**实验四 Linux程序库基础与系统调用和进程间通信与多线程编程**

1. **实验要求：**

**1、问题一：静态库和动态库的生成、开发、使用**

1. 程序库基础，静态库与动态库的生成，开发与使用
2. Linux系统调用

**2、问题二：进程间通信**

1. 使用共享内存进行进程间通信
2. 在进程中创建新线程、在一个进程中同步线程之间的数据访问、修改线程属性、在同一个进程中，从一个线程中控制另一个线程

**二、实验内容：**

**1、问题1**

**（1）查阅资料，了解静态库和动态库的生成、开发、使用。并完成以下实验**

**查阅资料：**

1. 编译库文件：

**静态库：**使用编译器（如GCC）将源代码编译成目标文件（.o文件），然后使用静态库管理工具（如ar）将这些目标文件打包成静态库文件（.a文件）。

示例命令：gcc -c file1.c file2.c # 编译源文件为目标文件

ar rcs libexample.a file1.o file2.o # 将目标文件打包成静态库

**动态库：**使用编译器将源代码编译成位置无关的目标文件（.o文件），然后将这些目标文件链接成动态库文件（.so文件）。

示例命令：gcc -fPIC -c file1.c file2.c # 使用-fPIC选项编译为位置无关的目标文件

gcc -shared -o libexample.so file1.o file2.o # 将目标文件链接成动态库

1. 使用库文件：

**静态库：**在开发过程中，可以将静态库作为编译器的输入文件，直接编译和链接源代码。或者，可以在Makefile或构建脚本中指定静态库的路径和名称。

示例命令：gcc -o program program.c libexample.a # 编译并链接程序与静态库

**动态库：**在开发过程中，可以使用动态库的头文件进行编译，并在链接时指定动态库的路径和名称。也可以使用环境变量来设置动态库的搜索路径。

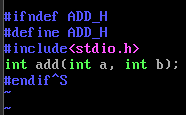
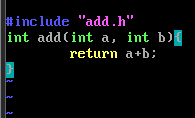
示例命令：gcc -o program program.c -L/path/to/lib -lexample # 编译并链接程序与动态库

**制作静态库：**

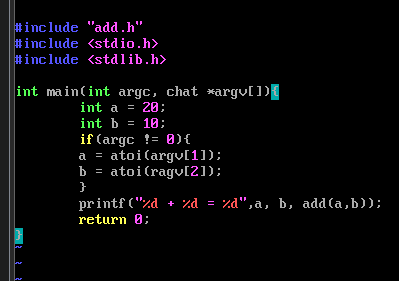
1. 分别编写add.h add.c main.c
2. add.h包含加法函数的定义 int add(int p1, intp2)
3. add.c 包含加法函数的实现

add.h，add.c，main.c中的代码如下：

add.h: add.c:

= 

main.c:



1. 对add.c进行编译生成目标文件add.o

命令：gcc -c file1.c file2.c # 编译源文件为目标文件



1. 执行ar命令，生成libadd.a

命令：ar rcs libexample.a file1.o file2.o # 将目标文件打包成静态库



1. 编写main.c文件，通过include add.h头文件和gcc编译实现对libadd.a中的加法函数的调用

gcc -o <output\_file> <input\_files> #将C代码编译成可执行文件。



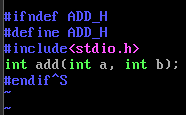
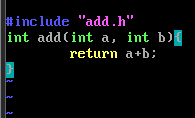
运行main.c:



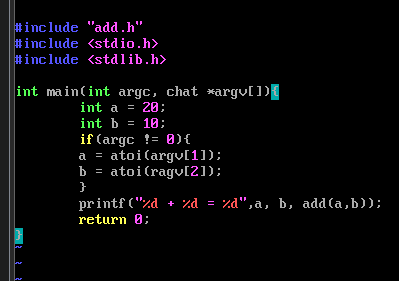
**制作动态库：**

1. 分别编写add.h add.c main.c
2. add.h包含加法函数的定义 int add(int p1, intp2)
3. add.c 包含加法函数的实现

add.h: add.c:

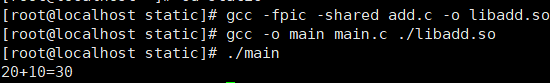
= 

main.c:



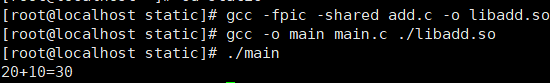
1. 对add.c进行编译生成共享库libadd.so

gcc -fPIC -c file1.c file2.c # 使用-fPIC选项编译为位置无关的目标文件。

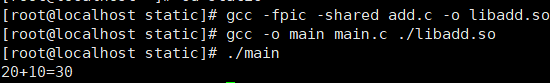


1. 编写main.c文件，通过include add.h头文件和gcc编译实现对libadd.so中的加法函数的调用

gcc -o <output\_file> <input\_files> #将C代码编译成可执行文件。



运行main.c:



1. **查阅相关资料，了解Linux系统调用函数，并完成以下实验**

**查阅资料：**

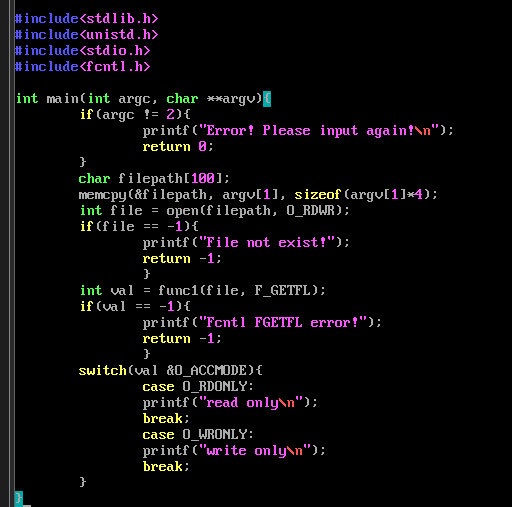
Linux系统调用函数是操作系统提供的一组接口，用于应用程序与内核进行交互。通过系统调用函数，应用程序可以请求操作系统执行特定的功能或访问底层系统资源。下面是一些常见的Linux系统调用函数及其功能的简要介绍：

* fork()：创建一个新的进程，成为父进程的子进程。
* exec()：用于在当前进程中执行一个新的程序。
* wait()和waitpid()：用于等待子进程的终止并获取其终止状态。
* exit()：终止当前进程。
* open()和close()：用于打开和关闭文件。
* read()和write()：用于从文件中读取数据和将数据写入文件。
* pipe()：创建一个管道，用于进程间的通信。
* kill()：向指定进程发送信号。
* signal()和sigaction()：用于处理信号，注册信号处理函数。
* socket()、bind()、listen()和accept()：用于创建和管理网络套接字。
* connect()：建立与远程主机的连接。
* send()和recv()：用于在网络套接字之间发送和接收数据。
* mmap()和munmap()：用于在进程的地址空间中映射和取消映射内存。
* fork()和clone()：创建新的进程或线程。
* ioctl()：用于设备IO控制。

**完成实验：**

* 检测文件当前的读写权限，如果文件具有读权限，则打印可读信息，如果有可写权限，则打印可写信息。否则返回错误信息。
* 如果打开文件为只读文件，则输出”read only”；
* 如果文件是只写文件，则输出”write only”；
* 如果文件可读可写，则输出”read write”；否则输出”unknown mode”
* 提示：可用fcntl()函数实现。

编写程序如下：



编译并运行如下：



**2、问题2**

**（1）查阅相关资料，了解Linux进程间通信。并完成以下实验**

**查阅资料：**

Linux提供了多种进程间通信（Inter-Process Communication，IPC）机制，用于不同进程之间的数据交换和协作。下面是一些常见的Linux进程间通信机制的简要介绍：

**管道（Pipe）：**管道是一种半双工的通信机制，用于在父子进程或具有共同祖先的进程之间传递数据。它可以分为匿名管道（使用pipe()系统调用）和命名管道（使用mkfifo()系统调用）。

**信号（Signal）：**信号是一种异步通信机制，用于在进程之间传递简短的消息。通过发送信号，一个进程可以通知另一个进程发生了某个事件。常见的信号处理函数包括signal()和sigaction()。

**共享内存（Shared Memory）：**共享内存是一种高效的进程间通信机制，允许多个进程共享同一块内存区域。通过映射共享内存区域到各个进程的地址空间，进程可以直接读写共享数据，而无需进行数据拷贝。

**编写程序，实现父子进程通过共享内存进行数据通信。**

1. 创建子进程；
2. 将运行参数argv[1]字符串写入共享内存；
3. 并打印写入字符串;
4. 在父进程中读出所写入的字符串并打印。

代码如下（比较长，直接粘贴代码）：

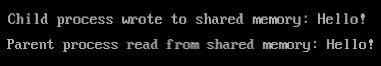
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <sys/types.h>
5. #include <sys/ipc.h>
6. #include <sys/shm.h>
7. #include <string.h>
9. #define SHM\_SIZE 128  // 共享内存大小
11. **int** main(**int** argc, **char** \*argv[]) {
12. **int** shmid;
13. **char** \*shmaddr;
14. pid\_t pid;
16. // 创建共享内存
17. shmid = shmget(IPC\_PRIVATE, SHM\_SIZE, IPC\_CREAT | 0666);
18. **if** (shmid == -1) {
19. perror("shmget");
20. exit(1);
21. }
23. // 创建子进程
24. pid = fork();
25. **if** (pid < 0) {
26. perror("fork");
27. exit(1);
28. }
30. **if** (pid == 0) {
31. // 子进程
32. shmaddr = shmat(shmid, NULL, 0);
33. **if** (shmaddr == (**char** \*)-1) {
34. perror("shmat");
35. exit(1);
36. }
38. // 将参数字符串写入共享内存
39. strncpy(shmaddr, argv[1], SHM\_SIZE);
40. printf("Child process wrote to shared memory: %s\n", shmaddr);
42. // 分离共享内存
43. **if** (shmdt(shmaddr) == -1) {
44. perror("shmdt");
45. exit(1);
46. }
47. } **else** {
48. // 父进程
49. wait(NULL);  // 等待子进程结束
51. shmaddr = shmat(shmid, NULL, 0);
52. **if** (shmaddr == (**char** \*)-1) {
53. perror("shmat");
54. exit(1);
55. }
57. // 从共享内存读取数据并打印
58. printf("Parent process read from shared memory: %s\n", shmaddr);
60. // 分离并删除共享内存
61. **if** (shmdt(shmaddr) == -1) {
62. perror("shmdt");
63. exit(1);
64. }
65. **if** (shmctl(shmid, IPC\_RMID, NULL) == -1) {
66. perror("shmctl");
67. exit(1);
68. }
69. }
71. **return** 0;
72. }

编译运行：

1688392312001

1688392441874

运行结果：



1. 最后练习memcpy和memmove内存段数据处理函数的使用

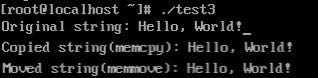
代码：

1. #include <stdio.h>
2. #include <string.h>
4. **int** main() {
5. **char** str1[20] = "Hello, World!";
6. **char** str2[20];
7. **char** str3[20];
9. // 使用memcpy复制字符串
10. memcpy(str2, str1, **sizeof**(str1));
12. // 使用memmove移动字符串
13. memmove(str3, str1, **sizeof**(str1));
15. printf("Original string: %s\n", str1);
16. printf("Copied string (memcpy): %s\n", str2);
17. printf("Moved string (memmove): %s\n", str3);
19. **return** 0;
20. }

编译运行：

1688393165315

运行结果：



**（2）查阅相关资料，了解Linux多线程编程，并完成以下实验**

**查阅资料：**

Linux多线程编程是在Linux操作系统上使用多线程技术来实现并发执行的编程方式。在多线程编程中，一个进程可以包含多个线程，每个线程可以独立执行不同的任务，共享进程的资源和地址空间。这使得多线程编程成为提高程序性能、响应能力和资源利用率的有效手段。

以下是一些关键概念和要点，介绍Linux多线程编程的基本原理和技术：

**线程创建：**Linux提供了一组函数用于创建线程。其中，最常用的函数是**pthread\_create()**，它允许我们创建一个新的线程，并指定线程要执行的函数。这个函数会返回一个线程标识符，我们可以使用它来操作和管理线程。

**线程同步**：由于多线程共享进程的资源，可能会导致资源竞争和数据不一致的问题。为了避免这些问题，我们需要使用同步机制。Linux提供了多种同步工具，如**互斥锁（Mutex）、信号量（Semaphore）**、条件变量（Condition Variable）等。这些同步工具能够确保线程之间的互斥访问和正确的执行顺序，从而保护共享资源的安全访问。

**线程退出：**线程执行完任务后会自动退出，也可以通过调用**pthread\_exit()**来显式地终止线程的执行。在主线程中，可以通过**pthread\_join()**函数等待其他线程的结束，确保主线程在其他线程执行完毕后再终止。

**线程取消：**Linux允许在任何时间取消一个线程的执行。可以使用pthread\_cancel()函数来请求取消指定线程的执行。被取消的线程需要在适当的时候检查取消请求并终止执行。

**线程局部数据：**在多线程环境下，有时候需要为每个线程分配独立的数据空间。Linux提供了线程局部存储（Thread-Local Storage，TLS）机制，可以用于实现线程私有的数据。

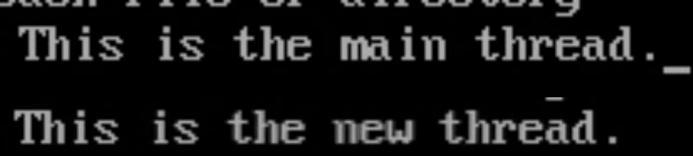
1. **编写程序实现如下功能：**编写程序，在主线程中创建一个新线程。要求在新线程中输出运行信息，在结束时返回主线程。使用函数pthread\_exit（0）退出线程。再试试exit(0)退出线程，看看是什么结果。比较一下二者的不同。

代码：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pthread.h>
5. **void** \*thread\_func(**void** \*arg) {
6. printf("This is the new thread.\n");
8. // 退出线程，分别尝试两种方法，这里将不使用的方法注释掉
9. // pthread\_exit(0);
10. exit(0);
11. }
13. **int** main() {
14. pthread\_t tid;
15. **int** ret;
17. // 创建新线程
18. ret = pthread\_create(&tid, NULL, thread\_func, NULL);
19. **if** (ret != 0) {
20. perror("pthread\_create");
21. exit(1);
22. }
24. // 主线程继续执行
25. printf("This is the main thread.\n");
27. // 等待新线程结束
28. ret = pthread\_join(tid, NULL);
29. **if** (ret != 0) {
30. perror("pthread\_join");
31. exit(1);
32. }
34. printf("New thread has exited.\n");
36. **return** 0;
37. }

运行结果：

**pthread\_exit(0)：**

****

**exit(0):**

**491ea7d17fce8817a3f19d83e574fe9**

**pthread\_exit(0)和exit(0)的区别：pthread\_exit(0)只会终止当前线程，而exit(0)会终止整个进程。在多线程程序中，使用pthread\_exit(0)可以确保只退出当前线程，其他线程可以继续执行。**

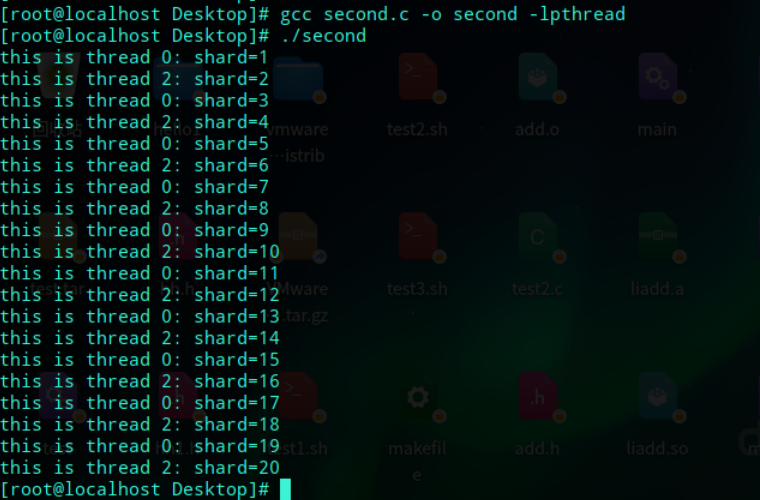
1. **在两个线程之间实现交替执行输出。**要求：例如主线程先输出“this is a thread0”,然后新线程输出“this is thread 2”一直交替直到结束。同时访问全局变量，修改变量的值，并打印看看输出结果。

（没太懂这题的意思），到网上查找资料，发现可以使用使用线程和互斥锁（Mutex）来实现两个线程之间的交替执行和对全局变量的访问和修改。

代码：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pthread.h>
5. **int** shared = 0;  // 全局变量
6. pthread\_mutex\_t mutex;  // 互斥锁
8. **void** \*thread0\_func(**void** \*arg) {
9. **for** (**int** i = 0; i < 5; i++) {
10. pthread\_mutex\_lock(&mutex);  // 上锁
11. shared++;  // 修改全局变量
12. printf("This is thread 0, shared = %d\n", shared);
13. pthread\_mutex\_unlock(&mutex);  // 解锁
14. }
15. pthread\_exit(NULL);
16. }
18. **void** \*thread1\_func(**void** \*arg) {
19. **for** (**int** i = 0; i < 5; i++) {
20. pthread\_mutex\_lock(&mutex);  // 上锁
21. shared++;  // 修改全局变量
22. printf("This is thread 1, shared = %d\n", shared);
23. pthread\_mutex\_unlock(&mutex);  // 解锁
24. }
25. pthread\_exit(NULL);
26. }
28. **int** main() {
29. pthread\_t tid0, tid1;
31. pthread\_mutex\_init(&mutex, NULL);  // 初始化互斥锁
33. pthread\_create(&tid0, NULL, thread0\_func, NULL);
34. pthread\_create(&tid1, NULL, thread1\_func, NULL);
36. pthread\_join(tid0, NULL);
37. pthread\_join(tid1, NULL);
39. pthread\_mutex\_destroy(&mutex);  // 销毁互斥锁
41. **return** 0;
42. }

运行结果：

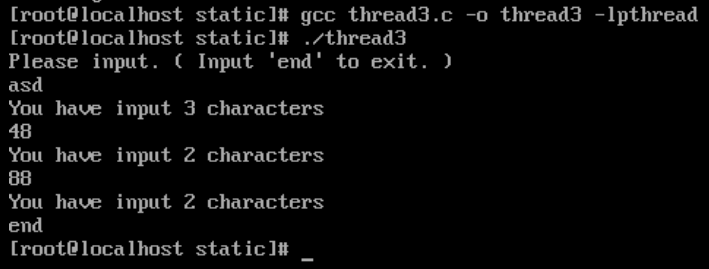


1. **编写程序实现使用信号量进行同步。**编码实现输入字符串，统计每行的字符个数，以“end”结束输入。

代码：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <pthread.h>
5. #include <semaphore.h>
7. #define MAX\_LINES 10
8. #define MAX\_LENGTH 100
10. sem\_t semaphore;
11. **char** input[MAX\_LINES][MAX\_LENGTH];
12. **int** line\_count = 0;
14. **void** \*input\_thread(**void** \*arg) {
15. **char** buffer[MAX\_LENGTH];
17. **while** (1) {
18. fgets(buffer, MAX\_LENGTH, stdin);
19. buffer[strcspn(buffer, "\n")] = '\0';  // 移除换行符
21. sem\_wait(&semaphore);  // 等待信号量
23. **if** (strcmp(buffer, "end") == 0) {
24. sem\_post(&semaphore);  // 释放信号量
25. **break**;
26. }
28. strcpy(input[line\_count], buffer);
29. line\_count++;
31. sem\_post(&semaphore);  // 释放信号量
32. }
34. pthread\_exit(NULL);
35. }
37. **void** \*count\_thread(**void** \*arg) {
38. **while** (1) {
39. sem\_wait(&semaphore);  // 等待信号量
41. **if** (line\_count == 0) {
42. sem\_post(&semaphore);  // 释放信号量
43. **continue**;
44. }
46. **int** current\_line = line\_count - 1;
47. **int** char\_count = strlen(input[current\_line]);
49. printf("You have input %d characters\n", char\_count);
51. sem\_post(&semaphore);  // 释放信号量
53. **if** (strcmp(input[current\_line], "end") == 0) {
54. **break**;
55. }
56. }
58. pthread\_exit(NULL);
59. }
61. **int** main() {
62. pthread\_t tid\_input, tid\_count;
64. sem\_init(&semaphore, 0, 1);  // 初始化信号量
66. pthread\_create(&tid\_input, NULL, input\_thread, NULL);
67. pthread\_create(&tid\_count, NULL, count\_thread, NULL);
69. pthread\_join(tid\_input, NULL);
70. pthread\_join(tid\_count, NULL);
72. sem\_destroy(&semaphore);  // 销毁信号量
74. **return** 0;
75. }

运行结果：



**三、实验分析与总结：**

结合理论课学到的知识通过这次实验，我对系统调用、进程与线程的知识有了更加直观的认识。但是感觉本次实验的难度比之前高了不少，主要难在编程上。有很多题目我刚开始全无头绪，到网络上查找资料和寻找帮助后才逐渐理解题意。比如实现交替输出的那题，感觉题目没有说清楚用什么算法，我在网上查阅资料才发现可以使用锁来解决。再比如一些代码编译一直报错，在网上查询之后，知道了编译时需要添加-lpthread参数，因为pthread不是Linux系统默认的库，连接时需要使用静态库libthread.a。在不断克服困难的过程中，我学到了更多更加实用的知识。

本学期的操作系统课程至此告一段落，我在课程中学到了很多，谢谢老师和助教的教导！