2022PMN4226
Mayank Lakhotia

Practical – 3

Implement a "pipe ()" system call for the following –

pipe() is a Linux system function. The pipe() system function is used to open file descriptors, which are used to communicate between different Linux processes. In short, the pipe() function is used for inter-process communication in Linux. In this article, I am going to show you how to use the pipe() system function in Linux. So, let's get started.

All About pipe() Function:

The syntax of the pipe() function is:

```
int pipe(int pipefd[2]);
```

Here, the pipe() function creates a unidirectional data channel for inter-process communication. You pass in an int (Integer) type array pipefd consisting of 2 array element to the function pipe(). Then the pipe() function creates two file descriptors in the pipefd array.

The first element of the pipefd array, pipefd[0] is used for reading data from the pipe.

The second element of the pipefd array, pipefd[1] is used for writing data to the pipe.

On success, the pipe() function returns 0. If an error occurs during pipe initialization, then the pipe() function returns -1.

The pipe() function is defined in the header unistd.h. In order to use the pipe() function in your C program, you must include the header unistd.h as follows:

#include <unistd.h>

For more information on the pipe() system function, check the man page of pipe() with the following command:

1) Perform inter-process communication between a Parent and Child process.

Source code: -

```
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
int main()
{
   int pipefds[2];
   int returnstatus;
   int pid;
   char writemessages[2][20] = {"Hi", "Hello"};
   char readmessage[20];
   returnstatus = pipe(pipefds);
```

```
    if (returnstatus == -1)
    {
        printf("Unable to create pipe\n");
        return 1;
    }
    pid = fork();
    bool cont = true;
    // bool terminate = false;
    int cnt = 0;
    while (cont)
    {
        if (cnt % 2 == 0)
        {
            // Child process
            if (pid == 0)
            {
                printf("Child process what you want to do \n 1. Write \n 2. Read \n 3. Quit");
                int choice;
                scanf("%d", &choice);
                switch (choice)
                {
                // Write
                case 1:
                    printf("Child Writing to pipe - Message 1 is %s\n",
```

```
writemessages[0]);
                write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
                printf("Child Writing to pipe - Message 2 is %s\n",
writemessages[1]);
                write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
                break;
            // Read
            case 2:
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Child Reading from pipe – Message 1 is %s\n",
readmessage);
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Child Reading from pipe – Message 2 is %s\n",
readmessage);
                break;
            case 3:
                printf("Child Stop Conversation, Conversation ended no one left to
chat with");
                cont = false;
                break;
            }
        }
        else
        { // Parent process
            printf("Parent process what you want to do \n 1. Write \n 2. Read \n 3.
Quit");
```

```
            int choice;
            scanf("%d", &choice);
            switch (choice)
            {
            // Write
            case 1:
                printf("Parent Writing to pipe - Message 1 is %s\n",
writemessages[0]);
                write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
                printf("Parent Writing to pipe - Message 2 is %s\n",
writemessages[1]);
                write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
                break;
            // Read
            case 2:
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Parent Reading from pipe – Message 1 is %s\n",
readmessage);
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Parent Reading from pipe – Message 2 is %s\n",
readmessage);
                break;
            case 3:
                printf("Parent Quiting Conversation, Conversation ended no one
left to chat with");
                cont = false;
```

```
            break;
        }
    }
}
else
{

    // Child process
    if (pid == 0)
    {
        printf("Child process what you want to do \n 1. Write \n 2. Read \n 3. Quit");
        int choice;
        scanf("%d", &choice);
        switch (choice)
        {
        // Write
        case 1:
            printf("Child Writing to pipe - Message 1 is %s\n", writemessages[0]);
            write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
            printf("Child Writing to pipe - Message 2 is %s\n", writemessages[1]);
            write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
            break;
```

```
            // Read
            case 2:
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Child Reading from pipe – Message 1 is %s\n",
readmessage);
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Child Reading from pipe – Message 2 is %s\n",
readmessage);
                break;
            case 3:
                printf("Child Quiting Conversation, Conversation ended no one left
to chat with");
                cont = false;
                break;
            }
        }
        else
        { // Parent process
            printf("Parent process what you want to do \n 1. Write \n 2. Read \n 3.
Quit ");
            int choice;
            scanf("%d", &choice);
            switch (choice)
            {
            // Write
            case 1:
```

```
                printf("Parent Process - Writing to pipe - Message 1 is %s\n",
writemessages[0]);

                write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

                printf("Parent Process - Writing to pipe - Message 2 is %s\n",
writemessages[1]);

                write(pipefds[1], writemessages[1], sizeof(writemessages[1]));

                break;

            // Read

            case 2:

                read(pipefds[0], readmessage, sizeof(readmessage));

                printf("Parent Process - Reading from pipe – Message 1 is %s\n",
readmessage);

                read(pipefds[0], readmessage, sizeof(readmessage));

                printf("Parent Process - Reading from pipe – Message 2 is %s\n",
readmessage);

                break;

            case 3:

                printf("Parent Quiting Conversation, Conversation ended no one
left to chat with");

                cont = false;

                break;

            }

        }

        cnt++;

    }

  }
```

```
    return 0;
}
```

Output: -

```
Parent process what you want to do
 1. Write
 2. Read
 3. QuitChild process what you want to do
 1. Write
 2. Read
 3. Quit1
Parent Writing to pipe - Message 1 is Hi
Parent Writing to pipe - Message 2 is Hello
Parent process what you want to do
 1. Write
 2. Read
 3. Quit2
Child Reading from pipe - Message 1 is Hi
Child Reading from pipe - Message 2 is Hello
Child process what you want to do
 1. Write
 2. Read
 3. Quit3
Child Stop Conversation, Conversation ended no one left to chat with
```

2)Perform inter-process communication between two child processes. This two-communication must continue till a specific key is pressed or a STOP message is sent by any one of the processes.

Source Code: -

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

 #define PIN_LENGTH 4

#define PIN_WAIT_INTERVAL 2
```

```
void getPIN(char pin[PIN_LENGTH + 1]) {
  srand(getpid() + getppid());
 pin[0] = 49 + rand() % 7;
 for(int i = 1; i < PIN_LENGTH; i++) {
    pin[i] = 48 + rand() % 7;
  }
 pin[PIN_LENGTH] = '\0';
}
 int main(void) {
  while(1) {
    int pipefds[2];
    char pin[PIN_LENGTH + 1];
    char buffer[PIN_LENGTH + 1];
pipe(pipefds);
  pid_t pid = fork();
  if(pid == 0) {
      getPIN(pin); // generate PIN
      close(pipefds[0]); // close read fd
      write(pipefds[1], pin, PIN_LENGTH + 1); // write PIN to pipe
  printf("Generating PIN in child and sending to parent...\n");
 sleep(PIN_WAIT_INTERVAL); // delaying PIN generation intentionally.
 exit(EXIT_SUCCESS);
    }
```

```
  if(pid > 0) {
      wait(NULL); // waiting for child to finish
close(pipefds[1]); // close write fd
      read(pipefds[0], buffer, PIN_LENGTH + 1); // read PIN from pipe
      close(pipefds[0]); // close read fd
      printf("Parent received PIN '%s' from child.\n\n", buffer);
    }
  }
return EXIT_SUCCESS;
}
/WE NNED TO PRESS CTRL+ C   TO STOP THE FORK PROCESS/
```

Output: -

```
mayank@mayank-virtual-machine:~/Desktop/parenttochildstop$ ./stop
Generating PIN in child and sending to parent...
Parent received PIN '3520' from child.

Generating PIN in child and sending to parent...
Parent received PIN '2410' from child.

Generating PIN in child and sending to parent...
Parent received PIN '2422' from child.

Generating PIN in child and sending to parent...
Parent received PIN '4323' from child.

Generating PIN in child and sending to parent...
Parent received PIN '6443' from child.

Generating PIN in child and sending to parent...
Parent received PIN '2606' from child.

Generating PIN in child and sending to parent...
Parent received PIN '3413' from child.

Generating PIN in child and sending to parent...
Parent received PIN '6166' from child.

Generating PIN in child and sending to parent...
^C
mayank@mayank-virtual-machine:~/Desktop/parenttochildstop$
```