出格式；
(4) 若进程被停止，要更新作业列表中的状态，避免用户输入 jobs 命令时输出错误状态信息；
(5) 注意保存和恢复 errno 并在删除作业前阻塞所有信号避免操作被中断造成意料之外的问题；
(6) 使用 wait.h 中解释 status 参数的宏实现对子进程状态的区分及相关信息的获取。

3.2 程序实现（tsh.c 的全部内容）（10 分）

重点检查代码风格：

（1）用较好的代码注释说明——5 分

（2）检查每个系统调用的返回值——5 分

第 4 章 TinyShell 测试

总分 15 分

4.1 测试方法

针对 tsh 和参考 shell 程序 tshref，完成测试项目 4.1-4.15 的对比测试，并将测试结果截图或者通过重定向保存到文本文件(例如：./sdriver.pl -t trace01.txt -s ./tsh -a "-p" > tshresult01.txt)，并填写完成 4.3 节的相应表格。

4.2 测试结果评价

tsh 与 tshref 的输出在以下两个方面可以不同：

(1) pid

(2)测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令，每次运行的输出都会不同，但每个 mysplit 进程的运行状态应该相同。

除了上述两方面允许的差异，tsh 与 tshref 的输出相同则判为正确，如不同则给出原因分析。

4.3 自测试结果

填写以下各个测试用例的测试结果，每个测试用例 1 分。

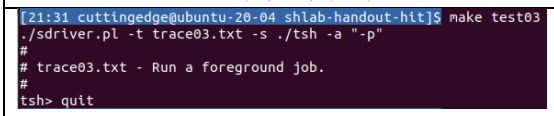
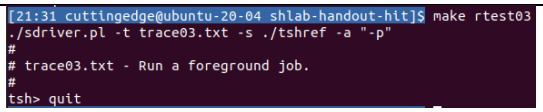
4.3.1 测试用例 trace01.txt

tsh 测试结果	tshref 测试结果
[21:27 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make test01 ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #	[21:28 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make rtest01 ./sdriver.pl -t trace01.txt -s ./tshref -a "-p" # # trace01.txt - Properly terminate on EOF. #
测试结论	相同

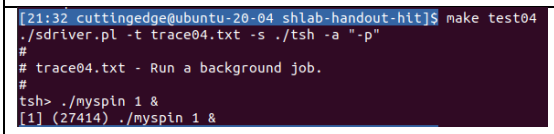
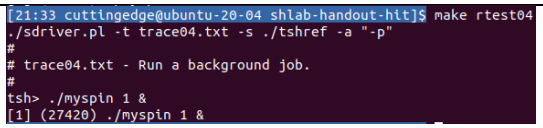
4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
[21:28 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. #	[21:30 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make rtest02 ./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. #
测试结论	相同

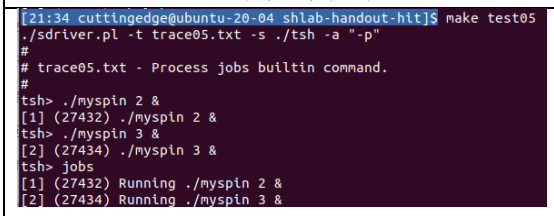
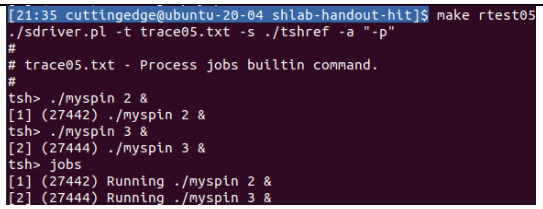
4.3.3 测试用例 trace03.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

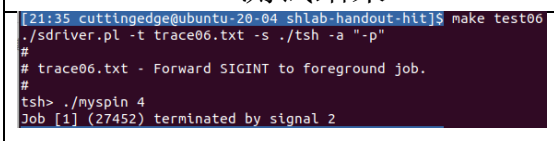
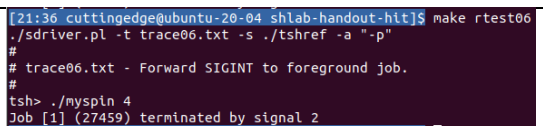
4.3.4 测试用例 trace04.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.5 测试用例 trace05.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.6 测试用例 trace06.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.7 测试用例 trace07.txt

tsh 测试结果	tshref 测试结果
[21:37 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make test07 ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (27466) ./myspin 4 & tsh> ./myspin 5 Job [2] (27468) terminated by signal 2 tsh> jobs [1] (27466) Running ./myspin 4 &	[21:38 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (27475) ./myspin 4 & tsh> ./myspin 5 Job [2] (27477) terminated by signal 2 tsh> jobs [1] (27475) Running ./myspin 4 &
测试结论	相同

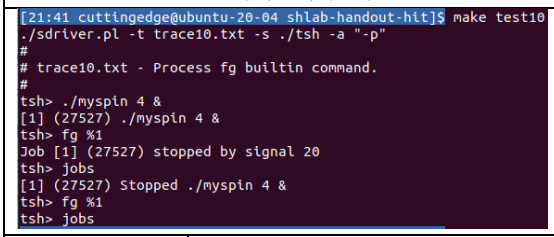
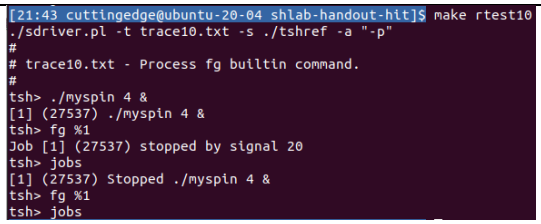
4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果
[21:38 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make test08 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (27484) ./myspin 4 & tsh> ./myspin 5 Job [2] (27486) stopped by signal 20 tsh> jobs [1] (27484) Running ./myspin 4 & [2] (27486) Stopped ./myspin 5	[21:40 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make rtest08 ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (27493) ./myspin 4 & tsh> ./myspin 5 Job [2] (27495) stopped by signal 20 tsh> jobs [1] (27493) Running ./myspin 4 & [2] (27495) Stopped ./myspin 5
测试结论	相同

4.3.9 测试用例 trace09.txt

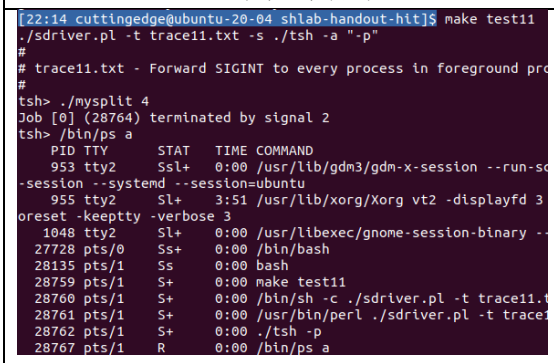
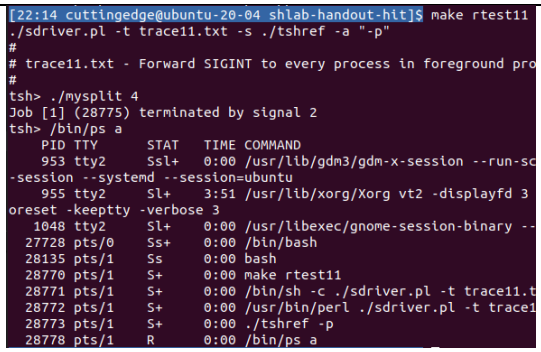
tsh 测试结果	tshref 测试结果
[21:40 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (27504) ./myspin 4 & tsh> ./myspin 5 Job [2] (27506) stopped by signal 20 tsh> jobs [1] (27504) Running ./myspin 4 & [2] (27506) Stopped ./myspin 5 tsh> bg %2 [2] (27506) ./myspin 5 tsh> jobs [1] (27504) Running ./myspin 4 & [2] (27506) Running ./myspin 5	[21:41 cuttingedge@ubuntu-20-04 shlab-handout-hit]$ make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (27515) ./myspin 4 & tsh> ./myspin 5 Job [2] (27517) stopped by signal 20 tsh> jobs [1] (27515) Running ./myspin 4 & [2] (27517) Stopped ./myspin 5 tsh> bg %2 [2] (27517) ./myspin 5 tsh> jobs [1] (27515) Running ./myspin 4 & [2] (27517) Running ./myspin 5
测试结论	相同

4.3.10 测试用例 trace10.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

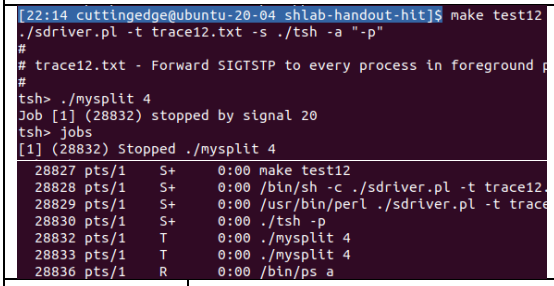
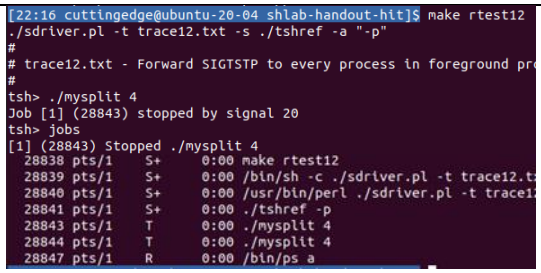
4.3.11 测试用例 trace11.txt

测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
	
测试结论	相同

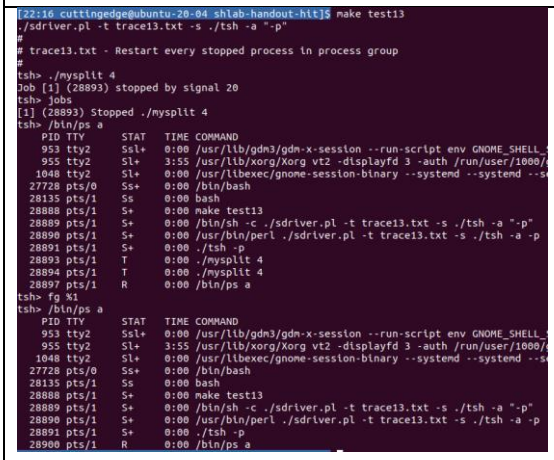
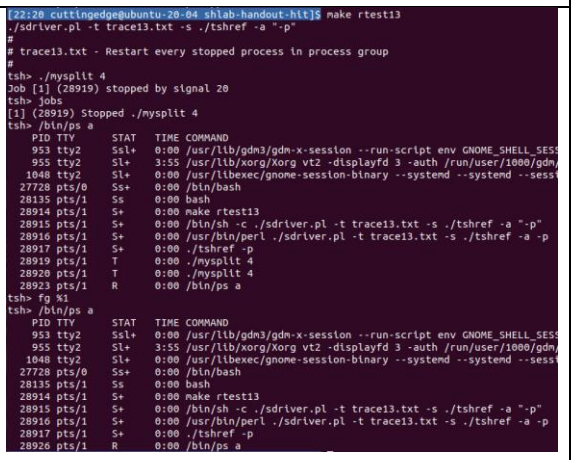
4.3.12 测试用例 trace12.txt

测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

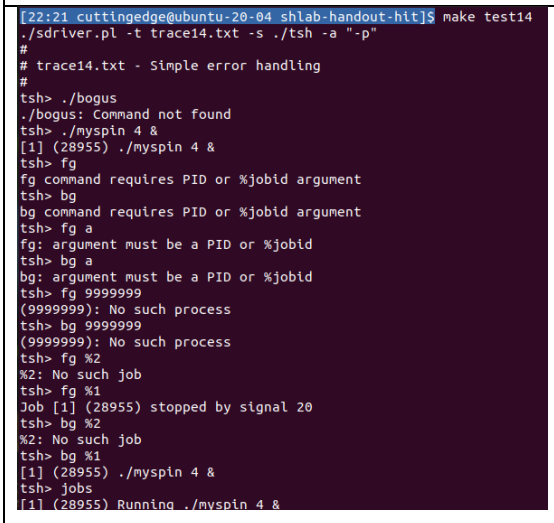
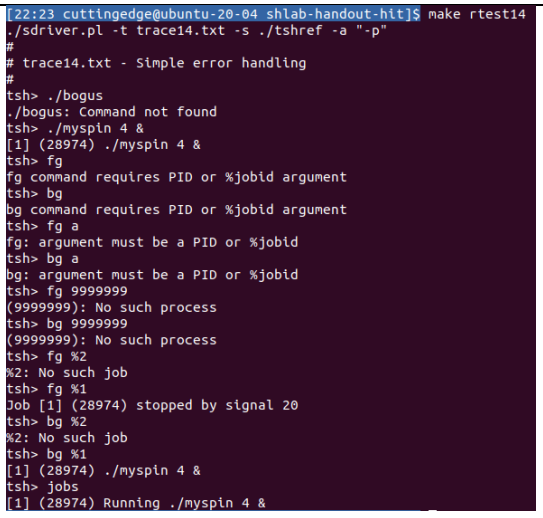
tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.13 测试用例 trace13.txt

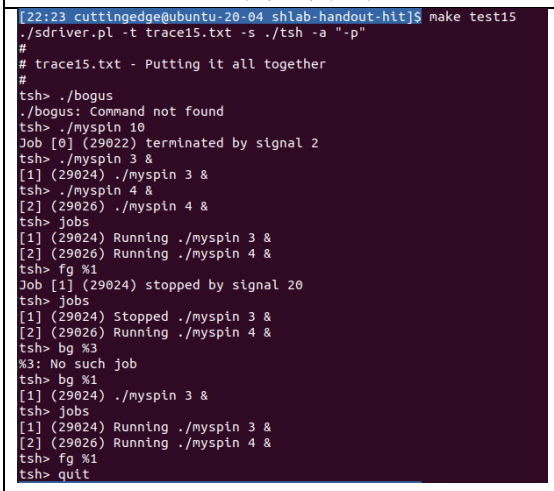
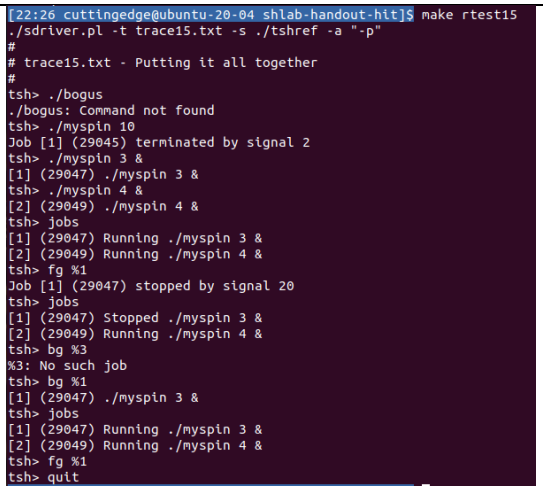
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.14 测试用例 trace14.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.15 测试用例 trace15.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

第 5 章 评测得分

总分 20 分

实验程序统一测试的评分（教师评价）：

（1）正确性得分：_____（满分 10）

（2）性能加权得分：_____（满分 10）

第 6 章 总结

6.1 请总结本次实验的收获

1. 深化了对异常控制流和信号机制的基本原理的理解；
2. 理解了操作系统中进程、进程组与并发的概念；
3. 掌握了进程、信号机制相关系统函数的简单使用；
4. 掌握了 shell 的基本原理及简易实现方法；
5. 初步学会了如何发现和解决信号机制可能导致的并发冲突问题。

6.2 请给出对本次实验内容的建议

建议自动化 tsh 测试（如果可能的话），人工逐个比较浪费了很多时间。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.