Bash Pipeline

P1-pre2

Bash pipeline syntax

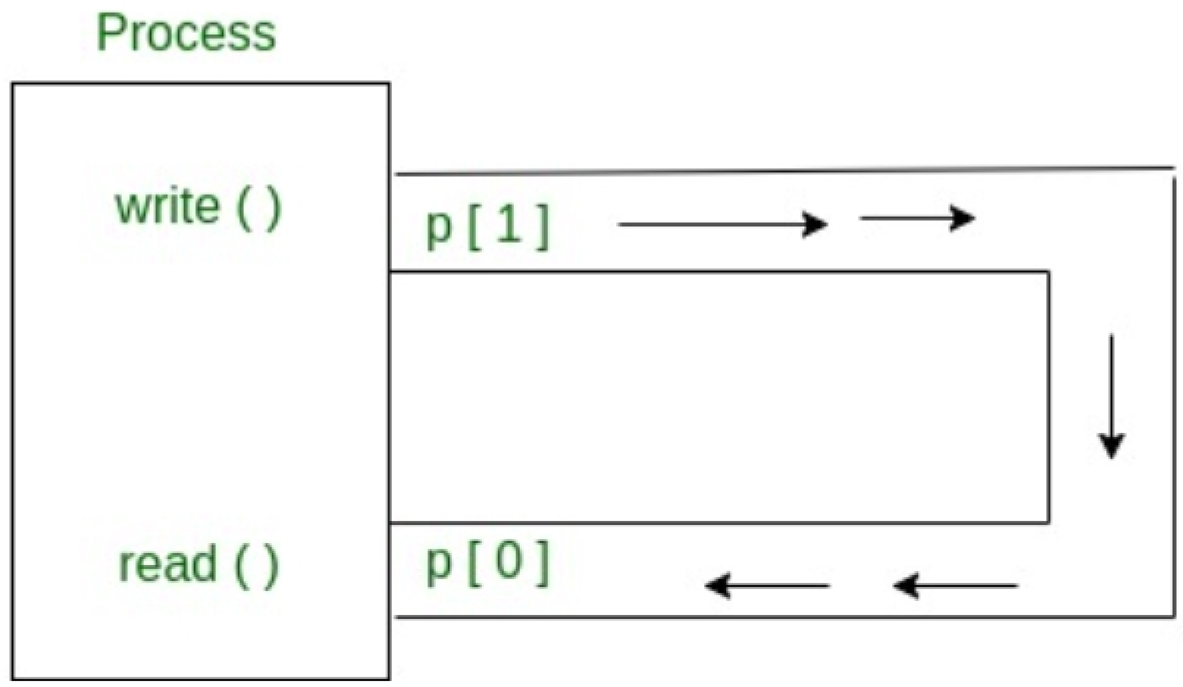
Syntax: [command1] | [command2] | … | [command_n]

The pipeline “|” is used for transferring the output of the previous command to the next command’s input.

For example, “ls | cat”, which will connect the output of “ls”, which is all the files in the current directory to the “cat” command’s input and output to the stdout by the “cat” command.

Pipe() system call

- Pipe is one-way communication only, we can use a pipe such that one process write to the pipe, and the other process reads from the pipe.
- If a process tries to read before something is written to the pipe, the process is suspended until something is written.



Pipe() continued

Function prototype:

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

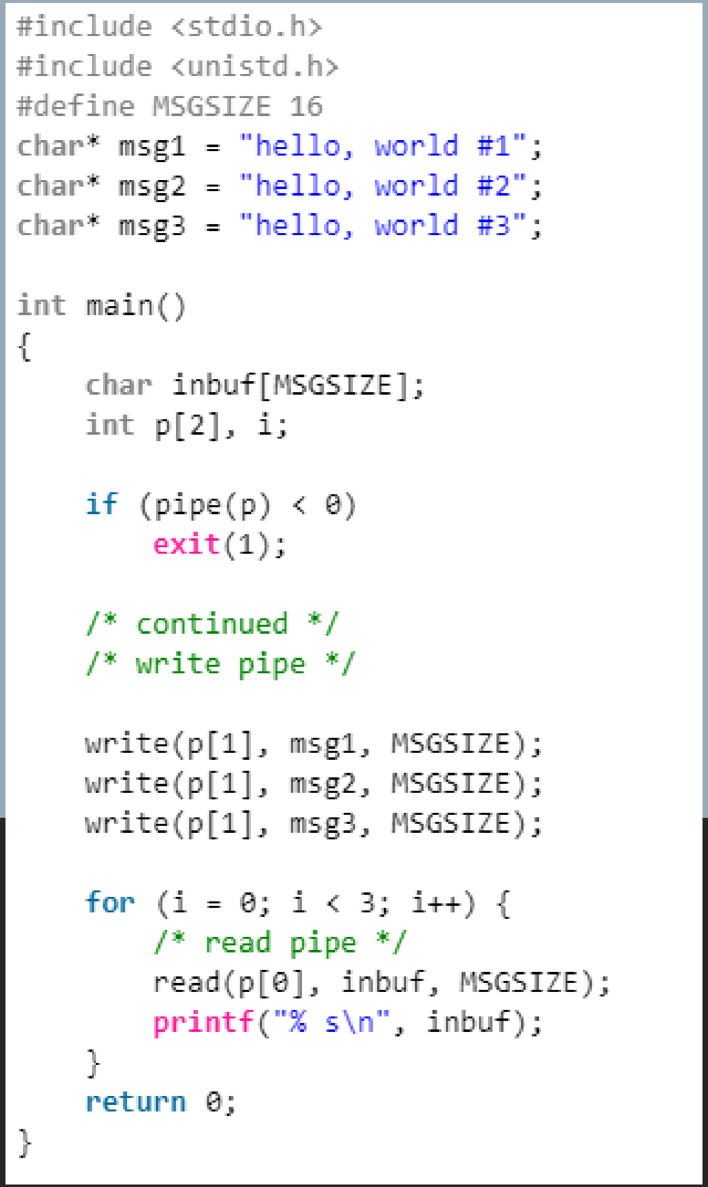
Where fd[0] is the file descriptor for the read end of the pipe, fd[1] is the file descriptor for the write end of the pipe, like the previous picture has shown.
If a pipe is created successfully, it will return 0, if failed, it will return -1.

Pipe behave like a queue structure, which serves as FIFO (first in first out).

Pipe() continued

When we use fork() in any process, file descriptors remain open across child process and also parent process. If we call fork after creating a pipe, then the parent and child can communicate via the pipe, in other words, share the pipe.

Pipe() continued



```c
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
    char inbuf[MSGSIZE];
    int p[2], i;

    if (pipe(p) < 0)
        exit(1);

    /* continued */
    /* write pipe */

    write(p[1], msg1, MSGSIZE);
    write(p[1], msg2, MSGSIZE);
    write(p[1], msg3, MSGSIZE);

    for (i = 0; i < 3; i++) {
        /* read pipe */
        read(p[0], inbuf, MSGSIZE);
        printf("% s\n", inbuf);
    }
    return 0;
}
```

The code on the left creates a pipeline with the code “pipe(p)”, then it writes the three messages defined above into the write end of the pipeline.
In the end, it takes a loop to output the buffer stored in the pipeline, which will output the three messages one by one.
This is because the reading process of the pipeline can only read a little number of characters at a same time.

Thank you