

**Lab Report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **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:**  The code of fifo\_reader as followed:  #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <string.h>  #define SIZE 1024  //ssize\_t read(int fd, void \*buf, size\_t count);  int main()  {      char str[SIZE];      int fd = open("myfifo", O\_RDONLY);      if(fd == -1)          printf("Open fifo error!\n");      else          printf("Fifo opened for reading!\n");      read(fd, str, SIZE);      while(strcmp(str, "88\n"))      {          printf("String read: %s", str);          read(fd, str, SIZE);      }      printf("String read: %s", str);      close(fd);      return 0;  }  The code for writer is similar to  Create pipelines, open and use them:    You can see that the content entered at the reader can be output at the writer until inputting “88”    **Task2:**  The code on one side is shown below, and the code on the other side is similar, except that it uses a different pipeline:  The main function creates two threads for input and output respectively. In the program, I use some additional thread operations, such as stopping the threads synchronously, so as to avoid duplicate outputs caused by content being stuck in the pipeline. In order to do this synchronously, I defined structs so that I can pass multiple parameters when creating a thread and thus be able to control the opening and closing of both threads to keep them synchronized.  #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <string.h>  #include <pthread.h>  #include <stdlib.h>  #include <signal.h>  #define SIZE 1024  void \*readMsg(void \*param);  void \*writeMsg(void \*param);  void stop();  int flag = 1;  struct pps {      int fd1, fd2;      char\* buf1;      char\* buf2;      pthread\_t tid1, tid2;  } pp;  int main() {      char buf1[SIZE];      int fd1 = open("fifo1", O\_RDONLY);      if(fd1 == -1)          printf("Open fifo1 error!\n");      else          printf("Fifo1 opened for reading!\n");        char buf2[SIZE];      int fd2 = open("fifo2", O\_WRONLY);      if(fd2 == -1)          printf("Open fifo2 error!\n");      else          printf("Fifo2 opened for writing!\n");        pp.fd1 = fd1;      pp.buf1 = buf1;      pp.fd2 = fd2;      pp.buf2 = buf2;        pthread\_t tid1,tid2;      pthread\_attr\_t attr1, attr2;      pthread\_attr\_init(&attr1);      pthread\_attr\_init(&attr2);      pthread\_create(&tid1, &attr1, readMsg, &pp);      pthread\_create(&tid2, &attr2, writeMsg, &pp);        pp.tid1 = tid1;      pp.tid2 = tid2;        pthread\_join(tid1, NULL);      pthread\_join(tid2, NULL);        return 0;  }  void stop(int fd1,int fd2) {      close(fd1);      close(fd2);      flag = 0;  }    void \*readMsg(void \*param) {      pthread\_setcanceltype(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);        struct pps \*pp;      pp = (struct pps\*) param;        while(flag)      {          read(pp->fd1, pp->buf1, SIZE);          printf("\r[B]: %s", pp->buf1);            if(!strcmp(pp->buf1, "88\n") || !strcmp(pp->buf2, "88\n"))              break;            printf("[A]: ");          fflush(stdout);      }        stop(pp->fd2, pp->fd1);      pthread\_cancel(pp->tid2);      pthread\_exit(NULL);  }  void \*writeMsg(void \*param) {      pthread\_setcanceltype(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);      struct pps \*pp;      pp = ( struct pps\*) param;        while(flag)      {          printf("[A]: ");          fgets(pp->buf2, SIZE, stdin);          write(pp->fd2, pp->buf2, SIZE);            if(!strcmp(pp->buf1, "88\n") || !strcmp(pp->buf2, "88\n"))              break;      }        stop(pp->fd2, pp->fd1);      pthread\_cancel(pp->tid1);      pthread\_exit(NULL);  }  Compile and run the program, we can see that the program can send messages to each other normally, and after entering "88", close the pipe and end the both program | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Task2 of this experiment was difficult and took me a lot of time. In order to finish it, I consulted a lot of information, used knowledge that was not taught in class, and at the same time, I actively discussed it with my classmates, and finally got a result that satisfied me. Through this experiment, I gained a deeper understanding of threads and learned how to use pipes and be able to communicate between threads by creating them. This experiment has achieved the purpose of the experiment well. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |