Politechnika Świętokrzyska
Kielce University of Technology

Operating systems 1

Pipes and named pipes in Linux

Tanmay Nandanikar | Lab report 3 | 10/30/20

dr inż. Karol Tomaszewski

INTRODUCTION

This lab primarily focused on communication between processes in Linux using streams, pipes, and named pipes (aka FIFO queues).

1 BRIEF OVERVIEW OF THEORY

1.1 STREAMS IN LINUX

Streams can be used for one-way communication between parent process and child processes. The parent process can only perform one of either read or write operations on a stream. Depending on whether it is read or written, it 'replaces' the keyboard input or screen output.

1.2 PIPES FOR COMMUNICATION BETWEEN PROCESSES

Pipes can be used for communication between two related processes (eg. Between a parent process and a child process or between 2 child processes). The `O_NONBLOCK` flag can be set to make a process wait for queue operations to be completed.

1.3 FIFO QUEUES

Named pipes, or FIFO queues, can be used to communicate between unrelated processes. I personally found these easier to work with compared to unnamed pipes.

2 EXERCISES

2.1 CALLING THE SORT COMMAND THROUGH C

A program that calls the sort command through C using popen(). Source code can be found in the source section (here). Output for this task is shown below:

2.1.1 Output for Task 1

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out




    }
    }
}
}
    if(a!=0){
 #include <signal.h>
```

2.2 CALLING THE WC COMMAND

A program that calls the wc command for an input string. Source code can be found in the source section (here). Output for this task is shown below:

2.2.1 Output for Task 2

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
Enter string as argument:
this_is_a_string
      0       1      17
```

2.3 PIPE FOR DATA TRANSFER

Program demonstrating use of pipes for data transfer between processes. Source code can be found in the source section (here). Output for this task is shown below:

2.3.1 Output for Task 3

```
input a number:
47
Got number from child process: 47
```

2.4 SETTING NONBLOCK FLAG BASED ON ARGUMENTS

If therer are more than 3 arguments, the O_NONBLOCK flag is set. Source code can be found in the source section (here). Output for this task is shown below:

```
1. input a number:
2. 34
3. Got number from child process: 47
4.
5.
6. sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out olad
   wisjnda dbhasjd 33 44 22 55 3242 fsdf
7. NONBLOCK set:
8. input a number:
9. 22
10.  Got number from child process: 22
```

2.5 TWO-WAY COMMUNICATION USING PIPES

Program demonstrating 2-way communication between processes. Source code can be found in the source section (here). Output for this task is shown below

2.5.1 Output for Task 5

```
1. sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
2. In Parent: Writing to pipe 1 – Message is Something
3. In Child: Reading from pipe 1 – Message is Something
4. In Child: Writing to pipe 2 – Message is this
5. In Parent: Reading from pipe 2 – Message is this
```

2.6 SET NONBLOCK FLAG AND UNLINK FIFO QUEUE ON TERMINATION

Program to demonstrate FIFO queue. Source code can be found in the source section (here). Output for this task is shown below:

2.6.1 Output for Task 6

```
1. output 1.1:
2. P1 wrote 3
3. P1 wrote 5
```

```
4. P1 wrote 7
5. P1 wrote 9
6. P1 wrote 11
7. P1 wrote 13
8. P1 wrote 15
9. P1 wrote 17
10.  P1 wrote 19
11.  P1 wrote 21
12.  output 1.2:
13.  User1: 3
14.  User1: 5
15.  User1: 7
16.  User1: 9
17.  User1: 11
18.  User1: 13
19.  User1: 15
20.  User1: 17
21.  User1: 19
22.  User1: 21
```

```
1. Output 2.1:
2. sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out hnf
   jbfak  fheabk kaf 33 33 44 55
3. NONBLOCK set:
4. P1 wrote 3
5. P1 wrote 5
6. P1 wrote 7
7. P1 wrote 9
8. P1 wrote 11
9. P1 wrote 13
10.  P1 wrote 15
11.  P1 wrote 17
12.  P1 wrote 19
13.  P1 wrote 21
14.  output 2.2:
15.  User1: 3
16.  User1: 5
17.  User1: 7
18.  User1: 9
19.  User1: 11
20.  User1: 13
21.  User1: 15
22.  User1: 17
23.  User1: 19
24.  User1: 21
```

2.7 TWO-WAY COMMUNICATION USING AN UNNAMED AND A NAMED PIPE

Two-way communication between processes using unnamed pipe for one direction and named pipe for the other direction. Source code can be found in the source section (here). Output for this task is shown below:

2.7.1 Output for Task 7

```
In Parent: Writing to pipe 1 – Message is Hi
In Child: Reading from pipe 1 – Message is Hi
In Child: Writing to pipe 2 – Message is Hello
In Parent: Reading from pipe 2 – Message is Hello
```

2.8 CHAT PROGRAM USING MKNOD()

Terminal chat program using mknod(). Source code can be found in the source section (here). Output for this task is shown below:

2.8.1 Output for Task 8

```
Output terminal 1:
sprices@pop-os:~/Documents/operating systems/lab4/ex8$ gcc ex7.c
sprices@pop-os:~/Documents/operating systems/lab4/ex8$ gcc -o ex7
ex7_2.c
sprices@pop-os:~/Documents/operating systems/lab4/ex8$ ./a.out
message 1
User2: message 2

msg3
msg4
 User2: msg4

 Output terminal 2:
 sprices@pop-os:~/Documents/operating systems/lab4/ex8$ ./ex7
 User1: message 1

 message 2
 User1: msg3

 msg4
 User1: msg4
```

2.9 TASK 9

I encountered faulty outputs for this task.

2.10 TASK 10

Source code can be found in the source section (here). Output for this task is shown below:

2.10.1 Output for Task 10

```
Output terminal 1:
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex10.1
P1 wrote 3
P1 wrote 5
P1 wrote 7
P1 wrote 9
P1 wrote 11
P1 wrote 13
P1 wrote 15
 P1 wrote 17
 P1 wrote 19
 P1 wrote 21
 P1 wrote 23
 sprices@pop-os:~/Documents/operating systems/lab4$ ./ex10.2
 P1 wrote 2
 P1 wrote 4
 P1 wrote 6
 P1 wrote 8
 P1 wrote 10
 P1 wrote 12
 P1 wrote 14
 P1 wrote 16
 P1 wrote 18
 P1 wrote 20

  Output terminal 2:
 sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
 User1: 3
 User1: 5
 User1: 7
 User1: 7
 User1: 9
 User1: 11
```

```
34.  User1: 13
35.  User1: 15
36.  User1: 17
37.  User1: 19
38.  User1: 21
39.  User1: 23
40.  User1: 23
41.  User1: 2
42.  User1: 4
43.  User1: 6
44.  User1: 8
45.  User1: 10
46.  User1: 12
47.  User1: 14
48.  User1: 16
49.  User1: 16
50.  User1: 18
51.  User1: 20
52.  User1: 20
```

3 CONCLUSION

During this lab, I learnt about communication between related as well as unrelated processes. I learnt how unnamed pipes and streams can be used within related functions to communicated between child and parent processes. I also learnt how to use named pipes to communicate between several unrelated programs.

4 SOURCE[1]

4.1 SOURCE CODE FOR EXERCISE ONE

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>


void call_1(){

    FILE *stream = popen("sort -o name.txt name.c","w");
    pclose(stream);
}

void call_2(){

    FILE *ptr=fopen("name.txt","r");


    if(ptr != NULL){
        char sort[50];

        for(int i=0;i<50;i++)
            sort[i]=fgetc(ptr);

       for(int i=0;i<50;i++)
            printf("%c", sort[i]);
        fclose(ptr);
    }
}

int main(){

    call_1();
    call_2();
    return 0;
}
```

[1] In several programs, I did not divide the code into functions because for these specific programs, except for division into child/parent processes no other functions seemed sensible. It would have resulted in just one function call in main leading to a different function.

4.2 SOURCE CODE FOR EXERCISE TWO

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>


void call_wc(char test[]){

FILE *ptr=fopen("hello.txt","r+");

 if(ptr != NULL){
 fprintf(ptr,"%s",test);
 fclose(ptr);
 }

 FILE *stream = popen("cat hello.txt | wc","w");
 pclose(stream);
 }


 int main(){

 char test[256];
 printf("Enter string as argument:\n");
 scanf(" %s", test);

 call_wc(test);

 return 0;
 }
```

4.3 SOURCE CODE FOR EXERCISE THREE

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>


void child_process(int fd[]){
    close(fd[0]);
    int x;
    printf("input a number:\n");
      scanf(" %d", &x);
      write(fd[1], &x, sizeof(int));
      close(fd[1]);
 }
```

```
14.
15.  void parent_process(int fd[]){
16.      close(fd[1]);
17.      int y;
18.      read(fd[0], &y, sizeof(int));
19.      close(fd[0]);
20.      printf("Got number from child process: %d\n", y);
21.  }
22.
23.  int main(int argc, char *argv[]){
24.  int fd[2];
25.
26.  if(pipe(fd)==-1){
27.      printf("Error occurred\n");
28.      return 1;
29.  }
30.  int id=fork();
31.  if(id<0)
32.          perror("fork error");
33.
34.
35.  if(id==0){
36.      child_process(fd);
37.  }
38.  else{
39.      parent_process(fd);
40.  }
41.  return 0;
42.  }
```

4.4 Source code for Exercise Four

```
1.  #include <fcntl.h>
2.  #include <stdio.h>
3.  #include <unistd.h>
4.  #include <stdlib.h>
5.
6.  void child_process(int fd1[]){
7.    close(fd1[0]);
8.          int x;
9.          printf("input a number:\n");
10.           scanf(" %d", &x);
11.           write(fd1[1], &x, sizeof(int));
12.           close(fd1[1]);
13.  }
14.
15.  void parent_process(int fd1[]){
```

```
        close(fd1[1]);
        int y;
        read(fd1[0], &y, sizeof(int));
        close(fd1[0]);
        printf("Got number from child process: %d\n", y);
}

int main(int argc, char *argv[]){
    int fd1[2];
    int fd;

    if(argc>3){
        fcntl(fd, F_SETFL, O_NONBLOCK);
        printf("NONBLOCK set:\n");
    }

    if(pipe(fd1)==-1){
        printf("Error occurred\n");
        return 1;
    }

    int id=fork();

    if(id<0){
        perror("fork error");
    }
    if(id==0){
    child_process(fd1);
    }
    else{
    parent_process(fd1);
    }

    return 0;
}
```

4.5 Source code for Exercise Five

```
#include<stdio.h>
#include<unistd.h>

int main() {
   int pipefds1[2], pipefds2[2];
   int returnstatus1, returnstatus2;
   int pid;
   char pipe1writemessage[20] = "Hi";
```

```
9.    char pipe2writemessage[20] = "Hello";
10.    char readmessage[20];
11.    returnstatus1 = pipe(pipefds1);
12.
13.    if (returnstatus1 == -1) {
14.       printf("Unable to create pipe 1 \n");
15.       return 1;
16.    }
17.    returnstatus2 = pipe(pipefds2);
18.
19.    if (returnstatus2 == -1) {
20.       printf("Unable to create pipe 2 \n");
21.       return 1;
22.    }
23.    pid = fork();
24.
25.    if (pid != 0)  {
26.       close(pipefds1[0]);
27.       close(pipefds2[1]);
28.       printf("In Parent: Writing to pipe 1 – Message is %s\n",
   pipe1writemessage);
29.       write(pipefds1[1], pipe1writemessage,
   sizeof(pipe1writemessage));
30.       read(pipefds2[0], readmessage, sizeof(readmessage));
31.       printf("In Parent: Reading from pipe 2 – Message is
   %s\n", readmessage);
32.    } else {
33.       close(pipefds1[1]);
34.       close(pipefds2[0]);
35.       read(pipefds1[0], readmessage, sizeof(readmessage));
36.       printf("In Child: Reading from pipe 1 – Message is %s\n",
   readmessage);
37.       printf("In Child: Writing to pipe 2 – Message is %s\n",
   pipe2writemessage);
38.       write(pipefds2[1], pipe2writemessage,
   sizeof(pipe2writemessage));
39.    }
40.    return 0;
41. }
```

4.6 Source code for Exercise Six

4.6.1 Part 1

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
      int fd2;
      if(argc>3){
          fcntl(fd, F_SETFL, O_NONBLOCK);
          printf("NONBLOCK set:\n");
      }
      char * myfifo = "/tmp/myfifo";

      mkfifo(myfifo, 0666);

      int n;
      while (1)
      {
          fd2 = open(myfifo,O_RDONLY);
          read(fd2, &n, sizeof(n));

          printf("User1: %d\n", n);
          if(n>20){
          break;
          }
          close(fd2);
      }
      if(unlink(myfifo)<0)
          perror("unlink");
      return 0;
 }
```

4.6.2 Part 2

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>

int main()
 {
     int fd1;
     int i=1;

     char * myfifo = "/tmp/myfifo";

     mkfifo(myfifo, 0666);

     char arr1[80], arr2[80];
     while (i<22)
     {
         i+=2;
         fd1 = open(myfifo, O_WRONLY);
         write(fd1, &i, sizeof(i));
         sleep(1);
         printf("P1 wrote %d\n",i);
         close(fd1);
     }
     return 0;
 }
```

4.7 SOURCE CODE FOR EXERCISE SEVEN

```
1. #include<stdio.h>
2. #include<unistd.h>
3. #include <fcntl.h>
4. #include <sys/stat.h>
5. #include <sys/types.h>
6.
7. int main() {
8.    int pipefds1[2];
9.    char * myfifo = "/tmp/myfifo";
10.    mkfifo(myfifo, 0666);
11.    int returnstatus1;
12.    int pid;
13.    char pipe1writemessage[20] = "Hi";
14.    char pipe2writemessage[20] = "Hello";
15.    char readmessage[20];
16.    returnstatus1 = pipe(pipefds1);
17.
18.    if (returnstatus1 == -1) {
19.       printf("Unable to create pipe 1 \n");
20.       return 1;
21.    }
22.
23.    pid = fork();
24.    int pipefds2;
25.
26.    if (pid != 0)  {
27.       close(pipefds1[0]);
28.       pipefds2=open(myfifo,O_RDONLY);
29.       printf("In Parent: Writing to pipe 1 – Message is %s\n",
   pipe1writemessage);
30.       write(pipefds1[1], pipe1writemessage,
   sizeof(pipe1writemessage));
31.       read(pipefds2, readmessage, sizeof(readmessage));
32.       printf("In Parent: Reading from pipe 2 – Message is
   %s\n", readmessage);
33.       close(pipefds2);
34.    } else {
35.       close(pipefds1[1]);
36.       pipefds2=open(myfifo,O_WRONLY);
37.       read(pipefds1[0], readmessage, sizeof(readmessage));
38.       printf("In Child: Reading from pipe 1 – Message is %s\n",
   readmessage);
39.       printf("In Child: Writing to pipe 2 – Message is %s\n",
   pipe2writemessage);
40.       write(pipefds2, pipe2writemessage,
   sizeof(pipe2writemessage));
```

```
        close(pipefds2);
    }
    return 0;
}
```

4.8 Source code for Exercise Eight

4.8.1 Part 1

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";

    int fifo_str = mknod("myfifo", 'b', 0);

    char arr1[80], arr2[80];
    while (1)
    {
        fd = open(myfifo, O_WRONLY);

        fgets(arr2, 80, stdin);

        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        fd = open(myfifo, O_RDONLY);

        read(fd, arr1, sizeof(arr1));

        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

4.8.2 Part 2

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
      int fd1;

      char * myfifo = "/tmp/myfifo";

      int fifo_str = mknod("myfifo", 'b', 0);

      char str1[80], str2[80];
      while (1)
      {
          fd1 = open(myfifo,O_RDONLY);
          read(fd1, str1, 80);

          printf("User1: %s\n", str1);
          close(fd1);

          fd1 = open(myfifo,O_WRONLY);
          fgets(str2, 80, stdin);
          write(fd1, str2, strlen(str2)+1);
          close(fd1);
      }
      return 0;
 }
```

4.10 Source code for Exercise Ten

4.10.1 Part 1

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
      int fd2;

      char * myfifo = "/tmp/myfifo";

      mkfifo(myfifo, 0666);

      int n;
      while (1)
      {
          fd2 = open(myfifo,O_RDONLY);
          read(fd2, &n, sizeof(n));

          printf("User1: %d\n", n);
          close(fd2);
      }
      return 0;
 }
```

4.10.2 Part 2

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>

int main()
 {
     int fd1;
     int i=1;

     char * myfifo = "/tmp/myfifo";

     mkfifo(myfifo, 0666);

     char arr1[80], arr2[80];
     while (i<22)
     {
         i+=2;
         fd1 = open(myfifo, O_WRONLY);
         write(fd1, &i, sizeof(i));
         sleep(1);
         printf("P1 wrote %d\n",i);
         close(fd1);
     }
     return 0;
 }
```

4.10.3 Part 3

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
 {
     int fd;
     int i=0;

     char * myfifo = "/tmp/myfifo";

     mkfifo(myfifo, 0666);

     char arr1[80], arr2[80];
     while (i<20)
     {
         i+=2;
         fd = open(myfifo, O_WRONLY);
         write(fd, &i, sizeof(i));
         sleep(1);
         printf("P1 wrote %d\n",i);
         close(fd);
     }
     return 0;
 }
```