

**Lab Report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 吴嘉诚 |
| **Student ID:** | 222019321062111 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Interprocess communication in Linux --- Named Pipe | | | |
| Date | | April, 2021 | 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 named pipe to write application programs    4. Review multithreaded programming | | | | | |
| 1. **Experimental environment (**platform and software**)**   Ubuntu 16.04 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 a named pipe (using mkfifo function call) , say **f**, that are shared by A and B. 5. A and B communicate through **f** as follows:    * 1. A repeatedly reads inputs from keyboard and then write the information to **f**      2. B repeatedly reads information from **f** and then output it to the screen   (hint: you many use fgets() to read inputs from keyboard)   1. Task 2 2. Create two processes, called A and B 3. Create two named pipes (using mkfifo function call) , say f1 and f2, that are shared by A and B. 4. 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 5. The threads in A or B for writing will accept an inputted string from the keyboard and then write it to the corresponding fifo 6. The threads in A or B for reading will read the string from the corresponding fifo and print it to the screen 7. When a thread in A reads“88”from or writes“88”to a corresponding fifo, then A quit; the same for process B. 8. Please provide your procedure to perform the tasks and source codes.   **Task1:**   1. **Codes**   t1\_reader.c  int main(int argc, char const \*argv[])  {  char buf[1024];  int fd = open("myfifo",O\_RDONLY);  if(fd == -1)  printf("Open error\n");  else  printf("Open for reading!%d\n", fd);  while(1)  {  if(read(fd,buf,1024) != 0)  printf("%s", buf);  }  close(fd);  return 0;  }    t1\_writer.c  int main(int argc, char const \*argv[])  {  char buf[1024];  int fd = open("myfifo",O\_WRONLY);  if(fd == -1)  printf("Open error\n");  else  printf("Opened for writing! %d\n", fd);  while(fgets(buf,1024,stdin) != NULL)  {  write(fd,buf,1024);  }  close(fd);  return 0;  }   1. **思路分析**   对reader，首先建立readBuf，并打开pipe文件的read端。之后在循环内使用read()不断检测是否有数据输入，若pipe中有数据，则打印到终端中。  对writer，建立writeBuf，打开pipe文件的write端。并循环调用fgets()从键盘读入输入，读入后，使用write()将数据写入到pipe文件中。   1. **测试**     **Task2:**   1. **Codes**   t2-A.c  void \*runnerSend(void \*para);  void \*runnerReceive(void \*para);  bool threadStat = true; //false if any thread exit  int main(int argc, char const \*argv[])  {  mkfifo("./A2B",0644);  mkfifo("./B2A",0644);  int fdA2B = open("A2B",O\_WRONLY);  int fdB2A = open("B2A",O\_RDONLY);  if(fdA2B == -1 || fdA2B == -1)  printf("Open error\n");  else  {  printf("Start Connecting...\n");  pthread\_t tid1, tid2;  pthread\_attr\_t attr1,attr2;  pthread\_attr\_init(&attr1);  pthread\_attr\_init(&attr2);  printf("Terminal A Connect success!\n============================================================\n");  pthread\_create(&tid1,&attr1,runnerSend,&fdA2B);  pthread\_create(&tid2,&attr2,runnerReceive,&fdB2A);  //thread create finish  if(pthread\_join(tid2,NULL) == 0 || pthread\_join(tid1,NULL) == 0);  }  return 0;  }  void \*runnerSend(void \*para)  {  int fd = \*(int \*)para;  char buf\_send[1024];  char\* c88 = "88\n\0";  while(threadStat)  {  printf("\b\b\b\b\b\b"); //backspace for [A]output  printf("[A]:> ");  fgets(buf\_send,1024,stdin);  write(fd,buf\_send,1024);  if(strcmp(buf\_send,c88) == 0) //check exit condition  break;  }  threadStat = false;  pthread\_exit(NULL);  }  void \*runnerReceive(void \*para)  {  int fd = \*(int \*)para;  char buf\_receive[1024];  char\* c88 = "88\n\0";  while(threadStat)  {  if(read(fd,buf\_receive,1024) != 0)  {  printf("\b\b\b\b\b\b"); //backspace for [B]output  printf("[B]:> %s", buf\_receive);  if(strcmp(buf\_receive,c88) == 0)  break;  printf("[A]:> ");  fflush(stdout); //put [A]:> again  }  }  threadStat = false;  pthread\_exit(NULL);  }  t2-B.c(quite similar to t2-A.c)  void \*runnerSend(void \*para);  void \*runnerReceive(void \*para);  bool threadStat = true;  int main(int argc, char const \*argv[])  {  mkfifo("./A2B",0644);  mkfifo("./B2A",0644);  int fdA2B = open("A2B",O\_RDONLY);  int fdB2A = open("B2A",O\_WRONLY);  if(fdA2B == -1 || fdA2B == -1)  printf("Open error\n");  else  {  printf("Start Connecting...\n");  pthread\_t tid1, tid2;  pthread\_attr\_t attr1,attr2;  pthread\_attr\_init(&attr1);  pthread\_attr\_init(&attr2);  printf("Terminal B Connect success!\n============================================================\n");  pthread\_create(&tid1,&attr1,runnerSend,&fdB2A);  pthread\_create(&tid2,&attr2,runnerReceive,&fdA2B);  if(pthread\_join(tid2,NULL) == 0 || pthread\_join(tid1,NULL) == 0);  }  close(fdA2B);  close(fdB2A);  return 0;  }  void \*runnerSend(void \*para)  {  int fd = \*(int \*)para;  char buf\_send[1024];  char\* c88 = "88\n\0";  while(threadStat)  {  printf("\b\b\b\b\b\b");  printf("[B]:> ");  fgets(buf\_send,1024,stdin);  write(fd,buf\_send,1024);  if(strcmp(buf\_send,c88) == 0)  break;  }  threadStat = false;  pthread\_exit(NULL);  }  void \*runnerReceive(void \*para)  {  int fd = \*(int \*)para;  char buf\_receive[1024];  char\* c88 = "88\n\0";  while(threadStat)  {  if(read(fd,buf\_receive,1024) != 0)  {  printf("\b\b\b\b\b\b");  printf("[A]:> %s", buf\_receive);  if(strcmp(buf\_receive,c88) == 0)  break;  printf("[B]:> ");  fflush(stdout);  }  }  threadStat = false;  pthread\_exit(NULL);  }   1. **思路分析**   首先创建pipe文件，并打开文件的read和write端，获得file descriptor。  打开成功后，创建两个线程，并将两个fd作为参数传递给两个线程。  对于发送端线程。在循环中循环打印”[A]:> ”并调用fgets()获得输入，储存到本线程的buf\_send中。获得输入后，调用write()将数据写入到pipe文件中，并用strcmp()检测输入是否为”88\n\0”(\n\0是由于fget)，若是，则退出循环，将threadStat置为false以提示receive线程退出，并且本线程退出。  对于接收端线程。循环检测pipe是否有新输入，若有，则将其储存到本线程的\_receive中，并打印退格符’\b’以删除已输出的”[B]:>”(或[A])。每次循环中都检测获得的数据是否为”88\n\0”，若是，则将threadStat置为false，并退出本线程。  进程中，在创建两个线程后，调用pthread\_join()等待线程结束。使用if的原因是为了使得receive线程结束后，不必等待send线程中的fgets()，直接结束进程。   1. **测试**     **成功！** | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   **实验结果：**  **经过调试修改，实验结果完全符合老师要求（尤其是任务2）**  **实验收获：**  **加深了对线程间信息共享的理解：全局变量（堆）数据共享**  **理解并实验了进程间通信的一种方式：管道pipe**  **加深了对fgets()的理解：它接收字符串后会在其后自动加上’\0’**  **理解了退格符’\b’的使用方式，以及输出缓冲区的意义和使用** | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |