Inter Process Communication - Named Pipes

Previous Page Next Page

Pipes were meant for communication between related processes. Can we use pipes for unrelated process communication, say, we want to execute client program from one terminal and the server program from another terminal? The answer is No. Then how can we achieve unrelated processes communication, the simple answer is Named Pipes. Even though this works for related processes, it gives no meaning to use the named pipes for related process communication.

We used one pipe for one-way communication and two pipes for bi-directional communication. Does the same condition apply for Named Pipes. The answer is no, we can use single named pipe that can be used for two-way communication (communication between the server and the client, plus the client and the server at the same time) as Named Pipe supports bi-directional communication.

Another name for named pipe is **FIFO (First-In-First-Out)**. Let us see the system call (mknod()) to create a named pipe, which is a kind of a special file.

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

int mknod(const char *pathname, mode_t mode, dev_t dev);
```

This system call would create a special file or file system node such as ordinary file, device file, or FIFO. The arguments to the system call are pathname, mode and dev. The pathname along with the attributes of mode and device information. The pathname is relative, if the directory is not specified it would be created in the current directory. The mode specified is the mode of file which specifies the file type such as the type of file and the file mode as mentioned in the following tables. The dev field is to specify device information such as major and minor device numbers.

File Type	Description	File Type	Description

S_IFBLK	block special	S_IFREG	Regular file
S_IFCHR	character special	S_IFDIR	Directory
S_IFIFO	FIFO special	S_IFLNK	Symbolic Link

File Mode	Description	File Mode	Description
S_IRWXU	Read, write, execute/search by owner	S_IWGRP	Write permission, group
S_IRUSR	Read permission, owner	S_IXGRP	Execute/search permission, group
S_IWUSR	Write permission, owner	S_IRWXO	Read, write, execute/search by others
S_IXUSR	Execute/search permission, owner	S_IROTH	Read permission, others
S_IRWXG	Read, write, execute/search by group	S_IWOTH	Write permission, others
S_IRGRP	Read permission, group	S_IXOTH	Execute/search permission, others

File mode can also be represented in octal notation such as 0XYZ, where X represents owner, Y represents group, and Z represents others. The value of X, Y or Z can range from 0 to 7. The values for read, write and execute are 4, 2, 1 respectively. If needed in combination of read, write and execute, then add the values accordingly.

Say, if we mention, 0640, then this means read and write (4 + 2 = 6) for owner, read (4) for group and no permissions (0) for others.

This call would return zero on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode)
```

This library function creates a FIFO special file, which is used for named pipe. The arguments to this function is file name and mode. The file name can be either absolute path or relative path. If full path name (or absolute path) is not given, the file would be created in the current folder of the executing process. The file mode information is described in mknod() system call.

This call would return zero on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

Let us consider a program of running the server on one terminal and running the client on another terminal. The program would only perform one-way communication. The client accepts the user input and sends the message to the server, the server prints the message on the output. The process is continued until the user enters the string “end”.

Let us understand this with an example –

Step 1 – Create two processes, one is fifoserver and another one is fifoclient.

Step 2 – Server process performs the following –

Creates a named pipe (using system call mknod()) with name “MYFIFO”, if not created.

Opens the named pipe for read only purposes.

Here, created FIFO with permissions of read and write for Owner. Read for Group and no permissions for Others.

Waits infinitely for message from the Client.

If the message received from the client is not “end”, prints the message. If the message is “end”, closes the fifo and ends the process.

Step 3 – Client process performs the following –

Opens the named pipe for write only purposes.

Accepts the string from the user.

Checks, if the user enters “end” or other than “end”. Either way, it sends a message to the server. However, if the string is “end”, this closes the FIFO and also ends the process.

Repeats infinitely until the user enters string “end”.

Now let’s take a look at the FIFO server file.

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

#define FIFO_FILE "MYFIFO"
int main() {
   int fd;
   char readbuf[80];
```

```
   char end[10];
   int to_end;
   int read_bytes;

   /* Create the FIFO if it does not exist */
   mknod(FIFO_FILE, S_IFIFO|0640, 0);
   strcpy(end, "end");
   while(1) {
      fd = open(FIFO_FILE, O_RDONLY);
      read_bytes = read(fd, readbuf, sizeof(readbuf));
      readbuf[read_bytes] = '\0';
      printf("Received string: \"%s\" and length is %d\n", readbuf, (int)strlen(readbuf));
      to_end = strcmp(readbuf, end);
      if (to_end == 0) {
         close(fd);
         break;
      }
   }
   return 0;
}
```

Compilation and Execution Steps

```
Received string: "this is string 1" and length is 16

Received string: "fifo test" and length is 9

Received string: "fifo client and server" and length is 22

Received string: "end" and length is 3
```

Now, let’s take a look at the FIFO client sample code.

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

#define FIFO_FILE "MYFIFO"
int main() {
   int fd;
   int end_process;
   int stringlen;
   char readbuf[80];
   char end_str[5];
   printf("FIFO_CLIENT: Send messages, infinitely, to end enter \"end\"\n");
   fd = open(FIFO_FILE, O_CREAT|O_WRONLY);
   strcpy(end_str, "end");

   while (1) {
      printf("Enter string: ");
      fgets(readbuf, sizeof(readbuf), stdin);
      stringlen = strlen(readbuf);
      readbuf[stringlen - 1] = '\0';
      end_process = strcmp(readbuf, end_str);

      //printf("end_process is %d\n", end_process);
```

```
      if (end_process != 0) {
         write(fd, readbuf, strlen(readbuf));
         printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));
      } else {
         write(fd, readbuf, strlen(readbuf));
         printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));
         close(fd);
         break;
      }
   }
   return 0;
}
```

Let’s take a at the arriving output.

Compilation and Execution Steps

```
FIFO_CLIENT: Send messages, infinitely, to end enter "end"
Enter string: this is string 1
Sent string: "this is string 1" and string length is 16
Enter string: fifo test
Sent string: "fifo test" and string length is 9
Enter string: fifo client and server
Sent string: "fifo client and server" and string length is 22
Enter string: end
Sent string: "end" and string length is 3
```

Two-way Communication Using Named Pipes

The communication between pipes are meant to be unidirectional. Pipes were restricted to one-way communication in general and need at least two pipes for two-way communication. Pipes are meant for inter-related processes only. Pipes can’t be used for unrelated processes communication, say, if we want to execute one process from one terminal and another process from another terminal, it is not possible with pipes. Do we have any simple way of communicating between two processes, say unrelated processes in a simple way? The answer is YES. Named pipe is meant for communication between two or more unrelated processes and can also have bi-directional communication.

Already, we have seen the one-directional communication between named pipes, i.e., the messages from the client to the server. Now, let us take a look at the bi-directional communication i.e., the client sending message to the server and the server receiving the message and sending back another message to the client using the same named pipe.

Following is an example –

Step 1 – Create two processes, one is fifoserver_twoway and another one is fifoclient_twoway.

Step 2 – Server process performs the following –

Creates a named pipe (using library function mkfifo()) with name "fifo_twoway" in /tmp directory, if not created.

Opens the named pipe for read and write purposes.

Here, created FIFO with permissions of read and write for Owner. Read for Group and no permissions for Others.

Waits infinitely for a message from the client.

If the message received from the client is not "end", prints the message and reverses the string. The reversed string is sent back to the client. If the message is "end", closes the fifo and ends the process.

Step 3 – Client process performs the following –

Opens the named pipe for read and write purposes.

Accepts string from the user.

Checks, if the user enters "end" or other than "end". Either way, it sends a message to the server. However, if the string is "end", this closes the FIFO and also ends the process.

If the message is sent as not "end", it waits for the message (reversed string) from the client and prints the reversed string.

Repeats infinitely until the user enters the string "end".

Now, let's take a look at FIFO server sample code.

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

#define FIFO_FILE "/tmp/fifo_twoway"
void reverse_string(char *);
int main() {
   int fd;
   char readbuf[80];
   char end[10];
   int to_end;
   int read_bytes;

   /* Create the FIFO if it does not exist */
   mkfifo(FIFO_FILE, S_IFIFO|0640);
   strcpy(end, "end");
   fd = open(FIFO_FILE, O_RDWR);
```

```
   while(1) {
      read_bytes = read(fd, readbuf, sizeof(readbuf));
      readbuf[read_bytes] = '\0';
      printf("FIFOSERVER: Received string: \"%s\" and length is %d\n", readbuf, (int)strlen(readb
      to_end = strcmp(readbuf, end);

      if (to_end == 0) {
         close(fd);
         break;
      }
      reverse_string(readbuf);
      printf("FIFOSERVER: Sending Reversed String: \"%s\" and length is %d\n", readbuf, (int) str
      write(fd, readbuf, strlen(readbuf));
      /*
      sleep - This is to make sure other process reads this, otherwise this
      process would retrieve the message
      */
      sleep(2);
   }
   return 0;
}

void reverse_string(char *str) {
   int last, limit, first;
   char temp;
   last = strlen(str) - 1;
   limit = last/2;
   first = 0;

   while (first < last) {
      temp = str[first];
      str[first] = str[last];
      str[last] = temp;
      first++;
      last--;
   }
   return;
}
```

Compilation and Execution Steps

```
FIFOSERVER: Received string: "LINUX IPCs" and length is 10

FIFOSERVER: Sending Reversed String: "sCPI XUNIL" and length is 10

FIFOSERVER: Received string: "Inter Process Communication" and length is 27

FIFOSERVER: Sending Reversed String: "noitacinummoC ssecorP retnI" and length is 27

FIFOSERVER: Received string: "end" and length is 3
```

Now, let’s take a look at FIFO client sample code.

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

```
#define FIFO_FILE "/tmp/fifo_twoway"
int main() {
   int fd;
   int end_process;
   int stringlen;
   int read_bytes;
   char readbuf[80];
   char end_str[5];
   printf("FIFO_CLIENT: Send messages, infinitely, to end enter \"end\"\n");
   fd = open(FIFO_FILE, O_CREAT|O_RDWR);
   strcpy(end_str, "end");

   while (1) {
      printf("Enter string: ");
      fgets(readbuf, sizeof(readbuf), stdin);
      stringlen = strlen(readbuf);
      readbuf[stringlen - 1] = '\0';
      end_process = strcmp(readbuf, end_str);

      //printf("end_process is %d\n", end_process);
      if (end_process != 0) {
         write(fd, readbuf, strlen(readbuf));
         printf("FIFOCLIENT: Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen
         read_bytes = read(fd, readbuf, sizeof(readbuf));
         readbuf[read_bytes] = '\0';
         printf("FIFOCLIENT: Received string: \"%s\" and length is %d\n", readbuf, (int)strlen(re
      } else {
         write(fd, readbuf, strlen(readbuf));
         printf("FIFOCLIENT: Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen
         close(fd);
         break;
      }
   }
   return 0;
}
```

Compilation and Execution Steps

```
FIFO_CLIENT: Send messages, infinitely, to end enter "end"
Enter string: LINUX IPCs
FIFOCLIENT: Sent string: "LINUX IPCs" and string length is 10
FIFOCLIENT: Received string: "sCPI XUNIL" and length is 10
Enter string: Inter Process Communication
FIFOCLIENT: Sent string: "Inter Process Communication" and string length is 27
FIFOCLIENT: Received string: "noitacinummoC ssecorP retnI" and length is 27
Enter string: end
FIFOCLIENT: Sent string: "end" and string length is 3
```

Previous Page Next Page