北京郵電大學

实验报告：实验 2 进程控制

——实验内容三

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

2024 年 10 月 14 号

目录

1 实验概述

1.1 实验内容

设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一个进程。比如，第一个进程发消息：“I am Here”，第二个进程收到后，将它改变为：“i AM hERE”之后，再发给第一个进程。

提示：

（1）需要创建子进程，父子进程之间通过普通管道进行通信。

（2）需要建立两个普通管道。

1.2 实验环境

1. Windows Subsystem for Linux 2：WSL（Windows Subsystem for Linux）是微软推出的一种在 Windows 操作系统上运行 Linux 的解决方案。WSL 允许用户在 Windows 上运行 Linux 操作系统及其相关的命令行工具和应用程序，而无需使用虚拟机或双重启动配置。
2. Ubuntu 22.04.5 LTS
3. Visual Studio Code 1.94.2：用于连接 wsl 直接进行代码编写，避免使用 vim 等命令行工具，提高编写效率。
4. gcc version 11.4.0

2 程序设计说明

本次实验的核心是利用普通管道（pipe）进行父子进程之间的通信。

具体设计如下：

2.1 管道通信的建立

为了实现双向通信，程序中创建了两个管道，分别用于父进程向子进程发送消息，以及子进程将处理后的消息返回父进程。

通过 pipe() 函数创建两个管道 pipe1 和 pipe2。其中 pipe1 用于父进程向子进程发送消息；pipe2 用于子进程向父进程发送修改后的消息。

具体代码如下：

```
1. if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
2.     perror("pipe");
3.     return 1;
4. }
```

同时需要注意：

- pipe1[0] 表示管道 1 的读端，pipe1[1] 表示管道 1 的写端。
- pipe2[0] 表示管道 2 的读端，pipe2[1] 表示管道 2 的写端。

2.2 处理消息的实现

遍历字符串中的每个字符。通过 isupper()，islower()，tolower()，toupper()函数实现 toggle_case()函数。具体代码如下：

```
1. void toggle_case(char* str) {
2.     for (int i = 0; str[i] != '\0'; i++) {
3.         if (isupper(str[i])) {
4.             str[i] = tolower(str[i]);
5.         } else if (islower(str[i])) {
6.             str[i] = toupper(str[i]);
7.         }
8.     }
9. }
```

2.3 父进程发送消息

父进程首先获取用户输入的字符串消息，并通过 pipe1 发送给子进程。父进程只需要使用 pipe1 的写端，因此关闭 pipe1 的读端。同时，为了接收子进程返回的消息，父进程保持 pipe2 的读端打开，关闭其写端。

具体实现如下：

```
close(pipe1[0]);  // 关闭 pipe1 的读端，因为父进程只需要写入
close(pipe2[1]);  // 关闭 pipe2 的写端，因为父进程只需要读取

// 将消息发送给子进程
write(pipe1[1], message, strlen(message) + 1);
close(pipe1[1]);  // 发送完后关闭 pipe1 的写端
```

2.4 子进程接收并处理消息

子进程从 pipe1 的读端接收父进程发送的消息，将字符串的大小写进行切换，然后通过 pipe2 的写端将修改后的消息发送回父进程。子进程只需要使用 pipe1 的读端和 pipe2 的写端，因此关闭其他不需要的管道端口。

具体实现如下：

```
close(pipe1[1]);  // 关闭 pipe1 的写端，因为子进程只需要读取
close(pipe2[0]);  // 关闭 pipe2 的读端，因为子进程只需要写入

 // 从父进程读取消息
read(pipe1[0], modified_message, BUFFER_SIZE);
close(pipe1[0]);  // 读取完后关闭 pipe1 的读端

toggle_case(modified_message);

// 将修改后的消息发送回父进程
write(pipe2[1], modified_message, strlen(modified_message) + 1);
close(pipe2[1]);  // 发送完消息后关闭 pipe2 的写端
exit(0);
```

2.5 父进程接收修改后的消息

父进程等待子进程结束后，从 pipe2 的读端接收子进程返回的修改后消息，并打印出来。

代码如下：

```
1. // 等待子进程结束并读取修改后的消息
2. wait(NULL);
3. read(pipe2[0], modified_message, BUFFER_SIZE);
4. close(pipe2[0]);
5.
6. printf("Modified message: %s\n", modified_message);
```

3 程序执行结果

3.1 基本功能测试

验证程序在常规情况下的功能，即父进程发送一条含有大小写字母的字符串，子进程能够正确转换其大小写并返回给父进程。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part3$ gcc -o pipetest.exe pipe_case_toggle.c
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: I am here
Modified message: i AM HERE
```

测试通过。

3.2 边界值测试

测试一些极端的输入情况，以确保程序在各种情况下的表现。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: a
Modified message: A
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: D
Modified message: d
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message:
Modified message:
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: 123456789
Modified message: 123456789
```

分别测试了：只输入单个小写字母，单个大写字母，空格，不含字母的字符串的情况下，程序的执行结果。经验证，测试均通过。

3.3 异常值测试

测试特殊输入情况，确保程序对非正常输入的处理能力。

测试结果如下：

```
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: !@#$%^&*()_+
Modified message: !@#$%^&*()_+
```

程序正确处理了含有特殊字符的字符串，即保持原有的特殊字符不变。

测试通过。

3.4 长字符串测试

验证程序处理较长字符串的能力。测试结果如下：

```
demo@SevenBill:~/OS/lab2/part3$ ./pipetest.exe
Please enter a message: This is a very long string to test whether the buffer can handle large in
puts correctly without any truncation or loss of data during the pipe communication.
Modified message: tHIS IS A VERY LONG STRING TO TEST WHETHER THE BUFFER CAN HANDLE LARGE INPUTS C
ORRECTLY WITHOUT ANY TRUNCATION OR LOSS OF DATA DURING THE PIPE COMMUNICATION.
```

测试通过。

4 心得总结

通过本次实验，我对进程间通信的机制，特别是如何通过普通管道在父子进程间传递数据有了更深刻的理解。

实验中，我通过 pipe() 函数创建了两个管道，实现了父子进程之间的双向通信。管道是 Unix/Linux 系统中最简单且高效的进程间通信方式，数据在管道中以字节流的形式传输。这种通信方式简单易用，但也要求严格管理管道的读写端，以避免死锁或数据丢失的问题。通过本次实验，我深刻认识到合理管理管道读写端口的重要性，例如必须在适当的时机关闭未使用的管道端口，以确保进程间通信的顺利进行。

除此之外，我又熟悉了字母大小写相关的函数：isupper()，islower()，tolower()，toupper()。让我对 C 语言的一些函数的使用更加熟练。

在编写代码的过程中，我还特别注意了系统调用可能产生的错误，例如 pipe() 和 fork() 调用失败的情况。通过错误处理机制，我能够确保程序在遇到问题时能够及时反馈，并避免异常退出。

总的来说，这次实验让我系统地学习了如何通过普通管道实现进程间通信，掌握了进程同步、数据传输、错误处理以及资源管理等关键知识点。在未来的编程实践中，我将继续探索更多进程间通信方式以及如何更好地管理并发任务。