操作系统实验报告

班级：计科 4 班 姓名：陈昊天 学号：2021329600006

实验二：进程通信（一）——管道及共享内存

一、 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。

二、 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）。

三、 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:

```
Parent Process!
Child Process!
This is an example
```

分析:

子进程的执行:子进程关闭了管道的读端（fd[0]），打印出“Child Process!”以标识自己是子进程。将字符串“This is an example\n”复制到缓冲区 buf 中，将 buf 的内容写入管道的写端（fd[1]）。

父进程的执行:父进程关闭了管道的写端（fd[1]），打印出“Parent Process!”以标识自己是父进程。

从管道的读端（fd[0]）读取数据到缓冲区 s 中，打印出从管道读取的字符串。

（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>
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];  // fd1 父进程写子进程读，fd2 子进程写父进程读
    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) {
        // 子进程
        close(fd1[1]);  // 0 读 1 写
        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 {
        // 父进程
        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:

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./"1-2"
Child process reads: Hello, World!
Parent process reads: !dlroW ,olleH
```

分析:

子进程逻辑:

关闭 fd1 的写端和 fd2 的读端。

从 fd1[0] 读取字符串，调用 reverseString 函数将其翻转。

将翻转后的字符串写入 fd2[1]。

关闭剩余的管道端口并退出。

父进程逻辑：

关闭 fd1 的读端和 fd2 的写端。

向 fd1[1] 写入一个字符串（"Hello, World!"）。

从 fd2[0] 读取翻转后的字符串。

打印出翻转后的字符串。

关闭剩余的管道端口。

字符串翻转函数：reverseString 函数通过交换字符的位置来反转字符串。

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：

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./"2"
First shared memory identifier is 65536
Second shared memory identifier is 65537
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcs
IPC status from <running system> as of Fri Dec 15 14:20:07 CST 2023
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m  65536 0x0000000f --rw-r--r-- nanmener    staff
m  65537 0x00000000 --rw-r--r-- nanmener    staff
```

第二次运行：

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./"2"
First shared memory identifier is 65536
Second shared memory identifier is 65538
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcs
IPC status from <running system> as of Fri Dec 15 14:20:29 CST 2023
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m  65536 0x0000000f --rw-r--r-- nanmener    staff
m  65537 0x00000000 --rw-r--r-- nanmener    staff
m  65538 0x00000000 --rw-r--r-- nanmener    staff
```

删除共享存储区：

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcrm -m 65536
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcrm -m 65537
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcrm -m 65538
(base) nanmener@Haotians-MacBook-Pro 实验2 % ipcs
IPC status from <running system> as of Fri Dec 15 14:27:07 CST 2023
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
```

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

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./3-1
addr 0x1009a4000
```

运行程序 2

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./3-2
0
1
2
3
4
5
```

```
248
249
250
251
252
253
254
255
(base) nanmener@Haotians-MacBook-Pro 实验2 %
```

分析：
程序 1：
SHMKEY 是共享内存的键，K 是以字节为单位的大小常量，shmid 存储共享内存段的标识符。
为前 20 个信号设置了一个清理函数 cleanup, 使用 shmget 创建或访问一个 16K 大小的共享内存段，通过 shmat 将共享内存段连接到进程的地址空间，在共享内存中写入 256 个整数。
使用 pause()，等待另一个进程读取数据。

程序 2：
使用 shmget 以相同的键访问共享内存，连接共享内存，从共享内存读取 256 个整数并打印。

（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>
#define SHM_SIZE 30
int main() {
    int shmid;
    char *shm;
    key_t key = 1234;
    if ((shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *)-1) {
        perror("shmat");
        exit(1);
    }
    strcpy(shm, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
```

```
        exit(1);
    } 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]);
        }
        if (shmdt(shm) < 0) {
            perror("shmdt");
            exit(1);
        }
        exit(0);
    } else {
        // 父进程
        wait(NULL);
        sleep(5);
        printf("Parent process reads: %s\n", shm);
        if (shmdt(shm) < 0) {
            perror("shmdt");
            exit(1);
        }
        if (shmctl(shmid, IPC_RMID, NULL) < 0) {
            perror("shmctl");
            exit(1);
        }
    }
    return 0;
}
```

运行结果:

```
(base) nanmener@Haotians-MacBook-Pro 实验2 % ./"3-3"
Child process reads: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Parent process reads: abcdefghijklmnopqrstuvwxyz
```

四、 心得体会

进程通信是操作系统中的重要概念之一，它允许不同的进程之间共享数据和协作执行任务。管道和共享内存是实现进程通信的两种重要方式，它们各自适用于不同的场景和需求。管道是一种在父子进程或者不相关进程之间进行通信的有效方式。通过创建管道，父子进程可以互相传递数据，实现数据交换和协作。在编写管道通信程序时，要注意关闭不需要的管道端口，以避免死锁和资源泄漏。共享内存是一种高效的进程通信方式，它允许多个进程共享同一块内存区域。在使用共享内存时，要谨慎处理进程间的竞争条件，同时要注意正确地连接和脱离

共享内存段，以避免内存泄漏。

在实验中，我们编写了一个父子进程之间通过共享内存进行数据交换的程序。这种技术在实际应用中也有很多用途，如多进程协同处理任务、进程间共享配置信息等。在编写进程通信程序时，要注意错误处理。使用错误处理函数和检查系统调用的返回值可以帮助我们及时发现和解决问题，提高程序的稳定性和可靠性。

本次实验让我更深入地理解了进程通信的概念和方法，并通过编写实际的程序来加深对管道和共享内存的理解。这些知识和经验对我今后在操作系统和并发编程领域的学习和工作都将有所帮助。