

**实验报告**

**实 验（四）**

题　　 目 微壳(TinyShell)

专　　 业 人工智能

学　　 号 2021112845

班　　 级 2103601

学 生 张智雄

指 导 老 师 郑贵滨

实 验 地 点 G.709

实 验 日 期 2023.4.30

**计算学部**

**目 录**

[第1章 实验基本信息 - 4 -](#_Toc136165136)

[1.1 实验目的 - 4 -](#_Toc136165137)

[1.2 实验环境与工具 - 4 -](#_Toc136165138)

[*1.2.1 硬件环境 - 4 -*](#_Toc136165139)

[*1.2.2 软件环境 - 4 -*](#_Toc136165140)

[*1.2.3 开发工具 - 4 -*](#_Toc136165141)

[1.3 实验预习 - 4 -](#_Toc136165142)

[第2章 实验预习 - 5 -](#_Toc136165143)

[2.1 进程的概念、创建和回收方法（5分） - 5 -](#_Toc136165144)

[2.2信号的机制、种类（5分） - 5 -](#_Toc136165145)

[2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5分） - 6 -](#_Toc136165146)

[2.4 什么是shell，简述其功能和处理流程（5分） - 7 -](#_Toc136165147)

[第3章 TinyShell的设计与实现 - 8 -](#_Toc136165148)

[3.1.1 void eval(char \*cmdline)函数（10分） - 8 -](#_Toc136165149)

[3.1.2 int builtin\_cmd(char \*\*argv)函数（5分） - 8 -](#_Toc136165150)

[3.1.3 void do\_bgfg(char \*\*argv) 函数（5分） - 9 -](#_Toc136165151)

[3.1.4 void waitfg(pid\_t pid) 函数（5分） - 10 -](#_Toc136165152)

[3.1.5 void sigchld\_handler(int sig) 函数（10分） - 11 -](#_Toc136165153)

[第4章 TinyShell测试 - 13 -](#_Toc136165154)

[4.1 测试方法 - 13 -](#_Toc136165155)

[4.2 测试结果评价 - 13 -](#_Toc136165156)

[4.3 自测试结果 - 13 -](#_Toc136165157)

[*4.3.1测试用例trace01.txt的输出截图（1分） - 13 -*](#_Toc136165158)

[*4.3.2测试用例trace02.txt的输出截图（1分） - 14 -*](#_Toc136165159)

[*4.3.3测试用例trace03.txt的输出截图（1分） - 14 -*](#_Toc136165160)

[*4.3.4测试用例trace04.txt的输出截图（1分） - 14 -*](#_Toc136165161)

[*4.3.5测试用例trace05.txt的输出截图（1分） - 15 -*](#_Toc136165162)

[*4.3.6测试用例trace06.txt的输出截图（1分） - 15 -*](#_Toc136165163)

[*4.3.7测试用例trace07.txt的输出截图（1分） - 15 -*](#_Toc136165164)

[*4.3.8测试用例trace08.txt的输出截图（1分） - 16 -*](#_Toc136165165)

[*4.3.9测试用例trace09.txt的输出截图（1分） - 16 -*](#_Toc136165166)

[*4.3.10测试用例trace10.txt的输出截图（1分） - 17 -*](#_Toc136165167)

[*4.3.11测试用例trace11.txt的输出截图（1分） - 17 -*](#_Toc136165168)

[*4.3.12测试用例trace12.txt的输出截图（1分） - 17 -*](#_Toc136165169)

[*4.3.13测试用例trace13.txt的输出截图（1分） - 18 -*](#_Toc136165170)

[*4.3.14测试用例trace14.txt的输出截图（1分） - 18 -*](#_Toc136165171)

[*4.3.15测试用例trace15.txt的输出截图（1分） - 19 -*](#_Toc136165172)

[4.4 自测试评分 - 20 -](#_Toc136165173)

[第5章 总结 - 21 -](#_Toc136165174)

[5.1 请总结本次实验的收获 - 21 -](#_Toc136165175)

[5.2 请给出对本次实验内容的建议 - 21 -](#_Toc136165176)

[参考文献 - 22 -](#_Toc136165177)

# 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；2.30GHz；16G RAM；1.5THD disk

### 1.2.2 软件环境

Windows11 64位；Vmware Workstation 17 Pro；Ubuntu 22.10

### 1.2.3 开发工具

Visual Studio 2019 64位；CodeBlocks 64位；vim+gcc

## 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）

**进程：**一个执行程序中的实例，系统中的每个程序都运行在某个进程的上下文(context)中。

每次当用户通过向shell输入一个可执行目标文件的名字，运行程序时，shell 就会创建一个新的进程，然后就在这个新进程的上下文中运行这个可执行目标文件。应用程序也能够创建新进程，并且在这个新进程的上下文中运行它们自己的代码或其他应用程序。

**创建进程：**父进程调用fork函数创建一个新的运行的子进程。新创建的子进程几乎但不完全与父进程相同。子进程将得到与父进程用户级虚拟地址空间相同的(但是独立的)一份部分，包括代码和数据段、堆、共享库以及用户栈。而父进程与新创建的子进程之间最大的区别在于它们有不同的PID。

**回收方法：**当进程终止时，他仍然消耗系统资源，因而需要主动回收。父进程使用wait或waitpid函数来等待子进程终止或者停止。具体过程为：父进程执行回收（wait或waitpid）、父进程挂起直至收到子进程的退出状态、内核从系统中删除僵死子进程。

## 2.2信号的机制、种类（5分）

**信号：**一个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。每种信号类型都对应某种系统的事件。

低层的硬件异常是由内核异常处理程序处理的，通常用户进程不可见，信号提供了一种机制，通知用户进程发生了这些异常。

信号可以被进程阻塞（延迟接收处理）、忽略、捕获并自定义处理或执行默认处理行为。但信号不能累计，每一种信号最多只能被发送1次直到被接收并清空标志位。

下图2-1展示了Linux系统上支持的30中不同类型的信号，其类型使用小整数ID来标识的，信号中唯一的信息是它的ID和它的到达。



图2-1 Linux信号

## 2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5分）

**发送信号：**内核能通过更新目的进程上下文中的某个状态项struct sigpending pending)，来实现发送（递送）一个信号给目的进程。其方法包括以下四种：

* 1. 用/bin/kill程序发送信号。/bin/kill程序可以向另外的进程发送任意的信号。
  2. 从键盘发送信号。在键盘上输入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。
  3. 用kill函数发送信号。进程通过调用kill函数发送信号给其他进程（包括它们自己）。
  4. 用alarm函数发送信号。进程可以通过调用alarm函数从而向它自己发送SIGALRM信号

**阻塞信号：一**个进程可以有选择性地阻塞接收某种信号。当一种信号被阻塞时，它仍可以被发送，但是产生的待处理信号不会被接受。Linux提供阻塞信号的隐式和显式的机制。

1. 隐式阻塞机制：内核默认阻塞任何当前处理程序正在处理信号类型的待处理信号。
2. 显式阻塞机制：应用程序可以使用 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。

**处理程序的设置：**进程可以通过使用signal函数修改和信号相关联的默认行为。唯一的例外是SIGSTOP和SIGKILL,它们的默认行为是不能修改的。

* 1. 函数原型sighandler\_t \*signal(int sig, sighandler\_t \*handler)
  2. 功能：设置信号的处理函数或恢复默认函数行为。
  3. handler的不同取值：
  + SIG\_IGN：忽略类型为sig的信号
  + SIG\_DFL：类型为sig的信号行为恢复为默认行为
  + handler是用户自定义函数的地址，这个函数称为信号处理程序
  1. 返回值：
  + 设置成功，返回原处理函数指针
  + 设置失败，返回SIG\_ERR
  + #define SIG\_ERR(void (\*)()) -1

## 2.4 什么是shell，简述其功能和处理流程（5分）

**Shell：**一个交互型用户程序，代表用户运行其他程序。具体而言，shell是指为使用者提供操作界面的软件（command interpreter，命令解析器）。它类似于DOS下的COMMAND.COM和后来的cmd.exe。它接收用户命令，然后调用相应的应用程序。

**功能：**shell独立于内核，是连接内核和应用程序的桥梁。其提供了一个图形化或命令行式界面，用户能够通过访问这个界面访问操作系统内核的服务。

**处理流程：**

1. 读取从键盘输入的命令。
2. 判断命令是否正确，且将命令行的参数改造为系统调用execve()内部处理所要求的形式。
3. 终端进程调用fork()来创建子进程，自身调用wait()来等待子进程完成。
4. 当子进程运行时，调用execve()函数，同时根据命令的名字指定的文件到目录中查找可行性文件，调入内存并执行这个命令。
5. 当子进程完成处理后，向父进程报告，此时终端进程被唤醒，做完必要的判别工作后，再发提示符，让用户输入新命令。

# 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：解析和解释命令行的主例程**

**参 数：**char \*cmdline 命令行字符串的起始指针

**处理流程：**

1. 提取命令参数：调用parseline函数将输入命令行字符串cmdline分割为参数数组argv（忽略空命令行），并根据结尾字符是否为’&’返回标志量bg，决定该操作后台运行还是前台运行。
2. 判断内置函数：调用buildin\_cmd函数判断输入指令是否为内置命令，若为内置命令则立即执行，执行后返回。
3. 设置阻塞信号：若不为内置命令，阻塞SIGCHLD、SIGINT和SIGTSTP信号，避免目标命令与上述阻塞信号到达之间的竞争。
4. 创建子进程：创建一个子进程，解除子进程对上述信号的阻塞，并赋予子进程一个新的进程组ID，而后在子进程中调用execve函数载入并运行该作业。
5. 父进程控制：同时父进程将此子进程加入jobs列表，此时再解除父进程对上述信号的阻塞。根据标志量bg判断此作业是否后台运行，若否，父进程挂起，执行waitfg直至作业完成；若是，打印该作业pid等信息，等待下一命令的输入。

**要点分析：**

1. 将内置命令集中判断并封装，增加了代码可读性以及健壮性。
2. 在父进程中，阻塞SIGCHLD、SIGINT和SIGTSTP信号，直至将作业添加到作业列表中。避免在将作业添加到作业列表时，shell捕获并处理上述信号。
3. 每个新作业必须获得一个新的进程组ID，这样内核就不会向所有shell作业（包括tsh和tsh创建的进程）发送ctrl-c和ctrl-z信号。

## 3.1.2 int builtin\_cmd(char \*\*argv)函数（5分）

**函数功能：**识别并解释内置命令(quit, jobs, bg or fg)

**参 数：**char \*\*argv 指向参数数组argv的指针

**处理流程：**利用if-else语句将输入命令与内置命令逐一比较，若匹配则直接运行相应的指令后，返回1（quit指令直接退出）；反之，则返回0。

**要点分析：**

1. 可以用else if取代连续if，减少冗余匹配次数。
2. 注意到bg和fg的执行函数相同，因而可以合并到一个if语句中。



图3-1-2 builtin\_cmd函数代码

## 3.1.3 void do\_bgfg(char \*\*argv) 函数（5分）

**函数功能：**实现内置命令bg和fg

**参 数：**char \*\*argv 指向参数数组argv的指针

**处理流程：**

1. 参数检查：检查bg或fg之后是否有PID或%JID等参数，若为空输出提示信息后返回；若不为空则继续执行。
2. 解析参数：判断参数属于PID还是%JID，提取信息后在进程列表jobs中寻找对应工作，若找不到则输出“No such process”后返回。
3. 具体执行：判断执行bg还是fg。利用kill函数向PID或%JID对应进程发送SIGCONT信号，将其工作状态置为相应的BG或FG。其中对于fg指令，需要通过waitfg函数等待目标进程结束后再返回。

**要点分析：**

1. 需要正确区分PID或%JID，最直接的方法是根据第一位是否为‘%’。
2. 需在程序各个阶段对输入进行检查，对非法输入的处理考虑要周全。
3. 利用kill函数实现一个进程向另一个进程发送信号。
4. 对于fg指令，需调用waitfg等待目标程序在前台运行结束后才返回。

## 3.1.4 void waitfg(pid\_t pid) 函数（5分）

**函数功能：**等待一个前台作业结束

**参 数：**pid\_t pid进程识别号

**处理流程：**

1. 调用fgpid函数检查当前前台作业的PID是否改变
2. 若改变即当前前台作业已结束，跳出循环，waitfg函数返回；反之则使用sigsuspend函数将当前进程挂起，等待下次循环条件判断。

**要点分析：**

1. fgpid函数返回当前前台作业的PID，如果没有，则返回0；利用其返回值是否改变判断目标进程是否结束。
2. 可以采取在循环体内插入pause，但pause会引入竞争，在while测试后和pause之前pid改变，pause会永远睡眠。
3. 也可选择采用sleep函数，但因为没有很好的办法确定休眠的间隔，仍存在资源的浪费。
4. 因而采用sigsuspend函数，不设置对任何信号的阻塞（在此与pause方法几乎等价），但增强了代码的可移植性。



图3-1-4 waitfg函数代码

## 3.1.5 void sigchld\_handler(int sig) 函数（10分）

**函数功能：**捕获并处理SIGCHILD函数

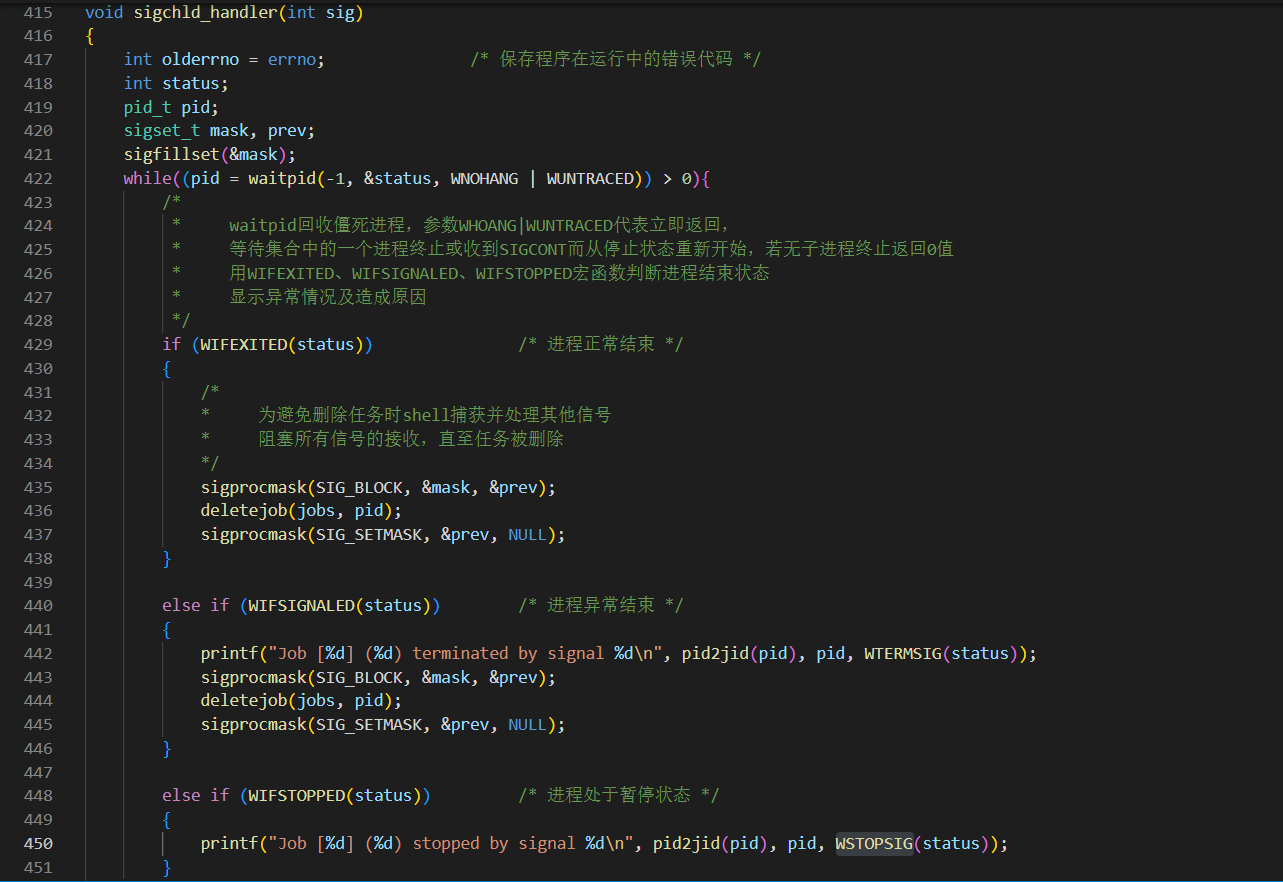
**参 数：**int sig信号小整数ID

**处理流程：**

1. 保护现场：存储程序在运行中的错误代码errno，防止处理过程中发生其他系统级的错误导致errno被修改，而影响进程上下文的正常运行。
2. 进程回收：调用waitpid函数回收僵死进程，参数WHOANG|WUNTRACED代表立即返回，等待集合中的一个进程终止或收到SIGCONT而从停止状态重新开始，若无子进程终止返回0值。
3. 状态检查：用WIFEXITED、WIFSIGNALED、WIFSTOPPED宏函数判断进程结束状态，正常结束则直接从工作列表删除此工作即可；异常情况或处于暂停状态则显示异常信息及造成原因，进程意外终止也需从工作列表删除。
4. 恢复现场：还原errno，函数返回。

**要点分析：**

1. 对处理前的进程信息进行保护，处理完成后恢复，以免发生意外错误。
2. waitpid函数立即返回，目的是实时回收，获取所有可用的僵尸子进程，但不等待任何其他当前正在运行的子进程终止。
3. 对异常情况使用WTERMSIG，WSTOPSIG追溯造成原因。
4. 为避免删除任务时shell捕获并处理其他信号，删除时阻塞所有信号的接收，直至任务被删除完成。



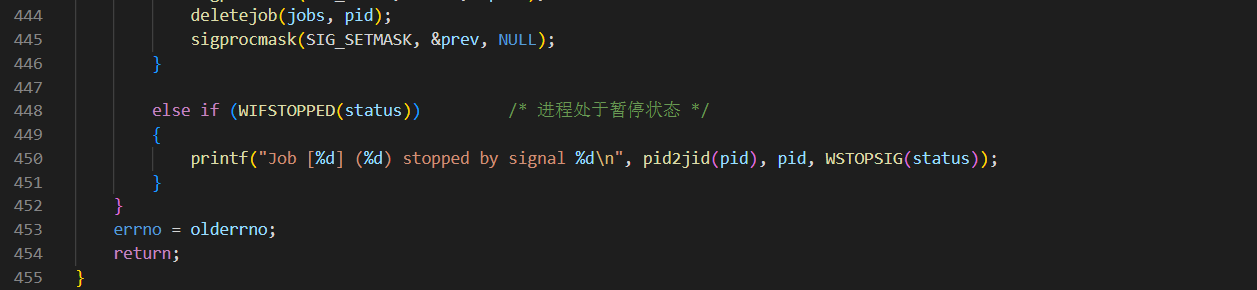


图3-1-5 sigchld\_handler函数代码

**3.2 程序实现（tsh.c的全部内容）（10分）**

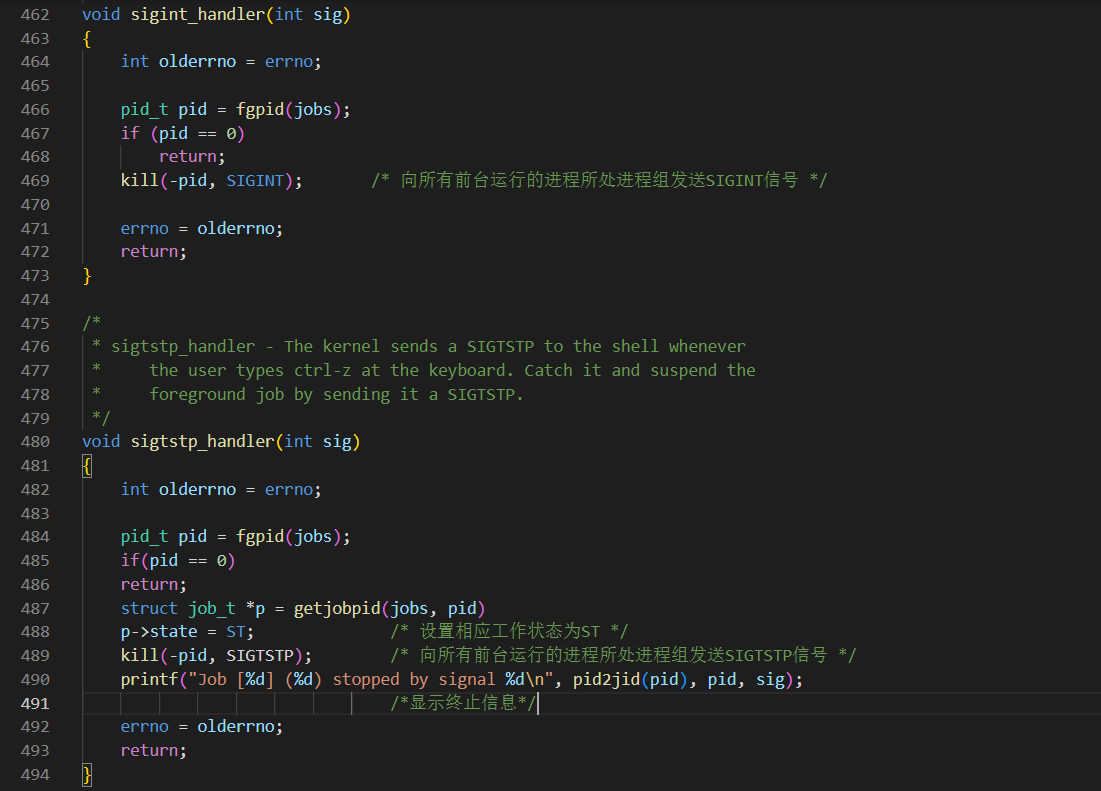
****

图3-2 其余作业要求函数代码

**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4.3.1-4.3.15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)。

## 4.2 测试结果评价

tsh与tshref的输出在一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

### 4.3.1测试用例trace01.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.2测试用例trace02.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.3测试用例trace03.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.4测试用例trace04.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
|  | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.5测试用例trace05.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.6测试用例trace06.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
|  | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.7测试用例trace07.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.8测试用例trace08.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.9测试用例trace09.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.10测试用例trace10.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
|  | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.11测试用例trace11.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.12测试用例trace12.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.13测试用例trace13.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
|  | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.14测试用例trace14.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成  文本  描述已自动生成  文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

### 4.3.15测试用例trace15.txt的输出截图（1分）

|  |  |  |
| --- | --- | --- |
| tsh测试结果 | | tshref测试结果 |
| 文本  描述已自动生成  文本  描述已自动生成 | | 文本  描述已自动生成 |
| 测试结论 | 相同 | |

## 4.4 自测试评分

根据节4.3的自测试结果，程序的测试评分为： 满分15分 。

# 第5章 总结

## 5.1 请总结本次实验的收获

1. 熟悉了进程fork、execve、waitpid/wait、kill等相关函数，对进程的工作原理有了更深的理解。
2. 熟悉了信号机制，信号发送与信号接收、信号处理的方法，尝试运用signal、sigprocmask等常见信号系统函数。
3. 对shell的工作原理进一步熟悉，了解了用户、shell、内核之间的交互过程。

## 5.2 请给出对本次实验内容的建议

1. 希望能增加对程序源码部分的讲解，对sigchld\_handler函数补写部分的任务描述及要求不是很清晰。
2. 希望增加对tinyshell如何进行调试等相关内容的讲解。

注：本章为酌情加分项。

# 参考文献

[1]RANDALE.BRYANT, DAVIDR.O’HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.

[2]C++信号处理signal(SIGINT, signalHandler) https://blog.csdn.net/u013288190/arti cle/details/119957441

[3]LINUX C中sigprocmask()函数用法https://blog.csdn.net/ShaoLiang\_Ge/article/ details/57984123

[4] C语言中errno与perror函数https://blog.csdn.net/hou09tian/article/details/910393

[5] linux c学习笔记----信号(sigaction,sigaddset,sigprocmask) https://blog.csdn.net/ weixin\_43667308/article/details/85250003