南京信息工程大学 实验（实习）报告

实验（实习）名称 进程的通信 实验（实习）日期2022.11.3 得分 指导教师

系 计算机学院专业 物联网工程年级 2021 班次 2 姓名 学号

**一、实验目的**

学习如何利用管道机制、消息缓冲队列、共享存储区进行进程间的通信，并加深对上述通信机制的理解。

**二、实验内容与步骤**

1.了解系统调用pipe()、msgget ()、msgsnd()、msgrcv ()、msgctl ( )、shmget ()、shmat ()、shmdt ()、shmctl () 的功能和实现过程。  
2.编写一段程序，使其用管道来实现父子进程之间的进程通信。子进程向父进程发送自己的进程标识符，以及字符串“is sending a message to parent!”。父进程则通过管道读出子进程发来的消息，将消息显示在屏幕上，然后终止。

3.编写一段程序，使其用消息缓冲队列来实现client进程和server进程之间的通信。

4.编写一段程序，使其用共享存储区来实现两个进程之间的进程通信。

1. **实验步骤**

1、了解系统调用pipe()、msgget ()、msgsnd()、msgrcv ()、msgctl ( )、shmget ()、shmat ()、shmdt ()、shmctl () 的功能和实现过程。

* pipe() 系统调用用于创建一个管道，即一个双向的数据流通道，它可以在具有亲缘关系的进程之间进行通信
* msgget() 系统调用用于创建或访问一个消息队列，即一个存放消息的链表，它可以在不相关的进程之间进行通信。
* msgrcv() 系统调用用于从一个消息队列接收一条消息。
* msgctl() 系统调用用于控制消息队列的属性。
* shmget() 系统调用用于创建或访问一个共享内存，即一段可以被多个进程访问的物理内存。
* shmat() 系统调用用于将一个共享内存连接到当前进程的地址空间，使得当前进程可以访问共享内存中的数据。
* shmdt() 系统调用用于将一个共享内存从当前进程的地址空间分离，使得当前进程不能再访问共享内存中的数据。
* shmctl() 系统调用用于控制共享内存的属性。

2.编写一段程序，使其用管道来实现父子进程之间的进程通信。子进程向父进程发送自己的进程标识符，以及字符串“is sending a message to parent!”。父进程则通过管道读出子进程发来的消息，将消息显示在屏幕上，然后终止。

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

int main()

{

    int fd[2]; *// 创建一个管道，fd[0]是读端，fd[1]是写端*

    pid\_t pid; *// 创建一个进程标识符*

    char buffer[100]; *// 创建一个缓冲区，用于存储消息*

    if (pipe(fd) == -1) *// 如果创建管道失败，打印错误信息并退出*

    {

        perror("pipe error");

        return -1;

    }

    pid = fork(); *// 创建一个子进程*

    if (pid == -1) *// 如果创建子进程失败，打印错误信息并退出*

    {

        perror("fork error");

        return -1;

    }

    if (pid == 0) *// 如果是子进程*

    {

        close(fd[0]); *// 关闭读端*

        sprintf(buffer, "%d is sending a message to parent!", getpid()); *// 将子进程的进程标识符和字符串拼接到缓冲区中*

        write(fd[1], buffer, strlen(buffer) + 1); *// 将缓冲区中的消息写入管道的写端*

        close(fd[1]); *// 关闭写端*

    }

    else *// 如果是父进程*

    {

        close(fd[1]); *// 关闭写端*

        read(fd[0], buffer, sizeof(buffer)); *// 从管道的读端读取消息到缓冲区中*

        printf("Parent received: %s\n", buffer); *// 将缓冲区中的消息显示在屏幕上*

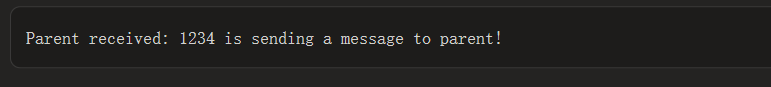
        close(fd[0]); *// 关闭读端*

        wait(NULL); *// 等待子进程结束*

    }

    return 0;

}



3.编写一段程序，使其用消息缓冲队列来实现client进程和server进程之间的通信。

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

int main()

{

    int fd[2]; *// 创建一个管道，fd[0]是读端，fd[1]是写端*

    pid\_t pid; *// 创建一个进程标识符*

    char buffer[100]; *// 创建一个缓冲区，用于存储消息*

    if (pipe(fd) == -1) *// 如果创建管道失败，打印错误信息并退出*

    {

        perror("pipe error");

        return -1;

    }

    pid = fork(); *// 创建一个子进程*

    if (pid == -1) *// 如果创建子进程失败，打印错误信息并退出*

    {

        perror("fork error");

        return -1;

    }

    if (pid == 0) *// 如果是子进程*

    {

        close(fd[0]); *// 关闭读端*

        sprintf(buffer, "%d is sending a message to parent!", getpid()); *// 将子进程的进程标识符和字符串拼接到缓冲区中*

        write(fd[1], buffer, strlen(buffer) + 1); *// 将缓冲区中的消息写入管道的写端*

        close(fd[1]); *// 关闭写端*

    }

    else *// 如果是父进程*

    {

        close(fd[1]); *// 关闭写端*

        read(fd[0], buffer, sizeof(buffer)); *// 从管道的读端读取消息到缓冲区中*

        printf("Parent received: %s\n", buffer); *// 将缓冲区中的消息显示在屏幕上*

        close(fd[0]); *// 关闭读端*

        wait(NULL); *// 等待子进程结束*

    }

    return 0;

}

图形用户界面

中度可信度描述已自动生成

4.编写一段程序，使其用共享存储区来实现两个进程之间的进程通信。

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

#define SHM\_KEY 1234 *// 定义共享存储区的键值*

#define SHM\_SIZE 1024 *// 定义共享存储区的大小*

int main()

{

    int shmid; *// 共享存储区的标识符*

    char \*shmaddr; *// 共享存储区的地址*

    pid\_t pid; *// 进程标识符*

*// 创建或获取一个共享存储区*

    shmid = shmget(SHM\_KEY, SHM\_SIZE, IPC\_CREAT | 0666);

    if (shmid == -1) *// 如果失败，打印错误信息并退出*

    {

        perror("shmget error");

        exit(-1);

    }

*// 创建一个子进程*

    pid = fork();

    if (pid == -1) *// 如果失败，打印错误信息并退出*

    {

        perror("fork error");

        exit(-1);

    }

    if (pid == 0) *// 如果是子进程*

    {

*// 将共享存储区连接到子进程的地址空间*

        shmaddr = shmat(shmid, NULL, 0);

        if (shmaddr == (char \*)-1) *// 如果失败，打印错误信息并退出*

        {

            perror("shmat error");

            exit(-1);

        }

*// 向共享存储区写入一条消息*

        strcpy(shmaddr, "Hello, this is child process!");

        printf("Child process wrote: %s\n", shmaddr);

*// 将共享存储区从子进程的地址空间分离*

        if (shmdt(shmaddr) == -1) *// 如果失败，打印错误信息并退出*

        {

            perror("shmdt error");

            exit(-1);

        }

        exit(0); *// 子进程结束*

    }

    else *// 如果是父进程*

    {

*// 等待子进程结束*

        wait(NULL);

*// 将共享存储区连接到父进程的地址空间*

        shmaddr = shmat(shmid, NULL, 0);

        if (shmaddr == (char \*)-1) *// 如果失败，打印错误信息并退出*

        {

            perror("shmat error");

            exit(-1);

        }

*// 从共享存储区读取一条消息*

        printf("Parent process received: %s\n", shmaddr);

*// 将共享存储区从父进程的地址空间分离*

        if (shmdt(shmaddr) == -1) *// 如果失败，打印错误信息并退出*

        {

            perror("shmdt error");

            exit(-1);

        }

*// 删除共享存储区*

        if (shmctl(shmid, IPC\_RMID, NULL) == -1) *// 如果失败，打印错误信息并退出*

        {

            perror("shmctl error");

            exit(-1);

        }

    }

    return 0;

}

