Bahir Dar Univerisity

Bahir Dar Institute Of Technology
Faculity Of Computing
Deparrtment Of Software Engineering
Operating System And System Programming
Individual Assignment 2
Title : Splice system call

ssize_t splice(int fd_in, loff_t *off_in, int fd_out, loff_t *off_out, size_t len, unsigned int flags)

Name: **ID**

Fentahun Mengie **1306919**

Submitted to:InstructorWendimu Baye
Submission date: 17-11-2014 E.C

Introduction

ssize_t splice(int fd_in, loff_t *off_in, int fd_out, loff_t *off_out, size_t len, unsigned int flags)

In this docment we will see what is splice system call?, why we use splice system call?, how we use system call and its parameters and its flags. Finally we will see code implementation of the splice system call.

splice() is a Linux-specific system call that moves data between two file descriptors without copying between kernel address space and user address space. It moves data between a file descriptor and a pipe without a round trip to user space.

It transfers up to len bytes of data from the file descriptor fd_in to the file descriptor fd_out, where one of the file descriptors must refer to a pipe.

With splice(), data can be transferred from one file descriptor to another without requiring any copies from user space into kernel space, which is typically necessary **to maintain system security** and to **provide processes with a straightforward interface** for reading and writing to files. splice() works by using the pipe buffer. A pipe buffer is an in-kernel memory buffer that is opaque to the user space process. A user process can splice the contents of a source file into this pipe buffer, then splice the pipe buffer into the destination file, all without moving any data through userspace.

Parameters and Flags

- **fd_in** and **fd_out** represent the input and output file descriptors, respectively, and one of these two file descriptors must point to a pipeline device, which is a less friendly restriction.
- **off_in** and **off_out** are pointers to the offsets of **fd_in** and **fd_out** respectively, indicating where the kernel reads and writes data from.

  The following semantics apply for fd_in and off_in:

  If fd_in refers to a pipe, then off_in must be NULL.

  If fd_in does not refer to a pipe and off_in is NULL, then bytes are read from fd_in starting from the file offset, and the file offset is adjusted appropriately.

  If fd_in does not refer to a pipe and off_in is not NULL, then off_in must point to a buffer which specifies the starting offset from which bytes will be read from fd_in; in this case, the file offset of fd_in is not changed.

  Analogous statements apply for fd_out and off_out.

- **len** indicates the number of bytes the call wishes to transfer

- **Flag** is the system call's flag option bitmask, which sets the behavior of the system call, and is a combination of 0 or more of the following values via the 'or' operation.

SPLICE_F_MOVE

SPLICE_F_NONBLOCK

SPLICE_F_MORE

SPLICE_F_GIFT

- **SPLICE_F_MOVE**: instructs splice() to try to just move memory pages instead of copying them; setting this value does not necessarily mean that memory pages will not be copied; whether they are copied or moved depends on whether the kernel can move memory pages from the pipeline, or whether the memory pages in the pipeline are intact. pages may be copied if the kernel can not move the pages from the pipe, or if the pipe buffers don't refer to full pages. The initial implementation of this flag had a lot of bugs, so it has been in place since Linux version 2.6.21, but it has been retained because it may be reimplemented in a future version.

- **SPLICE_F_NONBLOCK**: instructs splice() not to block I/O, i.e. makes the splice() call a non-blocking call that can be used to implement asynchronous data transfers, but note that it is also best to pre-mark the two file descriptors for data transfers as non-blocking I/O with O_NONBLOCK, Otherwise, the splice() call may still be blocked.

- **SPLICE_F_MORE**: informs the kernel that more data will be transferred with the next splice() system call, this flag is useful for scenarios where the output/fd_out/ side is a socket.

- **SPLICE_F_GIFT**: Unused for splice()

RETURN VALUE

Upon successful completion, splice() returns the number of bytes spliced to or from the pipe. A return value of 0 means end of input.

If fd_in refers to a pipe, then this means that there was no data to transfer, and it would not make sense to block because there are no writers connected to the write end of the pipe.

On error, splice() returns -1 and errno is set to indicate the error.

ERRORS

EAGAIN SPLICE_F_NONBLOCK was specified in flags or one of the file descriptors had been marked as nonblocking (O_NONBLOCK), and the operation would block.

EBADF One or both file descriptors are not valid, or do not have proper read-write mode.

EINVAL

- ✓ The target filesystem doesn't support splicing.
- ✓ The target file is opened in append mode
- ✓ Neither of the file descriptors refers to a pipe.
- ✓ An offset was given for nonseekable device (e.g., a pipe).
- ✓ fd_in and fd_out refer to the same pipe.

ENOMEM Out of memory.

ESPIPE Either off_in or off_out was not NULL, but the corresponding file descriptor refers to a pipe.

code implementation

```
#define _GNU_SOURCE
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(int argc, char **argv) {
  if(argc < 2) {
    fputs("it is copied succesfully\n", stderr);
    return 1;
  }

  FILE *handle = fopen(argv[1], "r");
  if(!handle) {
    perror("Error opening file:");
    goto error;
  }

  int handle_fd = fileno(handle);

  int filedes[2];
  if(pipe(filedes)) {
    perror("Error creating pipe:");
    goto error;
  }

  while(1) {
    ssize_t rc = splice(handle_fd, NULL, filedes[1], NULL, BUFSIZ, 0);

    if(rc == -1) {
      // Error occurred
      perror("Error copying data:");
      goto error;
    } else if(rc == 0) {
      break;
```

```
    }

    splice(filedes[0], NULL, STDOUT_FILENO, NULL, BUFSIZ, 0);
  }

  return 0;

error:
  if(fclose(handle)) {
    perror("Error closing file:");
  }

  return 1;
}
```

Here is the output. As we can see from below it is copied
I hav run it in online c compiler