

**Lab Report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2023-2024** |
| **Major**: | Software Engineering |
| **Class**: | 2022 |
| **Student Name**: | 吴孜远 |
| **Student ID:** | 222022321062009 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Interprocess communication in Linux --- Named Pipe | | | |
| Date | | May, 2024 | Type | | □Confirmatory  √ Design  √ Comprehensive |
| 1. **Objective & Requirements**    1. Understand named pipe inter-process communication (IPC) in Linux    2. Grasp-named pipe operations    3. Can use a named pipe to write application programs    4. Review multithreaded programming | | | | | |
| 1. **Experimental environment (**platform and software**)**   Ubuntu 18.04 or higher versions | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Task 1 3. Create two processes, called A and B 4. Create two named pipes (using mkfifo function call) , say f1 and f2, that are shared by A and B. 5. Let A and B communicate through f1 and f2.    1. Inside A, create two threads, one reads from f1, and the other writes to f2    2. Inside B, create two threads, one writes to f1, and the other reads from f2 6. The threads in A or B for writing will repeatedly accept an inputted string from the keyboard and then write it to the corresponding fifo 7. The threads in A or B for reading will repeatedly read the string from the corresponding fifo and print it to the screen 8. When a thread in A reads“88”，then both A and B quit; the same for process B. 9. Please provide your procedure to perform the tasks and source codes.   我们直接从main函数开始构建代码，由于AB进程在逻辑上没有什么差别，所以我们看A。  首先我们用int mkfifo(const char \*pathname, mode\_t mode)函数创建两个pipe，分别是myfifo1和myfifo2。.这里传入的mode为0644是配置访问控制的代码，第二位6表示允许User的read和write操作，execute不允许，所以是110=6。第三位和第四位都是4，分别表示只允许Group和other的read操作，100=4。  然后我们就可以开始创建两个线程，先创建两个变量，创建和初始化attribute。然后使用pthread\_create函数创建线程，一个线程是reads，一个是writes。这里我们传入的参数为pipe名，为了方便告诉函数对哪个管道进行操作。  最后用pthread\_join等待线程结束。  代码如下：   1. int main() 2. { 3. mkfifo("./myfifo1", 0644); 4. mkfifo("./myfifo2", 0644); 6. pthread\_t writer, reader; */\* the thread identifier \*/* 8. pthread\_attr\_t attr; */\* set of thread attributes \*/* 10. */\*set the default attributes \*/* 11. pthread\_attr\_init(&attr); 13. pthread\_create(&writer, &attr, writes, (void \*)"./myfifo1"); 14. pthread\_create(&reader, &attr, reads, (void \*)"./myfifo2"); 16. printf("link start!\n"); 18. pthread\_join(writer, NULL); 19. pthread\_join(reader, NULL); 21. return 0; 22. }   接下来我们分别来看线程函数reads和writes，  先看reads，这个函数连接pipe的读端，从管道中读取数据并打印到屏幕上。  首先我们创建一个buf字符数组，用于存储读到的数据，然后使用open(pipe, O\_RDONLY)函数来打开pipe文件，这里是myfifo2，传入的flag是O\_RDONLY，代表以只读的方式打开。.返回值是一个int类型的文件标识符fd，它唯一标识电脑中的某个文件。  之后我们创建一个while无限循环，每次循环使用read(fd, buf, SIZE)函数从文件fd中读取SIZE个字符并存储到buf中。  如果读到“88”（使用srtncmp函数比较，比较前两个字符，如果相同则返回0），我这里先使用close(fd)先关闭文件，然后直接exit(0)直接结束进程。  至于这后面的打印操作，可就讲究了，我们等分析完writes后一起讲。  Reads代码如下：   1. void \*reads(void \*arg){ 2. char \*pipe = (char \*)arg; 3. char buf[SIZE]; 4. int fd = open(pipe, O\_RDONLY); 5. while (1) { 6. int bytes\_read = read(fd, buf, SIZE); 7. printf("\b\b\b\b\b[B]: %s", buf); 8. if (strncmp(buf, "88", 2) == 0) { 9. printf("exit\n"); 10. close(fd); 11. *//pthread\_exit(NULL);* 12. exit(0); 13. } 14. *// if not "88", then continue to write* 15. printf("[A]: "); 16. *// to clean the buffer and output* 17. fflush(stdout); 18. } 19. }   我们看writes函数，这个函数连接pipe的写端，还是重复刚才read的步骤，int fd = open(pipe, O\_WRONLY)，只是flag改为O\_WRONLY，也就是只写模式，使用fgets(buf, SIZE, stdin)，读取屏幕输入到buf中，然后还是判断是否要写“88”，是的话关闭文件，exit程序。  注意我们再每次读之前都先加一个前缀”[A]: ”，这样好看  代码如下：   1. void \*writes(void \*arg){ 2. char \*pipe = (char \*)arg; 3. char buf[SIZE]; 4. int fd = open(pipe, O\_WRONLY); 5. while(1) 6. { 7. printf("[A]: "); 8. fgets(buf, SIZE, stdin); 9. write(fd, buf, SIZE); 10. if (strncmp(buf, "88", 2) == 0) { 11. printf("exit\n"); 12. close(fd); 13. exit(0); 14. } 15. } 16. pthread\_exit(NULL); 17. }   进程B的设置基本全部一样，只不过B的使用myfifo1的读端和myfifo2的写端，两个通道两个线程，同时执行互不干扰。  至于刚才没分析的打印时的问题，我们来看看，以进程A写进程B读为例，在B的reads函数中，我们的打印是printf("\b\b\b\b\b[A]: %s", buf); 在打印读出内容之前我们有5个\b，作用是将占位符向前移动5位（加空格），目的是为了跨过我们在A的writes函数中提前打印的“[A]: ”。然后打印我们真正想要打印的东西，覆盖掉原来我们跨过的“[A]: ”，然后为了恢复原样，我们在下一行补上一个“[A]: ”**（这里的fflush就有作用了，它可以直接将printf缓冲区的内容输出，不需要等待缓存区满，如果不加的话就无法输出）**。这样我们就达到了目的：让每次的读入显示都在写入占位符的上一行。  我还是用make tool来管理编译和链接工作：  Makefile：   1. procA: proc\_A.o 2. gcc proc\_A.o -o procA -lpthread 4. procB: proc\_B.o 5. gcc proc\_B.o -o procB -lpthread 6. proc\_A.o: proc\_A.c 7. gcc -c proc\_A.c 9. proc\_B.o: proc\_B.c 10. gcc -c proc\_B.c 11. clean: 12. rm \*.o 13. rm main   我们看看效果：  首先还是经典的编译连接    运行进程A：    运行进程B：    发现已经建立了两个pipe文件：    互相发消息的测试：  进程A：    进程B：    测试输入88，终止聊天：  进程A：    进程B：    至此，通信成功。  接下来附上完整文件：  proc\_A.c:   1. #include <stdio.h> 2. #include <stdlib.h> 3. #include <sys/types.h> 4. #include <sys/stat.h> 5. #include <fcntl.h> 6. #include <unistd.h> 7. #include <string.h> 8. #include <pthread.h> 9. #include <string.h> 10. *//ssize\_t read(int fd, void \*buf, size\_t count);* 11. #define SIZE 1024 12. void \*writes(void \*arg); 13. void \*reads(void \*arg); 14. int main() 15. { 16. mkfifo("./myfifo1", 0644); 17. mkfifo("./myfifo2", 0644); 19. pthread\_t writer, reader; */\* the thread identifier \*/* 21. pthread\_attr\_t attr; */\* set of thread attributes \*/* 23. */\*set the default attributes \*/* 24. pthread\_attr\_init(&attr); 26. pthread\_create(&writer, &attr, writes, (void \*)"./myfifo1"); 27. pthread\_create(&reader, &attr, reads, (void \*)"./myfifo2"); 29. printf("link start!\n"); 31. pthread\_join(writer, NULL); 32. pthread\_join(reader, NULL); 34. return 0; 35. } 36. void \*writes(void \*arg){ 37. char \*pipe = (char \*)arg; 38. char buf[SIZE]; 39. int fd = open(pipe, O\_WRONLY); 40. while(1) 41. { 42. printf("[A]: "); 43. fgets(buf, SIZE, stdin); 44. write(fd, buf, SIZE); 45. if (strncmp(buf, "88", 2) == 0) { 46. printf("exit\n"); 47. close(fd); 48. exit(0); 49. } 50. } 51. pthread\_exit(NULL); 52. } 53. void \*reads(void \*arg){ 54. char \*pipe = (char \*)arg; 55. char buf[SIZE]; 56. int fd = open(pipe, O\_RDONLY); 57. while (1) { 58. int bytes\_read = read(fd, buf, SIZE); 59. printf("\b\b\b\b\b[B]: %s", buf); 60. if (strncmp(buf, "88", 2) == 0) { 61. printf("exit\n"); 62. close(fd); 63. *//pthread\_exit(NULL);* 64. exit(0); 65. } 66. *// if not "88", then continue to write* 67. printf("[A]: "); 68. *// to clean the buffer and output* 69. fflush(stdout); 70. } 71. }   proc\_B.c:   1. #include <stdio.h> 2. #include <stdlib.h> 3. #include <sys/types.h> 4. #include <sys/stat.h> 5. #include <fcntl.h> 6. #include <unistd.h> 7. #include <string.h> 8. #include <pthread.h> 9. #include <string.h> 10. *//ssize\_t read(int fd, void \*buf, size\_t count);* 11. #define SIZE 1024 12. void \*writes(void \*arg); 13. void \*reads(void \*arg); 14. int main() 15. { 16. mkfifo("./myfifo1", 0644); 17. mkfifo("./myfifo2", 0644); 19. pthread\_t writer, reader; */\* the thread identifier \*/* 21. pthread\_attr\_t attr; */\* set of thread attributes \*/* 23. */\*set the default attributes \*/* 24. pthread\_attr\_init(&attr); 26. pthread\_create(&reader, &attr, reads, (void \*)"./myfifo1"); 27. pthread\_create(&writer, &attr, writes, (void \*)"./myfifo2"); 28. printf("link start!\n"); 30. pthread\_join(writer, NULL); 31. pthread\_join(reader, NULL); 33. return 0; 34. } 35. void \*writes(void \*arg){ 36. char \*pipe = (char \*)arg; 37. char buf[SIZE]; 38. int fd = open(pipe, O\_WRONLY); 39. while(1) 40. { 41. printf("[B]: "); 42. fgets(buf, SIZE, stdin); 43. write(fd, buf, SIZE); 44. if (strncmp(buf, "88", 2) == 0) { 45. printf("exit\n"); 46. close(fd); 47. exit(0); 48. } 50. } 51. close(fd); 52. pthread\_exit(NULL); 53. } 54. void \*reads(void \*arg){ 55. char \*pipe = (char \*)arg; 56. char buf[SIZE]; 57. int fd = open(pipe, O\_RDONLY); 58. while (1) { 59. int bytes\_read = read(fd, buf, SIZE); 60. printf("\b\b\b\b\b[A]: %s", buf); 61. if (strncmp(buf, "88", 2) == 0) { 62. printf("exit\n"); 63. close(fd); 64. *//pthread\_exit(NULL);* 65. exit(0); 66. } 67. *// if not "88", then continue to write* 68. printf("[B]: "); 69. fflush(stdout); 70. } 71. } | | | | | |
| 1. **Result analysis and discussion**   In this part, you are required to provide your analysis of experimental results and summing up the harvest and the existing problems; besides, you are required to provide your thinkings about the questions:   * Recall what you have learned in the Computer Networks class, and think about how to extend the duplex communicating program in this lab to an Instant Messaging tool over the Internet.   这时候我们的通信就不是通过这个管道了，而是需要socket编程，建立客户端程序和服务器端程序，服务器作为用户之间通信的中介。用户的消息就可以通过socket进入传输层，这种通信一般是选择可靠性较高的TCP协议。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |