**电子科技大学**信息与软件工程**学院**

**标 准 实 验 报 告**

**（实验）课程名称 操作系统基础实验**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：成棋伟 学 号：2022130102013**

**指导教师： 刘瑶**

**实验地点：西306 实验时间： 2025年5月12日**

1. **实验室名称：**

信软楼西306

1. **实验项目名称：**

文件状态测试

1. **实验学时：**

4学时

1. **实验原理：**

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。

Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。

Ken Thompson 的 sh 是第一种 Unix Shell，Windows Explorer 是一个典型的图形界面 Shell。

Shell 脚本（shell script），是一种为 shell 编写的脚本程序。

业界所说的 shell 通常都是指 shell 脚本，但读者朋友要知道，shell 和 shell script 是两个不同的概念。

由于习惯的原因，简洁起见，本文出现的 "shell编程" 都是指 shell 脚本编程，不是指开发 shell 自身。

1. **实验目的：**

熟悉UNIX的基本SHELL程序设计方法，包括命令行参数检测、变量设置、文件状态检测与特定信息读取、程序运行控制。

1. **实验内容：**

编写一个SHELL程序，动态检测指定文件的状态信息，当文件的大小发生改变时，给出提示信息，并继续前进检测。当文件的大小的变化次数或持续检查无变化次数达到一定值时，退出检查，程序结束。

1. **实验器材（设备、元器件）：**

PC计算机，操作系统：Ubuntu

1. **实验步骤：**

1.编写、调试shell程序

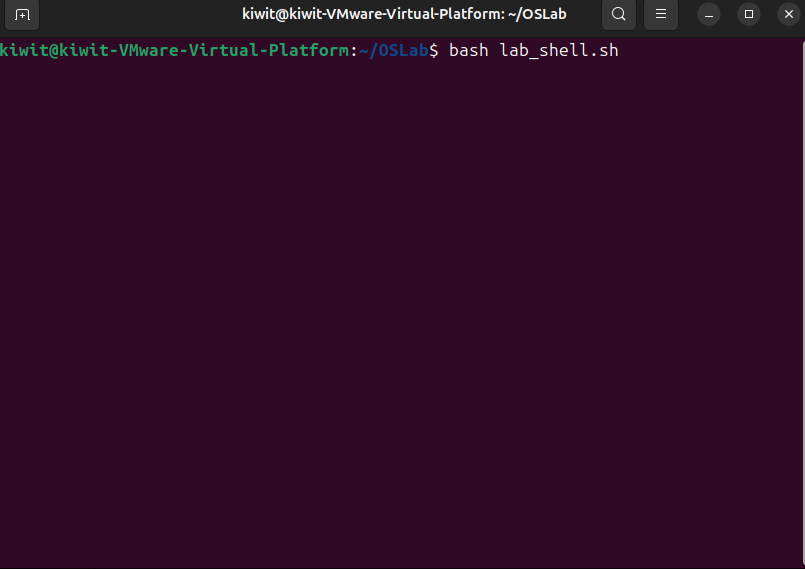
2.运行shell程序并观察结果

1. **实验数据及结果分析：**

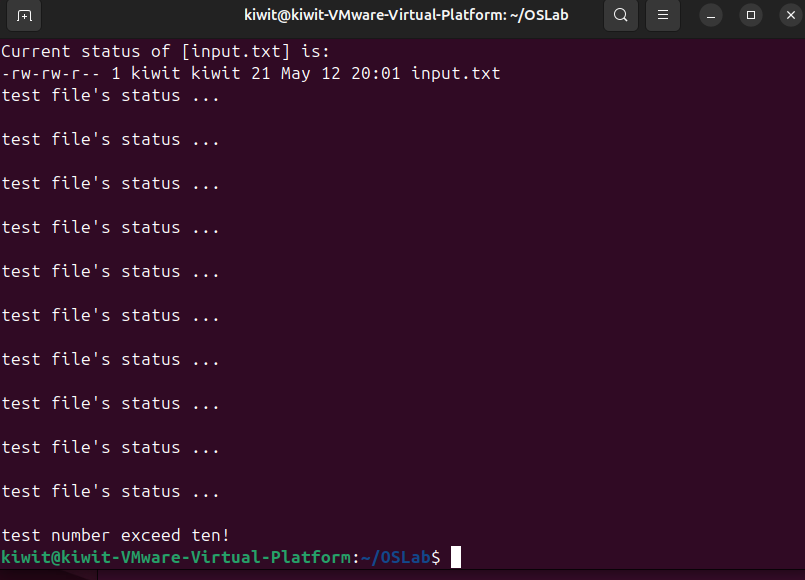
shell程序代码如下：

|  |
| --- |
| # 清空终端屏幕  clear  # 输入文件名  echo "input file name: "  read filename  # 在 ls -l 命令的输出中查找指定文件名，并保存在变量 filestatus 中  filestatus=`ls -l|grep $filename`  # 如果查找失败，则报错并退出脚本  if [ ${#filestatus} = 0 ]  then      echo "Can't find the file [$filename]"      exit  fi  # 输出文件信息并保存文件大小  echo "Current status of [$filename] is: "  echo $filestatus  filesize=`ls -l | grep $filename | awk '{print $5}'`  # 循环次数，初始化为 0  cnt1=0  # 改变次数，初始化为 0  cnt2=0  # 开始循环  while [ $cnt1 -lt 10 ]  do      sleep 5 # 等待 5 秒      echo "test file's status ..."      echo ""        # 再次在 ls -l 命令的输出中查找指定文件名，并保存在变量 filestatus 中      filestatus=`ls -l|grep $filename`        # 获取当前文件大小      filesize\_c=`ls -l | grep $filename | awk '{print $5}'`      # 如果文件大小发生了变化，则输出文件信息      if [ ! $filesize = $filesize\_c ]      then          cnt2=`expr $cnt2 + 1`          echo "file [$filename] size changed!"          echo $filestatus            # 如果发现两次以上文件大小发生了改变，测试结束          if [ $cnt2 = 2 ]          then              echo "Change number exceed two, test end!"              exit          else              filesize=$filesize\_c          fi      fi        cnt1=`expr $cnt1 + 1` # 循环次数自增  done  # 如果循环次数超过了 10 次，则测试失败  echo "test number exceed ten!"  exit |

创建文件：

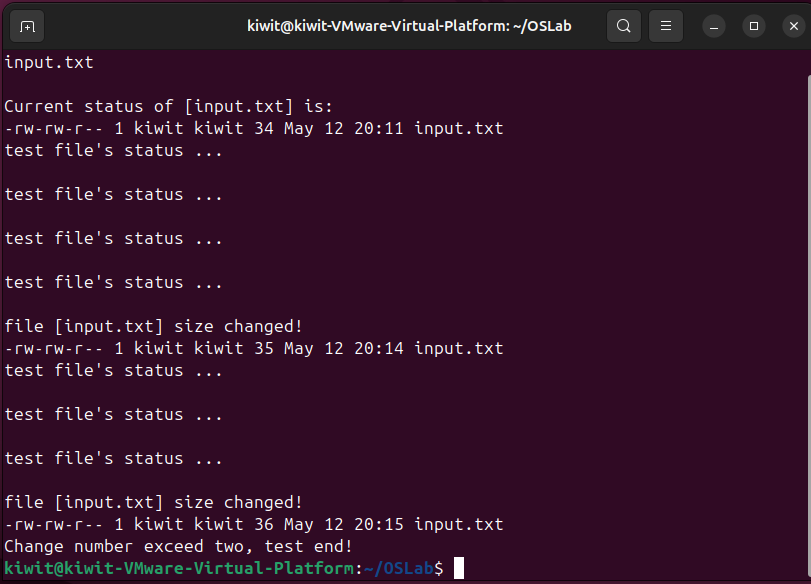


在不改变文件大小的情况下运行shell程序：



由截图可见10次检测后程序正常退出

在运行shell程序过程中更改文件内容：



运行过程中先后增加了1、2两个字符，由截图可见检测到文件大小更改两次后程序退出。

1. **总结及心得体会：**

不同于其他高级语言的编译执行，shell程序并不需要编译，而是直接执行，即使发现语句有错误也会依次执行后续语句。同时shell语言对空格的要求非常严格，如if后的方括号两侧必须有空格，变量赋值时等号两侧不能有空格等，与其他语言对空格的宽泛检测大相径庭。

通过本次实验我了解了shell语言的基本语法，熟悉了UNIX的基本SHELL程序设计方法，包括命令行参数检测、变量设置、文件状态检测与特定信息读取、程序运行控制。在这个过程中，使我更熟悉Linux中SHELL程序的编程流程，以及命令使用。同时，深入了我对 Linux 操作系统各个功能的理解和体会。

1. **实验改进与建议：**

在程序中添加可调参数，例如文件路径、检查的时间间隔、变化次数限制等，以便对程序进行各种场景的测试和验证。

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：成棋伟 学 号：2022130102013**

**指导教师： 刘瑶**

1. **实验地点：西306 实验时间： 2025年5月12日**
2. **实验室名称：**

信软楼西306

1. **实验项目名称：**

基于管道的进程间数据传输

1. **实验学时：**

4学时

1. **实验原理：**

首先创建两个子进程，注意Linux下使用fork()函数创建进程的方法。父进程和两个子进程间需要同步，使用waitpid（）函数实现父进程等待子进程运行完毕后从管道中读取数据并打印，只有子进程将数据写入管道后，父进程才能够执行打开管道操作。

由于fork函数让子进程完整地拷贝了父进程的整个地址空间，所以子进程都有管道的读端和写端。所以在相关进程中最好关掉不用的那一端。根据要求，“父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。”存在两个同步问题，两个子进程和父进程之间（先子写后父读）同步、子进程1和子进程2之间（先1写，再2写）

1. **实验目的：**

熟悉Linux下的应用程序开发；熟悉Linux的进程控制原语的使用；掌握Linux操作系统的进程间通信机制管道的使用；掌握Linux操作系统中父进程与子进程的同步。

1. **实验内容：**

在Linux系统中使用系统调用fork()创建两个子进程，使用系统调用pipe()建立一个管道，两个子进程分别向管道各写一句话：

Child process 1 is sending a message!

Child process 2 is sending a message!

而父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。然后分别结束两个子进程的运行。

要求：

1.父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

2.在Linux平台下实现。

1. **实验器材（设备、元器件）：**

PC计算机，操作系统：Ubuntu

1. **实验步骤：**

1.编写c语言程序

2.使用gcc将.c程序编译成可执行文件

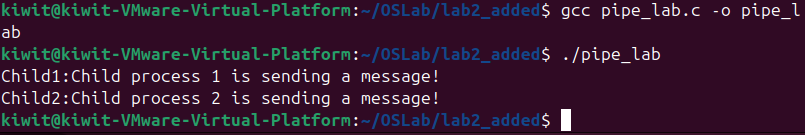
3.执行程序并观察结果

1. **实验数据及结果分析：**

程序代码如下：

|  |
| --- |
| #include<stdio.h>  #include<wait.h>  #include<pthread.h>  #include<stdlib.h>  #include<unistd.h>  #include<string.h>  int main(){      int fd[2]; //定义一个长度为2的int类型数组，用于存放管道的两个文件描述符      pid\_t child[2]; //定义一个长度为2的pid\_t类型数组，用于存放两个子进程的进程ID      if(pipe(fd)==-1){ //创建管道，fd[0]为读取端，fd[1]为写入端，如果返回值为-1说明创建失败          printf("error\n");      }      char buf1[]="Child process 1 is sending a message!";    //定义字符串buf1，存放子进程1要写入管道的信息      char buf2[]="Child process 2 is sending a message!";    //定义字符串buf2，存放子进程2要写入管道的信息      char buf3[100];             //定义一个长度为100的字符数组，用于存放从管道中读取到的信息        child[0]=fork();            //创建子进程1，获取进程ID      if(child[0]==0){            //子进程1执行以下代码块          close(fd[0]);           //关闭管道读取端          write(fd[1],buf1,strlen(buf1)+1);   //向管道写入buf1字符串，并添加结尾符，即strlen(buf1)+1个字符          exit(0);                //子进程结束      }      waitpid(child[0],NULL,0);   //等待子进程1结束      child[1]=fork();            //创建子进程2，获取进程ID      if(child[1]==0){            //子进程2执行以下代码块          close(fd[0]);           //关闭管道读取端          write(fd[1],buf2,strlen(buf2)+1);   //向管道写入buf2字符串，并添加结尾符，即strlen(buf2)+1个字符          exit(0);                //子进程结束      }      waitpid(child[1],NULL,0);   //等待子进程2结束      close(fd[1]);               //关闭管道写入端      read(fd[0],buf3,strlen(buf1)+1);    //从管道读取数据，存放到buf3中，读取的字符数为strlen(buf1)+1个字符      printf("Child1:%s\n",buf3); //输出读取到的数据      read(fd[0],buf3,strlen(buf2)+1);    //从管道读取数据，存放到buf3中，读取的字符数为strlen(buf2)+1个字符      printf("Child2:%s\n",buf3); //输出读取到的数据  } |

编译运行程序结果：



由截图可见，父进程成功将子进程通过管道传输的消息打印

1. **总结及心得体会：**

通过本次实验，我对进程间通信的机制有了进一步的了解。通过编程利用管道实现两个进程的通信，我对多进程的创建和使用掌握得更加熟练，同时在实践中对于管道的使用机制有了更深一步的了解。在这个过程中，也锻炼了我在Linux环境下编程实践的能力。通过实验课，对理论课上的内容，有了更加深入的理解。

1. **实验改进与建议：**

可以添加异常处理机制，来处理子进程fork()或者pipe()失败等导致的错误。

**报告评分：**

**指导教师签字：**