**操作系统实验报告**

班级： 22计科3班 姓名：张雅瑞 学号：2022334323029

**实验一：进程通信（一）——管道及共享内存**

1. 实验目的

1.理解和掌握进程通信的基本概念和方法。

2.学习管道通信的使用。

3.掌握共享内存的基本操作。

1. 实验内容和步骤

### 任务一

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  main() {  int x, fd[2];  char buf[30], s[30];  pipe(fd);  while ((x = fork()) == -1);  if (x == 0) {  close(fd[0]);  printf("Child Process!\n");  strcpy(buf, "This is an example\n");  write(fd[1], buf, 30);  exit(0);  } else {  close(fd[1]);  printf("Parent Process!\n");  read(fd[0], s, 30);  printf("%s\n", s);  }  } |

（1）阅读以上父子进程利用管道进行通信的例子，写出程序的运行结果并分析。

（2）编写程序：父进程利用管道将一字符串交给子进程处理。子进程读字符串，将里面的字符反向后再交给父进程，父进程最后读取并打印反向的字符串。

### 任务二

例2

|  |
| --- |
| #include <sys/ipc.h>  #include <sys/shm.h>  #include <sys/types.h>  main() {  key\_t key = 15;  int shmid\_1, shmid\_2;  if ((shmid\_1 = shmget(key, 1000, 0644 | IPC\_CREAT)) == -1) {  perror("shmget shmid\_1");  exit(1);  }  printf("First shared memory identifier is %d\n", shmid\_1);  if ((shmid\_2 = shmget(IPC\_PRIVATE, 20, 0644)) == -1) {  perror("shmget shmid\_2");  exit(2);  }  printf("Second shared memory identifier is %d\n", shmid\_2);  exit(0);  } |

1. 阅读例2的程序，运行一次该程序，然后用ipcs命令查看系统中共享存储区的情况，再次执行该程序，再用ipcs命令查看系统中共享内存的情况，对两次的结果进行比较，并分析原因。最后用ipcrm命令删除自己建立的共享存储区。

例3 程序1

|  |
| --- |
| #include <sys/ipc.h>  #include <sys/shm.h>  #include <sys/types.h>  #define SHMKEY 75  #define K 1024  int shmid;  main() {  int i, \*pint;  char \*addr;  extern char \*shmat();  extern cleanup();  for (i = 0; i < 20; i++) signal(i, cleanup);  shmid = shmget(SHMKEY, 16 \* K, 0777 | IPC\_CREAT); /\*建立16K共享区SHMKEY \*/  addr = shmat(shmid, 0, 0); /\*挂接，并得到共享区首地址 \*/  printf("addr 0x%x\n", addr);  pint = (int \*)addr;  for (i = 0; i < 256; i++) \*pint++ = i;  pause(); /\*等待接收进程读 \*/  }  cleanup() {  shmctl(shmid, IPC\_RMID, 0);  exit();  } |

例3 程序2

|  |
| --- |
| #include <sys/ipc.h>  #include <sys/shm.h>  #include <sys/types.h>  #define SHMKEY 75  #define K 1024  int shmid;  main() {  int i, \*pint;  char \*addr;  extern char \*shmat();  shmid = shmget(SHMKEY, 8 \* K, 0777); /\*取共享区SHMKEY的id \*/  addr = shmat(shmid, 0, 0); /\*连接共享区\*/  pint = (int \*)addr;  for (i = 0; i < 256; i++) printf("%d\n", \*pint++); /\*打印共享区中 的内容\*/  } |

（2）每个同学登陆两个窗口，先在一个窗口中运行例3程序1（或者只登陆一个窗口，先在该窗口中以后台方式运行程序1），然后在另一个窗口中运行例3程序2，观察程序的运行结果并分析。运行结束后可以用ctrl+c结束程序1的运行。

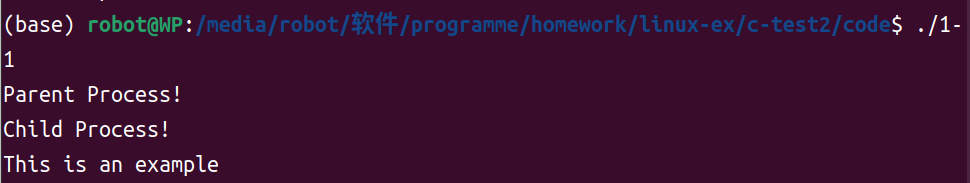
（3）编写程序：使用系统调用shmget()，shmat()，shmdt()，shmctl()，编制程序。要求在父进程中生成一个30字节长的私有共享内存段。接下来，设置一个指向共享内存段的字符指针，将一串大写字母写入到该指针指向的存贮区。调用fork()生成子进程，让子进程显示共享内存段中的内容。接着，将大写字母改成小写，子进程修改共享内存中的内容。之后，子进程将脱接共享内存段并退出。父进程在睡眠5秒后，在此显示共享内存段中的内容（此时已经是小写字母）。

1. 代码及运行结果分析

### 任务一

**（1）阅读以上父子进程利用管道进行通信的例子，写出程序的运行结果并分析。**

运行结果：

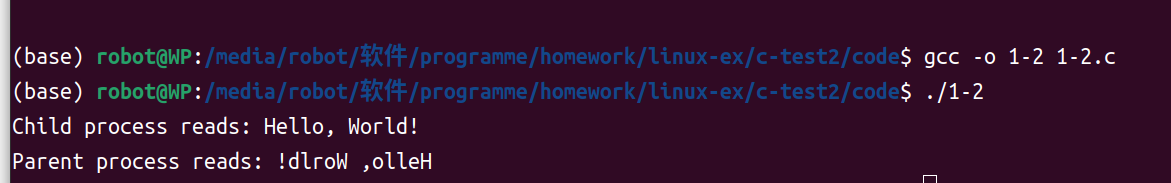
分析：

这个程序创建了一个管道，并通过 `fork()` 创建子进程。子进程向管道写入一段字符串，父进程从管道读取该字符串并打印出来，实现了父子进程间的简单通信。

**（2）编写程序：父进程利用管道将一字符串交给子进程处理。子进程读字符串，将里面的字符反向后再交给父进程，父进程最后读取并打印反向的字符串。**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <sys/types.h>  void reverseString(char \*str) {  int n = strlen(str);  for (int i = 0; i < n / 2; i++) {  char temp = str[i];  str[i] = str[n - i - 1];  str[n - i - 1] = temp;  }  }  int main() {  int fd1[2], fd2[2];  char buf[100];  if (pipe(fd1) == -1 || pipe(fd2) == -1) {  perror("pipe");  exit(1);  }  pid\_t pid = fork();  if (pid < 0) {  perror("fork");  exit(1);  } else if (pid == 0) {  // Child process  close(fd1[1]);  close(fd2[0]);  read(fd1[0], buf, sizeof(buf));  printf("Child process reads: %s\n", buf);  reverseString(buf);  write(fd2[1], buf, strlen(buf) + 1);  close(fd1[0]);  close(fd2[1]);  exit(0);  } else {  // Parent process  close(fd1[0]);  close(fd2[1]);  char inputStr[] = "Hello, World!";  write(fd1[1], inputStr, strlen(inputStr) + 1);  read(fd2[0], buf, sizeof(buf));  printf("Parent process reads: %s\n", buf);  close(fd1[1]);  close(fd2[0]);  }  return 0;  } |

运行结果：

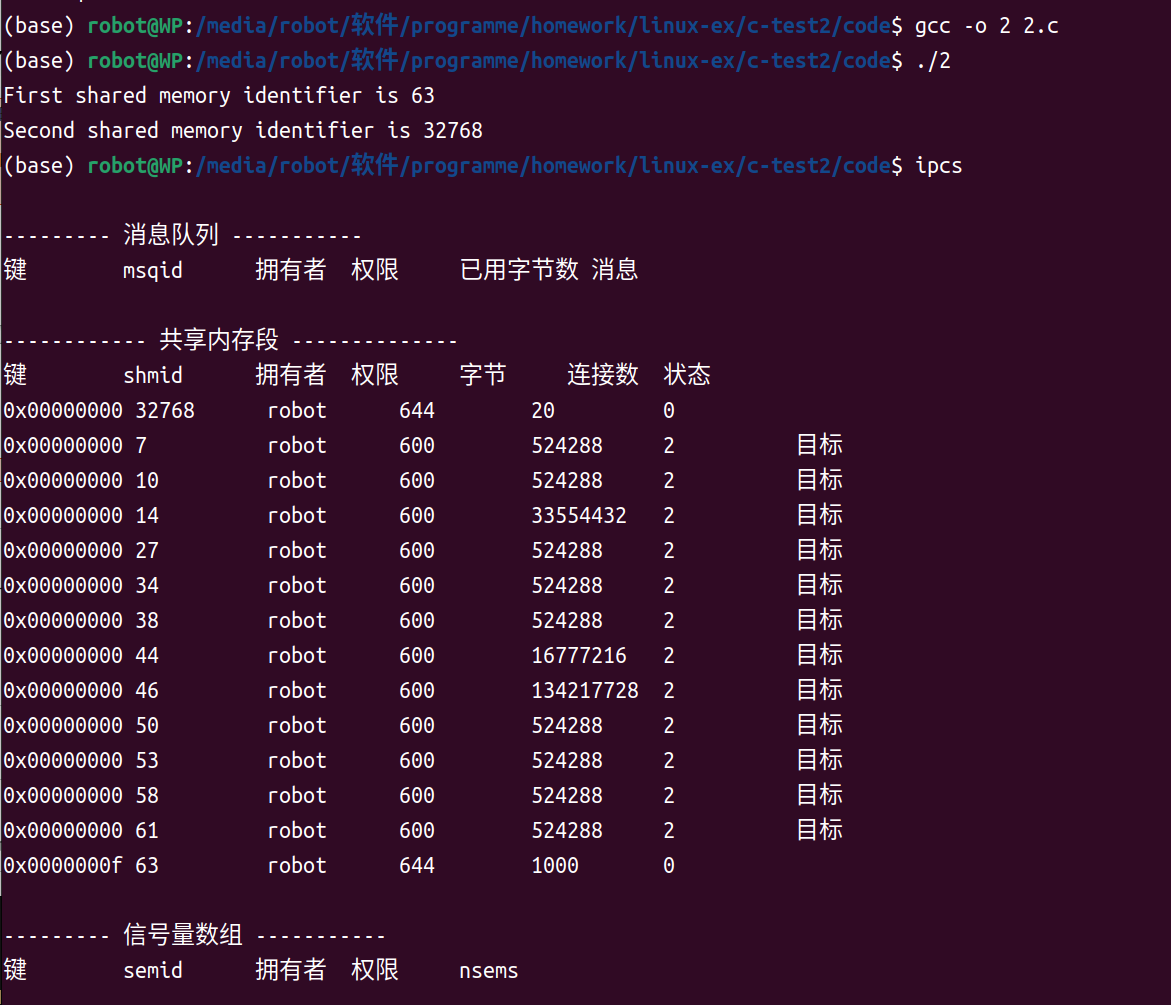
分析：

这个程序创建了两个管道，并通过 `fork()` 创建子进程。父进程向第一个管道写入字符串，子进程从第一个管道读取字符串并将其反转，然后将反转后的字符串写入第二个管道。父进程从第二个管道读取反转后的字符串并打印出来，实现了父子进程间的双向通信。

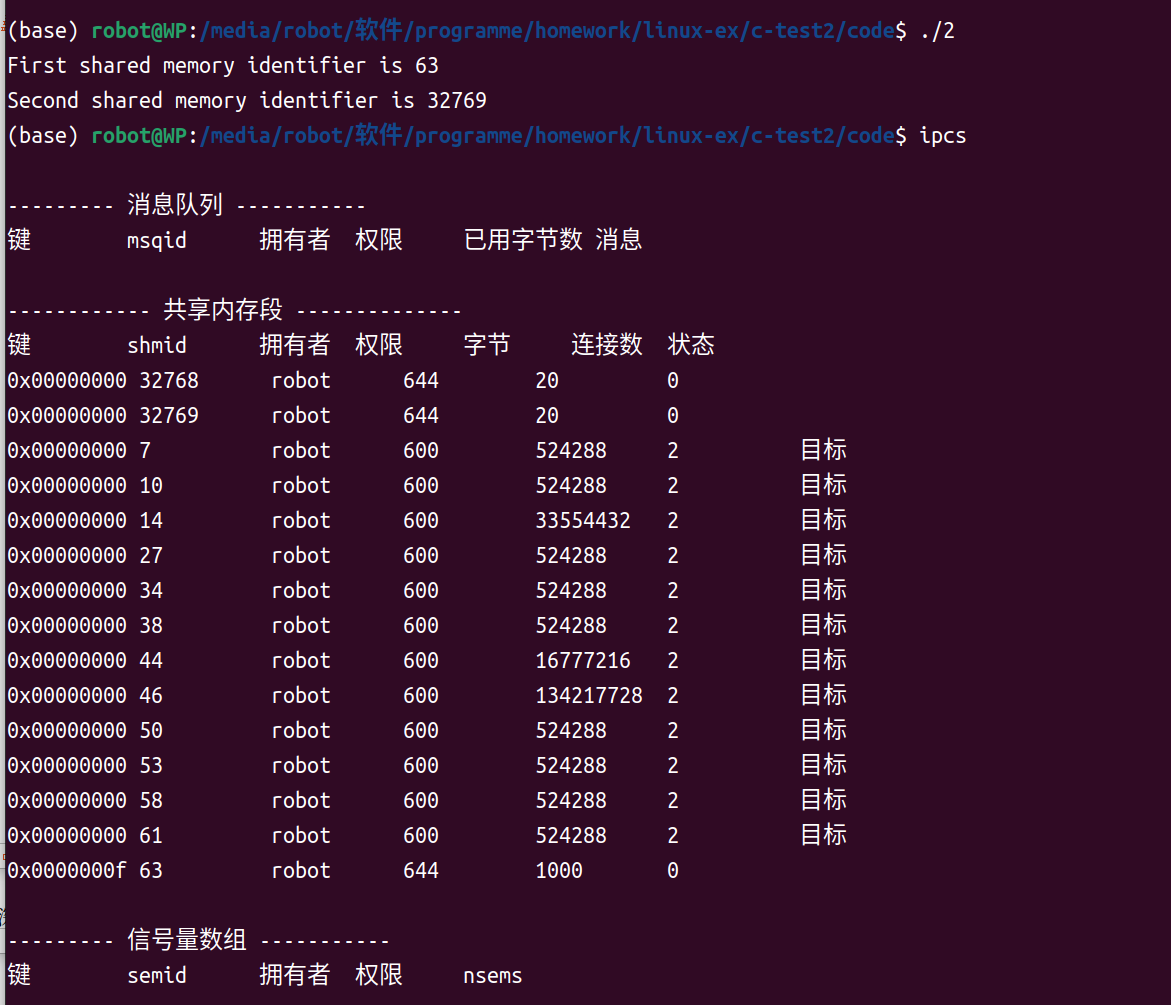
### 任务二

1. **阅读例2的程序，运行一次该程序，然后用ipcs命令查看系统中共享存储区的情况，再次执行该程序，再用ipcs命令查看系统中共享内存的情况，对两次的结果进行比较，并分析原因。最后用ipcrm命令删除自己建立的共享存储区。**

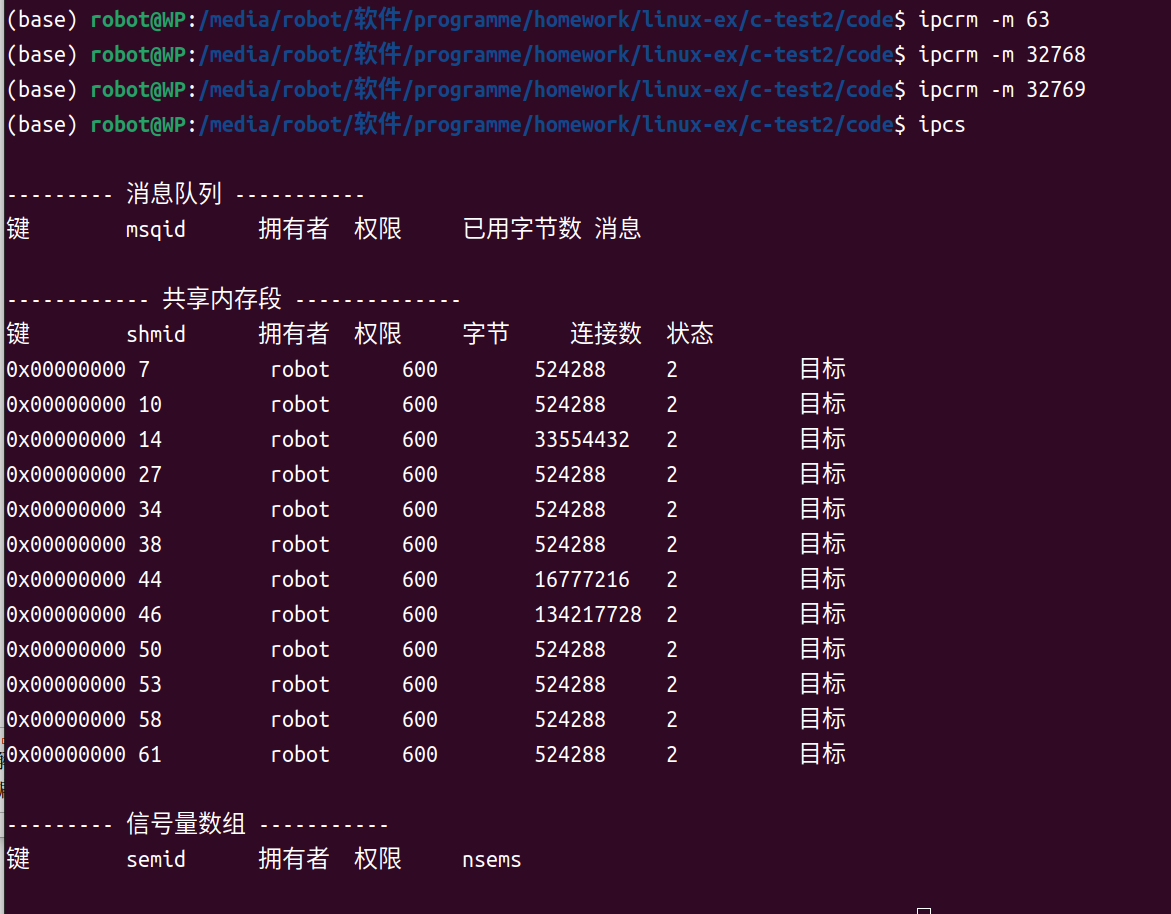
第一次运行：



**第二次运行：**

****

**删除共享区：**

分析：

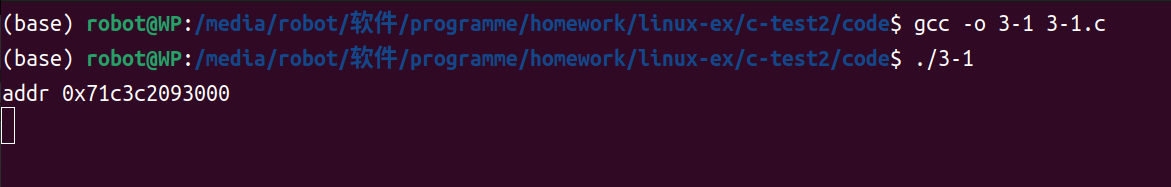
第一次执行程序:一个键为15的共享内存段和一个键为IPC\_PRIVATE（显示为0）的共享内存段。

第二次执行程序:键为15的共享内存段仍然存在，而使用IPC\_PRIVATE会创建一个新的段。

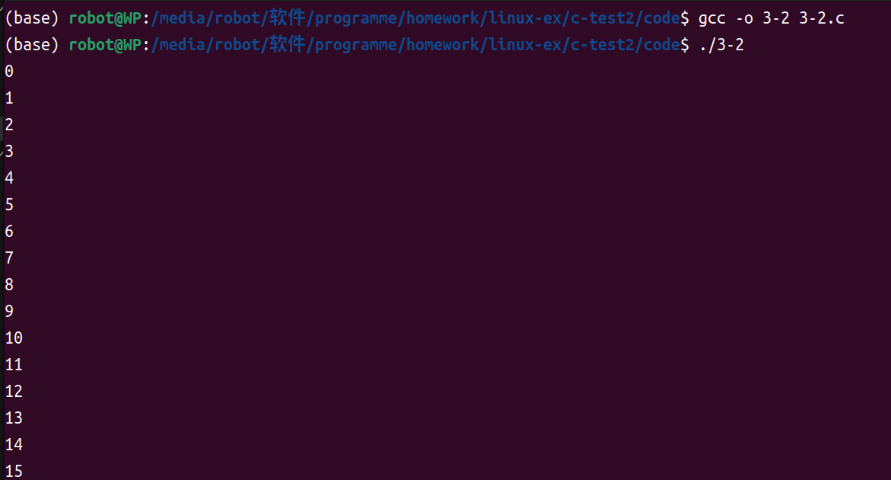
键为15的共享内存段在两次运行之间保持不变，使用IPC\_PRIVATE创建的共享内存段每次执行都不同。

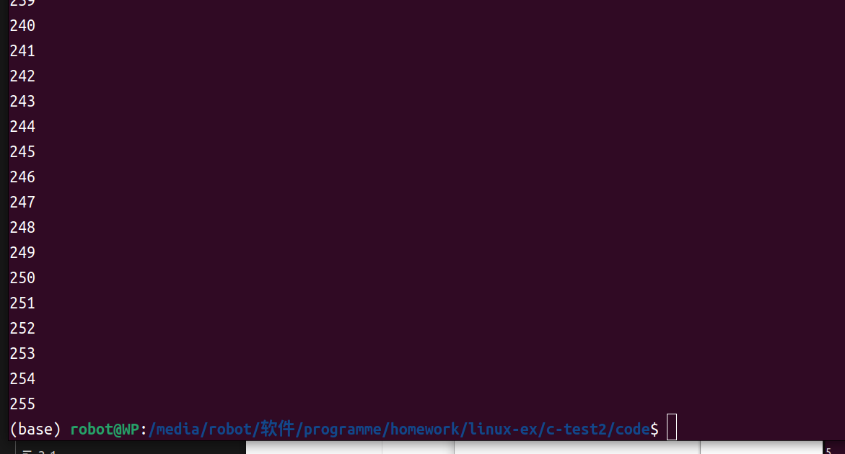
**（2）每个同学登陆两个窗口，先在一个窗口中运行例3程序1（或者只登陆一个窗口，先在该窗口中以后台方式运行程序1），然后在另一个窗口中运行例3程序2，观察程序的运行结果并分析。运行结束后可以用ctrl+c结束程序1的运行。**

运行程序1



运行程序2





分析：

程序1——创建并初始化共享内存:

- 创建一个16K的共享内存区，并将其初始化为从0到255的整数。

- 使用 `signal` 函数注册信号处理程序 `cleanup`，在接收到信号时删除共享内存。

- 使用 `pause` 函数等待信号，以便其他进程可以读取共享内存中的数据。

程序2——连接并读取共享内存:

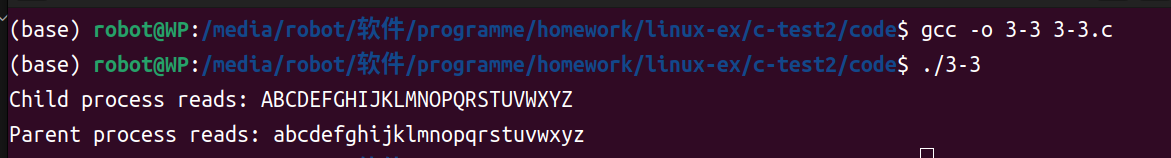
- 连接到一个已存在的8K共享内存区。

- 读取共享内存中的数据，并打印前256个整数。

**（3）编写程序：使用系统调用shmget()，shmat()，shmdt()，shmctl()，编制程序。要求在父进程中生成一个30字节长的私有共享内存段。接下来，设置一个指向共享内存段的字符指针，将一串大写字母写入到该指针指向的存贮区。调用fork()生成子进程，让子进程显示共享内存段中的内容。接着，将大写字母改成小写，子进程修改共享内存中的内容。之后，子进程将脱接共享内存段并退出。父进程在睡眠5秒后，在此显示共享内存段中的内容（此时已经是小写字母）。**

|  |
| --- |
| #include <ctype.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <unistd.h>  #include <sys/wait.h>  #define SHM\_SIZE 30  #define SHM\_KEY 1234  void handle\_error(const char \*msg) {  perror(msg);  exit(1);  }  int main() {  int shmid;  char \*shm;  // 创建共享内存  if ((shmid = shmget(SHM\_KEY, SHM\_SIZE, IPC\_CREAT | 0666)) < 0) {  handle\_error("shmget");  }  // 连接到地址空间  if ((shm = shmat(shmid, NULL, 0)) == (char \*)-1) {  handle\_error("shmat");  }  strcpy(shm, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");  pid\_t pid = fork();  if (pid < 0) {  handle\_error("fork");  } else if (pid == 0) {  // 子进程  printf("Child process reads: %s\n", shm);  for (int i = 0; shm[i] != '\0'; i++) {  shm[i] = tolower(shm[i]);  }  // 脱接  if (shmdt(shm) < 0) {  handle\_error("shmdt");  }  exit(0);  } else {  // 父进程  wait(NULL);  sleep(5);  printf("Parent process reads: %s\n", shm);  // 脱接  if (shmdt(shm) < 0) {  handle\_error("shmdt");  }  // 删除  if (shmctl(shmid, IPC\_RMID, NULL) < 0) {  handle\_error("shmctl");  }  }  return 0;  } |

运行结果：



1. 心得体会

通过这次实验，我对进程通信的两种主要方式——管道通信和共享内存通信——有了更深刻的理解和实践经验。管道通信部分让我掌握了进程间如何通过管道进行数据传输，理解了管道的创建、读写和关闭等操作，并通过编写程序加深了对这一过程的认识。共享内存通信部分则让我学会了如何创建和操作共享内存段，并通过实际编程实践理解了共享内存的同步和通信机制。