**操作系统原理**

**实验报告**

**专业: 通信工程**

**班级学号: 18班 20171817**

**学生姓名: 徐梓铭**

**实验地点: 一教502室**

**指导教师: 孙大洋**

**日期： 2019.10.30 2019.11.19**

**实验一 Linux操作系统基本命令与编程初步**

1. **实验目的**

（1）熟悉Linux系统下命令行操作方式；

（2）掌握Linux操作系统基本命令；

（3）了解shell编程。

**2、实验内容**

打开操作系统终端terminal，如图1所示。在命令行终端即可输入操作系统命令。

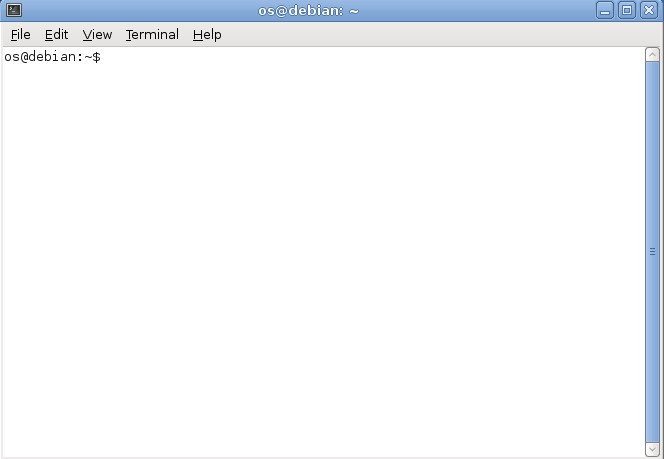


图1操作系统终端

对输入的操作系统命令结果可通过以下方式验证：在终端输入命令操作，通过可视化界面或相关操作系统命令查看操作结果。例如新建了文件夹，可使用ls命令查看是否新建成功，也可通过可视化界面查看。

本指导书仅提供相关命令简单说明，具体命令规范及细节请查阅相关资料学习。

本实验需要验证以下操作系统命令：

（1） 可通过以下方式查看操作系统命令的相关帮助，<command>为需要查看的操作系统命令名：

1. <command> --help
2. man <command>

（2）进入文件夹命令 cd

1. cd /home 进入 '/ home' 目录'
2. cd .. 返回上一级目录
3. cd ../.. 返回上两级目录
4. cd ~user1 进入user1个人的主目录
5. cd - 返回上次所在的目录

（3）创建目录命令 mkdir

1. mkdir dir1 创建一个叫做 'dir1' 的目录'
2. mkdir dir1 dir2 同时创建两个目录
3. mkdir -p /tmp/dir1/dir2 创建一个目录树
4. rmdir dir1 删除一个叫做 'dir1' 的目录'

（4）复制文件命令cp

1. cp file1 file2 复制一个文件
2. cp dir/\* . 复制一个目录下的所有文件到当前工作目录
3. cp -a /tmp/dir1 . 复制一个目录到当前工作目录
4. cp -a dir1 dir2 复制一个目录

（5）显示目录文件列表命令ls

1. ls
2. ls -al

（6）文件权限设置命令 chmod

Linux系统中的每个文件和目录都有访问许可权限，用来确定谁可以通过何种方式对文件和目录进行操作。文件或目录的访问权限分为只读，只写和可执行三种。Linux中又有三种不同类型的用户可对文件或目录进行访问：文件所有者，同组用户、其他用户。所有者可以允许同组用户有权访问文件，还可以将文件的访问权限赋予系统中的其他用户。文件被创建时，文件所有者自动拥有对该文件的读、写和可执行权限，以便于对文件的阅读和修改。用户也可根据需要把访问权限设置为需要的任何组合。  
　　Linux将每一文件或目录的访问权限分为三组，每组用三位表示，分别为文件所有者的读、写和执行权限；与所有者同组的用户的读、写和执行权限；系统中其他用户的读、写和执行权限。当用ls -l命令显示文件或目录的详细信息时，最左边的一列为文件的访问权限。   
　　chmod命令用于改变linux系统文件或目录的访问权限，其权限参数说明如下：  
u ：目录或者文件的当前的用户  
g ：目录或者文件的当前的群组  
o ：除了目录或者文件的当前用户或群组之外的用户或者群组  
a ：所有的用户及群组  
r ：读权限，用数字4表示  
w ：写权限，用数字2表示  
x ：执行权限，用数字1表示  
- ：删除权限，用数字0表示  
s ：特殊权限

chmod命令有两种用法。一种是包含字母和操作符表达式的文字设定法，如：

chmod ［who］ ［+ | - | =］ ［mode］ 文件名

chmod命令的另一种用法是包含数字的数字设定法：0表示没有权限，1表示可执行权限，2表示可写权限，4表示可读权限，然后将其相加。所以数字属性的格式应为3个从0到7的八进制数，其顺序是（u）（g）（o）。例如，如果想让某个文件的属主有“读/写”二种权限，需要把4（可读）+2（可写）＝6（读/写）。数字设定法的一般形式为：

chmod ［mode］ 文件名

数字与字符对应关系：r=4，w=2，x=1。若要rwx属性则4+2+1=7；若要rw-属性则4+2=6；若要r-x属性则4+1=7。

例：将文件 file1.txt 设为所有人皆可读取

chmod ugo+r file1.txt

或chmod 444 file1.txt

（7）正则表达式及grep命令

grep是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep命令在一个或多个文件中搜索字符串模板，模板后的所有字符串被看作文件名。搜索的结果被送到屏幕，不影响原文件内容。

grep可用于shell脚本，因为grep通过返回一个状态值来说明搜索的状态，如果模板搜索成功，则返回0，如果搜索不成功，则返回1，如果搜索的文件不存在，则返回2。用这些返回值就可进行一些自动化的文本处理工作。

grep命令实例：

1. grep 'test' d\* 显示所有以d开头的文件中包含test的行。
2. grep 'test' aa bb cc 显示在aa，bb，cc文件中匹配test的行。
3. grep '[a-z]\{5\}' aa 显示所有包含每个字符串至少有5个连续小写字符的字符串的行。'[a-z]\{5\}'为正则表达式模板，对此感兴趣的同学可查阅“正则表达式”相关资料学习。

（8）[Linux 管道命令](http://www.live-in.org/archives/908.html)

管道命令在Linux中十分有用，操作符为“|”，它能将经由前一个命令输出的标准输出，作为管道后一个命令的标准输入。

如：在hi.txt中查找字符串Hello

grep 'Hello' hi.txt

（9）编写shell程序并运行

从程序员的角度来看， Shell本身是一种用C语言编写的程序，从用户的角度来看，Shell是用户与Linux操作系统沟通的桥梁。用户既可以输入命令执行，又可以利用 Shell脚本编程，完成更加复杂的操作。

利用vi等文本编辑器编写Shell脚本的格式是固定的，如下：

#!/bin/sh

#comments

此处编写命令代码

首行中的符号#!告诉系统其后路径所指定的程序即是解释此脚本文件的Shell程序。如果首行没有这句话，在执行脚本文件的时候，将会出现错误。后续的部分就是主程序，Shell脚本像高级语言一样，也有变量赋值，也有控制语句。除第一行外，以#开头的行就是注释行，直到此行的结束。如果一行未完成，可以在行尾加上"，这个符号表明下一行与此行会合并为同一行。

编辑完毕，将脚本存盘为filename.sh，文件名后缀sh表明这是一个Bash脚本文件。执行脚本的时候，要先将脚本文件的属性改为可执行的：

chmod +x filename.sh

执行脚本的方法是：

./filename.sh

以下是最简单的shell脚本代码，经典的“hello world” Shell脚本：

#!/bin/sh

#print hello world in the console window

a = "hello world"

echo $a

Shell Script是一种弱类型语言，使用变量的时候无需首先声明其类型。新的变量会在本地数据区分配内存进行存储，这个变量归当前的Shell所有，任何子进程都不能访问本地变量。这些变量与环境变量不同，环境变量被存储在另一内存区，叫做用户环境区，这块内存中的变量可以被子进程访问。变量赋值的方式是：

variable\_name = variable\_value

如果对一个已经有值的变量赋值，新值将取代旧值。取值的时候要在变量名前加$，$variable\_name可以在引号中使用，这一点和其他高级语言是明显不同的。如果出现混淆的情况，可以使用花括号来区分，例如：

echo "Hi, $as"

就不会输出“Hi, hello worlds”，而是输出“Hi，”。这是因为Shell把$as当成一个变量，而$as未被赋值，其值为空。正确的方法是：

echo "Hi, ${a}s"

单引号中的变量不会进行变量替换操作。

关于变量，还需要知道几个与其相关的Linux命令。

env用于显示用户环境区中的变量及其取值；set用于显示本地数据区和用户环境区中的变量及其取值；unset用于删除指定变量当前的取值，该值将被指定为NULL；export命令用于将本地数据区中的变量转移到用户环境区。

带流程控制的简单shell代码如下：

#!/bin/sh

cd ~

mkdir shell\_tut

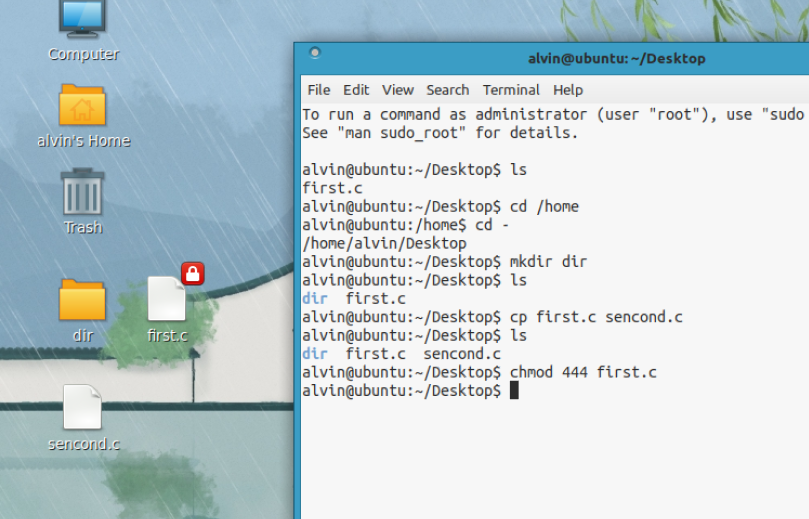
cd shell\_tut

for ((i=0; i<10; i++)); do

touch test\_$i.txt

done

**3、实验结果**



**实验二 进程间通信实验**

**1、实验目的**

掌握操作系统进程间通信技术。

**2、实验内容**

（2）Unix下进程创建实验，理解exec命令对进程控制块的操作。

#include <stdio.h>

#include <unistd.h>

int main()

{

pid\_t pid;

/\* fork a child process \*/

pid = fork();

if (pid < 0) { /\* error occurred \*/

fprintf(stderr, "Fork Failed");

return 1;

}

else if (pid == 0) { /\* child process \*/

execlp("/bin/ls","ls",NULL);

}

else { /\* parent process \*/

/\* parent will wait for the child to complete \*/

wait(NULL);

printf("Child Complete");

}

return 0;

}

（4）POSIX共享内存进程间通信实验。(Ubuntu 下需要-lrt库)

写共享内存进程示例代码如下：

#include <stdio.h>

#include <stlib.h>

#include <string.h>

#include <fcntl.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/mman.h>

int main()

{

/\* the size (in bytes) of shared memory object \*/

const int SIZE=4096;

/\* name of the shared memory object \*/

const char \*name = "OS";

/\* strings written to shared memory \*/

const char \*message\_0 = "Hello";

const char \*message\_1 = "World!";

/\* shared memory file descriptor \*/

int shm\_fd;

/\* pointer to shared memory obect \*/

void \*ptr;

/\* create the shared memory object \*/

shm\_fd = shm\_open(name, O\_CREAT | O\_RDWR, 0666);

/\* configure the size of the shared memory object \*/

ftruncate(shm\_fd, SIZE);

/\* memory map the shared memory object \*/

ptr = mmap(0, SIZE, PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

/\* write to the shared memory object \*/

sprintf(ptr,"%s",message\_0);

ptr += strlen(message\_0);

sprintf(ptr,"%s",message\_1);

ptr += strlen(message\_1);

return 0;

}

读共享内存进程示例代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/mman.h>

int main()

{/\* the size (in bytes) of shared memory object \*/

const int SIZE= 4096;

/\* name of the shared memory object \*/

const char \*name = "OS";

/\* shared memory file descriptor \*/

int shm\_fd; /\* pointer to shared memory obect \*/

void \*ptr;

/\* open the shared memory object \*/

shm\_fd = shm\_open(name, O\_RDONLY, 0666);

/\* memory map the shared memory object \*/

ptr = mmap(0, SIZE, PROT\_READ, MAP\_SHARED, shm\_fd, 0);

/\* read from the shared memory object \*/

printf("%s",(char \*)ptr);

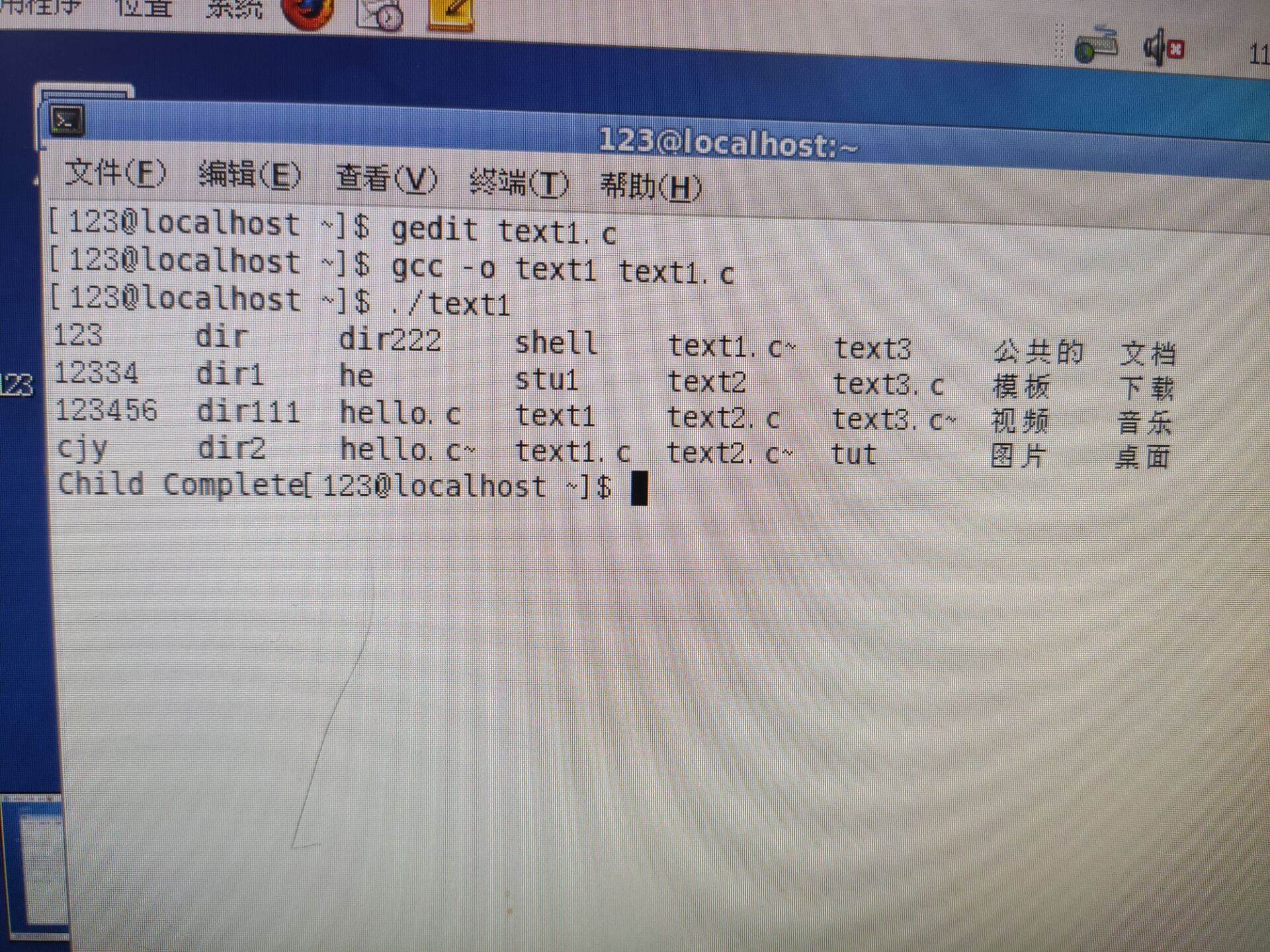
/\* remove the shared memory object \*/

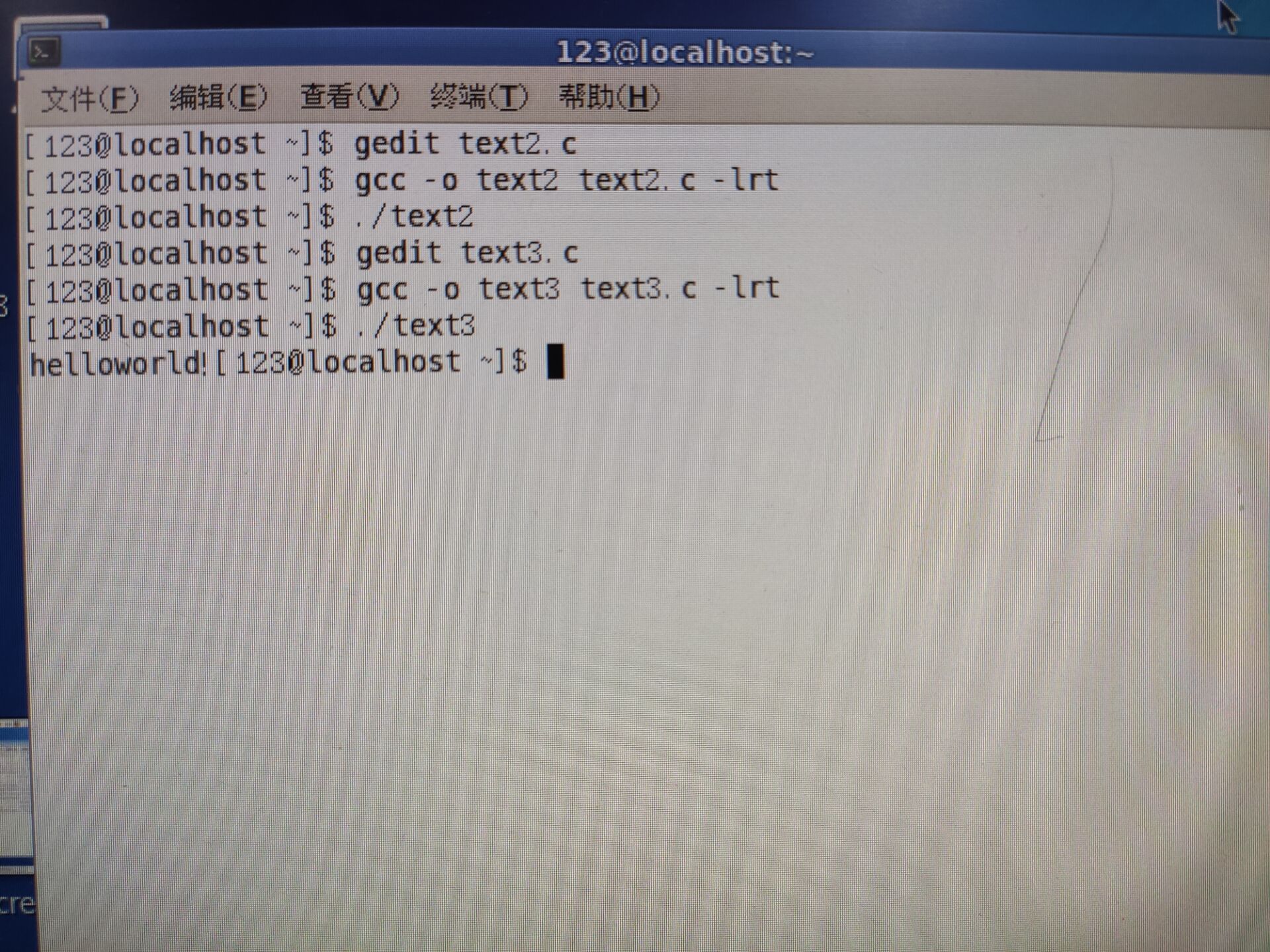
shm\_unlink(name);

return 0;

}

**3、实验结果**





**实验三 线程创建实验**

**1、实验目的**

掌握操作系统线程技术。

**2、实验内容**

（3）Linux下使用信号灯变量进行线程同步。

信号灯同步示例代码1：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: hello.c

\* DESCRIPTION:

\* A "hello world" Pthreads program. Demonstrates thread creation and termination.

\* AUTHOR: Blaise Barney

\* LAST REVISED: 08/09/11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

char str[64]="";

sem\_t sem;

void \*runner(void \*threadid)

{

sem\_wait(&sem);

printf("hello child\n");

// printf("%s",str);

pthread\_exit(NULL);

}

int main(int argc, char \*argv[])

{

pthread\_t thread;

int rc;

long t;

sem\_init(&sem,0,0);

rc = pthread\_create(&thread, NULL, runner, (void \*)t);

if (rc){

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

}

scanf("%s",str);

sem\_post(&sem);

pthread\_join(thread,NULL);

printf("Parent hello\n");

pthread\_exit(NULL);

}

信号灯同步示例代码2：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: hello.c

\* DESCRIPTION:

\* A "hello world" Pthreads program. Demonstrates thread creation and termination.

\* AUTHOR: Blaise Barney

\* LAST REVISED: 08/09/11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

int sharevalue=0;

sem\_t sem;

void \*runner1(void \*threadid)

{

sem\_wait(&sem);

while(sharevalue<20)

{

if(sharevalue%2==0)

printf("thread1:%d\n",sharevalue);

sharevalue++;

}

sem\_post(&sem);

pthread\_exit(NULL);

}

void \*runner2(void \*threadid)

{

sem\_wait(&sem);

while(sharevalue<20)

{

if(sharevalue%3==0)

printf("thread2:%d\n",sharevalue);

sharevalue++;

}

sem\_post(&sem);

pthread\_exit(NULL);

}

int main(int argc, char \*argv[])

{

pthread\_t thread1,thread2;

int rc;

long t;

sem\_init(&sem,0,1);

rc = pthread\_create(&thread1, NULL, runner1, (void \*)t);

if (rc){

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

}

rc = pthread\_create(&thread2, NULL, runner2, (void \*)t);

if (rc){

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

}

pthread\_join(thread1,NULL);

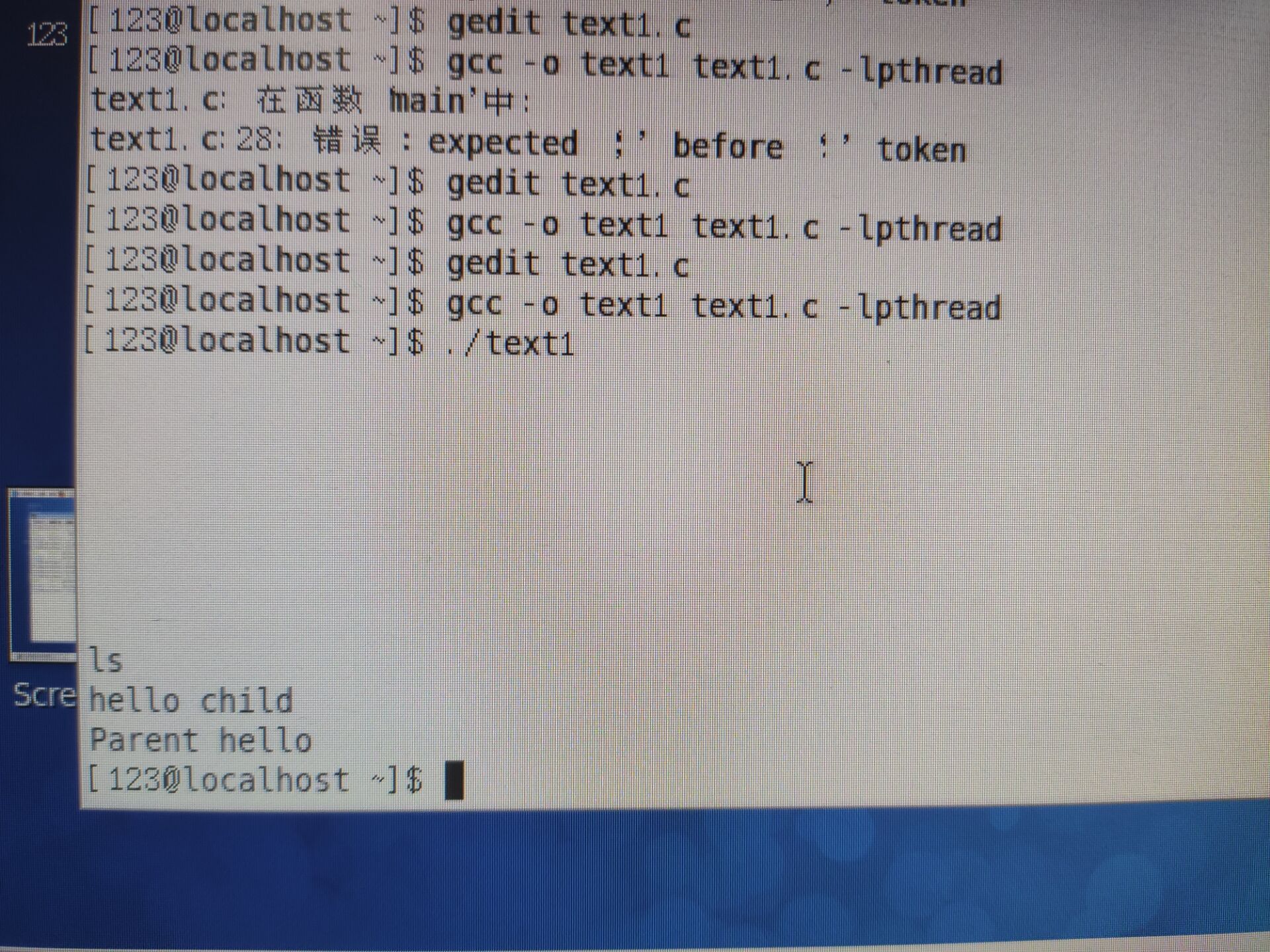
pthread\_join(thread2,NULL);

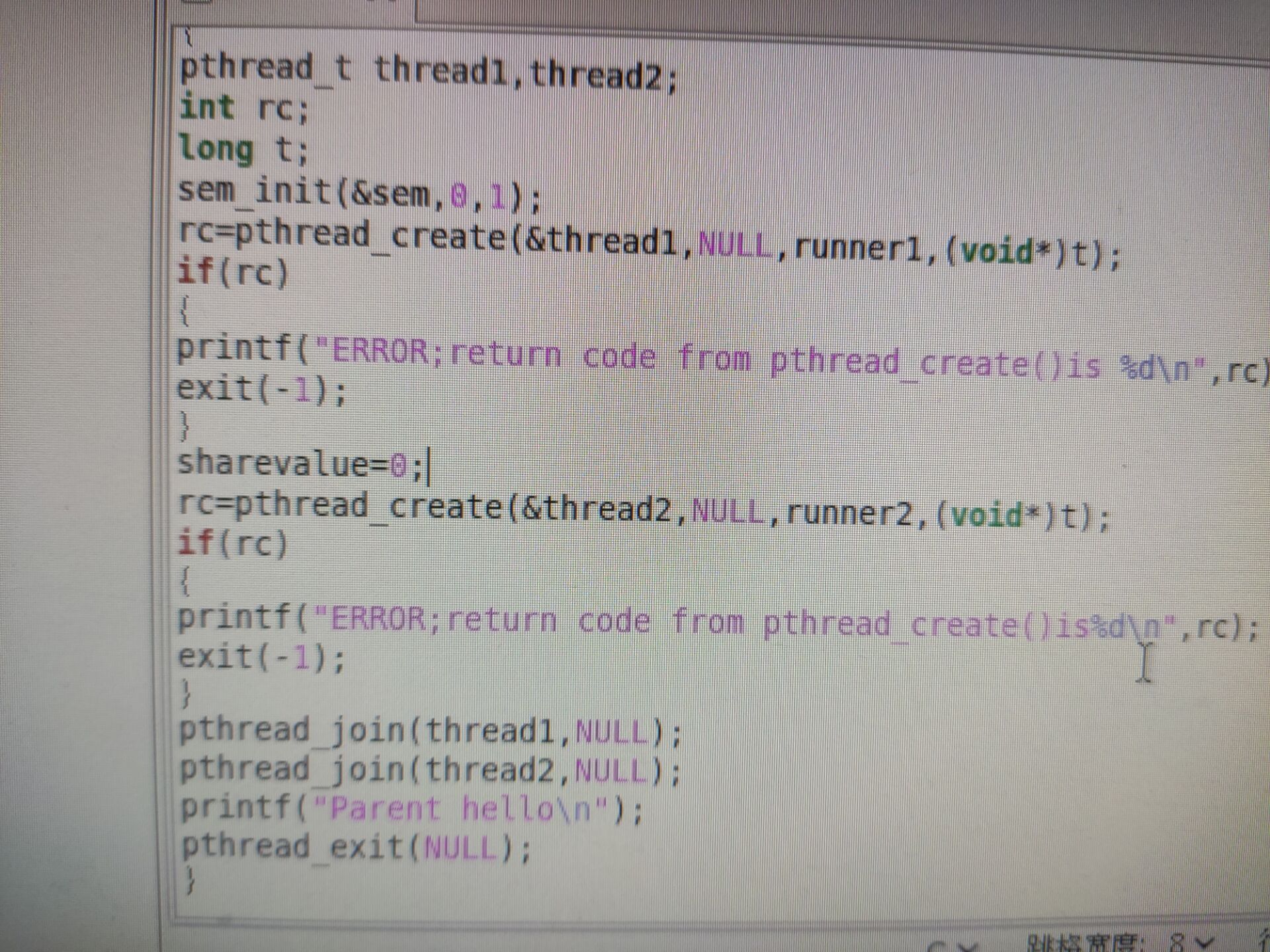
printf("Parent hello\n");

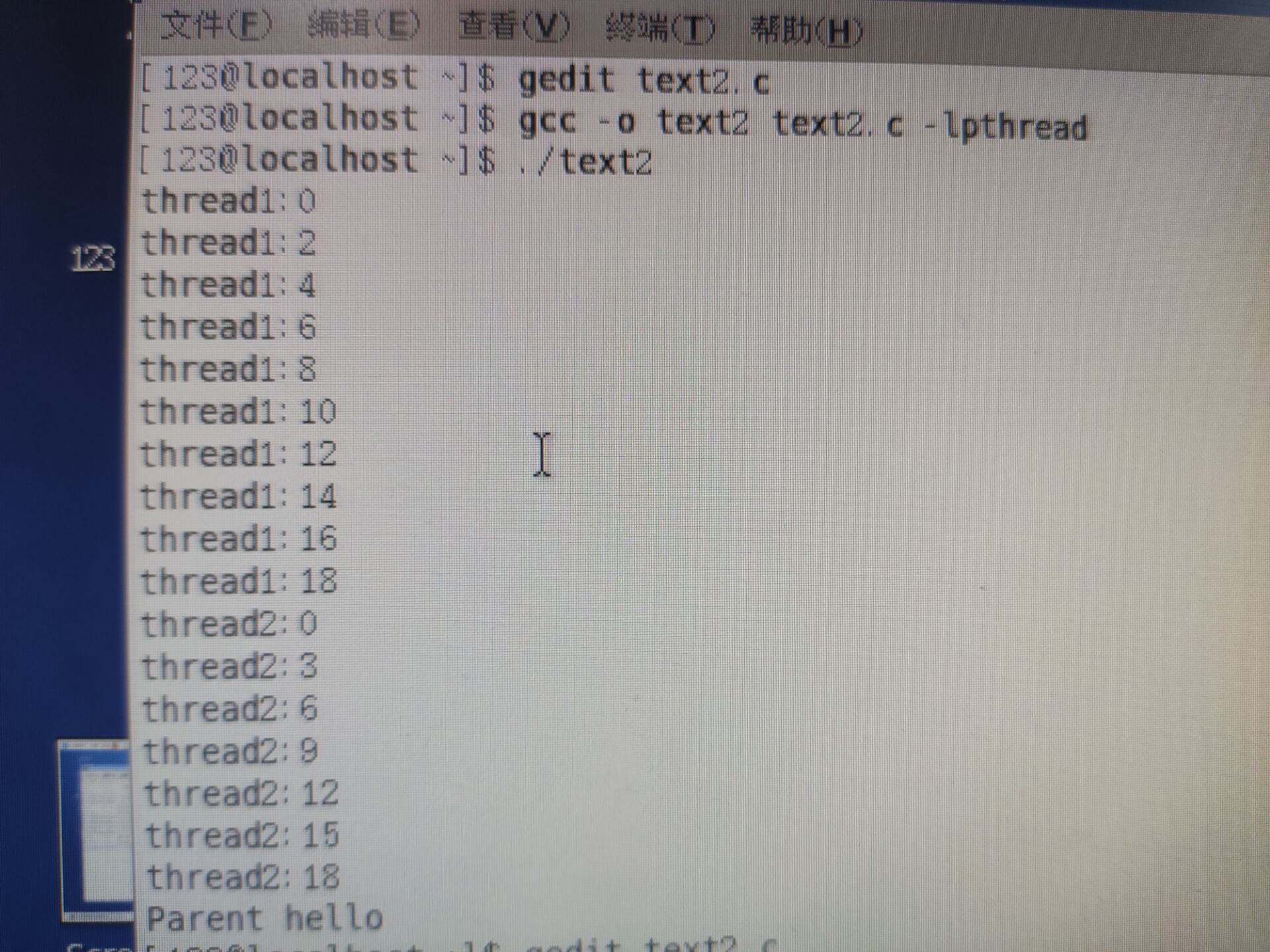
pthread\_exit(NULL);

}

**3、实验结果**







**实验四 并发系统设计实验**

**1、实验目的**

掌握并发系统设计，掌握操作系统互斥死锁等概念及相关技术。

**2、实验内容**

1. 设计并发处理系统，实现并发处理。

可选题目一：电梯调度系统

原理：电梯调度算法的基本原则就是如果在电梯运行方向上有人要使用电梯则继续往那个方向运动，如果电梯中的人还没有到达目的地则继续向原方向运动。具体而言，如果电梯现在朝上运动，如果当前楼层的上方和下方都有请求，则先响应所有上方的请求，然后才向下响应下方的请求；如果电梯向下运动，则刚好相反。

设计要求：模拟多人在不同楼层同时要求到各自目的地时电梯的响应顺序，要求使用C语言编程，定义合适的数据结构。最后，需要说明设计思想，同时给出能够运行的源程序，并给出对应的程序流程图。

设计提示：可以用一个结构体表示乘电梯的人，其中内容包括人的姓名、起始楼层、目的楼层；建立一个结构体的数组模拟当前所有需要乘电梯的人。把这个结构体数组作为程序的输入。需要线程1监视电梯运行状态，即电梯运行的方向和当前楼层，线程2监视输入信息，包括输入时间，起始楼层和目的楼层，线程3模拟电梯运行。

可选题目二：生产者-消费者问题

原理：生产者线程(或进程)产生数据并填入缓冲区，消费者线程（或进程）将数据取出并显示出来。

设计要求：生产者线程计算当前的时间,把时间、第几次计算时间的序号(循环次 数)和线程ID作为一个消息,把消息放入缓冲区,消费者线程从缓冲区读出一个消息 并显示消息。缓冲区大小为5个,每个生产者线程生产10个消息,每个消费者线程消费 10个消息,即生产和消费分别为20次。至少2个生产者线程和2个消费者线程。

设计提示：生产者及消费者之间需要实现同步和互斥，同时多个生产者之间需要互斥，pthread线程库互斥锁机制和POSI 的信号量机制可以完成这一任务。