BAHIR DAR UNIVERSITY

FACULTY OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

OPERATING SYSTEM AND SYSTEM PROGRAMMING

INDIVIDUAL ASSIGNMENT : SYSTEM PROGRAMMING

SYSTEM CALL: ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count)

NAME : MELAT TESFAYE

ID-NUMBER : 1307207

INSTRUCTORS NAME – WENDIMU B.( )

DATE OF SUBMISSION – 17/11/2014E.C

WHAT/ WHY AND HOW IS THIS SYSTEM CALL?

ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count)

SSIZE_T :

ssize_t is the same as size_t(The datatype size_t is **unsigned integral type**. It represents the size of any object in bytes and returned by sizeof operator. It is used for array indexing and counting. It can never be negative. The return type of strcspn, strlen functions is size_t.), but is a signed type - read ssize_t as “signed size_t”. ssize_t is able to represent the number -1, which is returned by several system calls and library functions as a way to indicate error.

SENDFILE():

sendfile() copies data between two file descripters within kernel space.It is Linux-specific. It tells the kernel to do zero-copy I/O from a file to a socket. (Note that it only works when the source fd is a file and the destination is a socket. sendfile() can not only reduce the number of switching but also the number of copies. Its library is Standard C Library (libc, -lc) main things contained in the sendfile function are, including flags and system calls:

```
int sendfile(int fd, int s, off_t offset, size_t nbytes, struct
sf_hdtr *hdtr, off_t *sbytes,int flags);
```

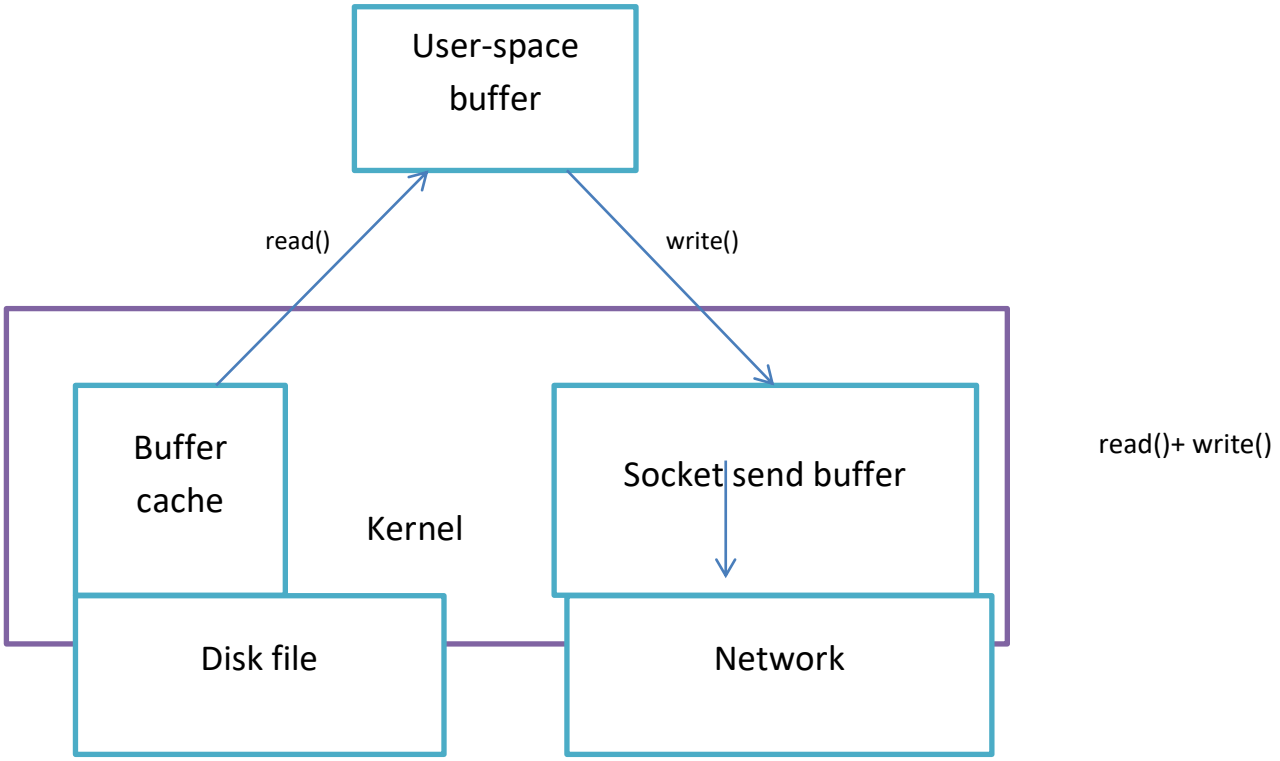
implementation of **sendfile**() is "zero-copy", meaning that it has been optimized so that copying of the file data is avoided.
copies data from in_fd to out_fd, starting at an offset of off bytes and continuing for a length of len bytes.

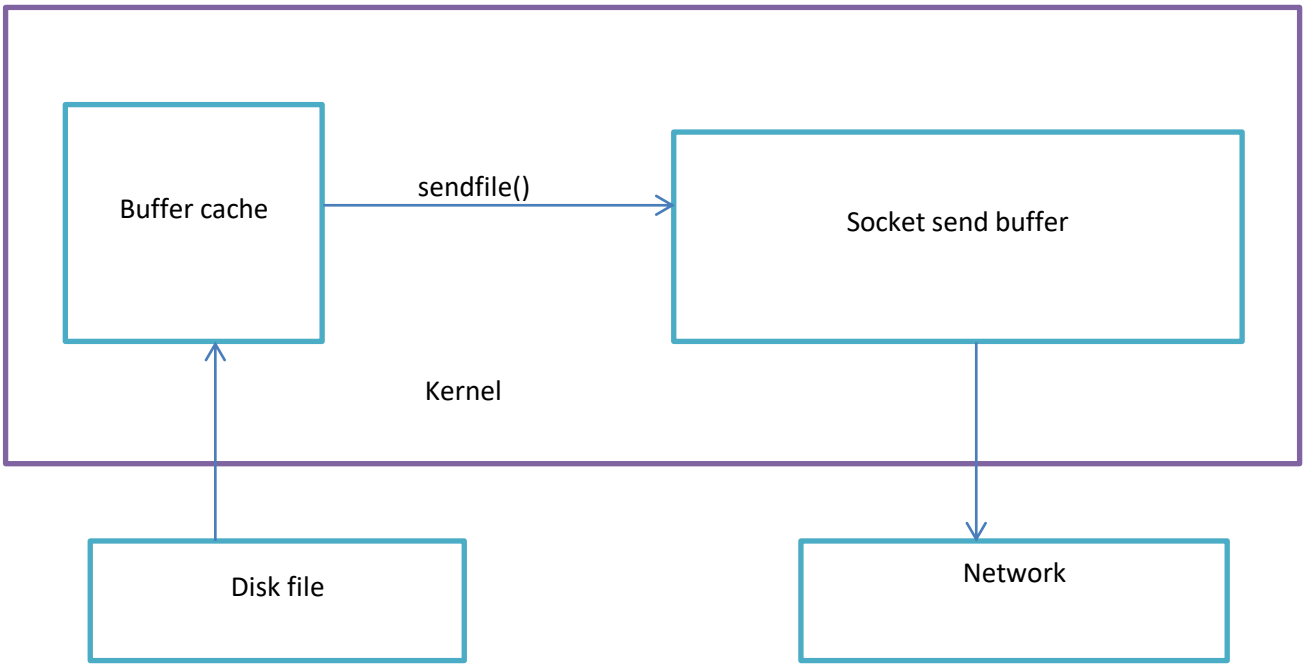
The offset argument specifies where to begin in the file. Should offset fall beyond the end of file, the system will return success and report 0 bytes sent. If the off argument is null, data is read from in_fd starting at its own file offset, and the file offset of in_fd is updated to the offset of the byte following the last byte that was read.

The sendfile() is created to solve or to eliminate the wasteful application process that is created by the read() and write() functions . When applications call sendfile(), the file contents are transferred directly to the socket, without passing through user space as shown(zero-copy transfer)#include <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd, off_t*offset, size_t count);

E : //Retuns number of bytes ransfered , or -1 on error


User-space buffer
read()
write()
Buffer cache
Kernel
Socket send buffer
read()+ write()
Disk file
Network


Buffer cache
sendfile()
Socket send buffer
Kernel
Disk file
Network

sendfile()

Servers frequently needed to transfer unaltered contents of a disk file through a socket and the read and write functions where used in a loop to achieve that but as we consider larger files this technique appears to be inefficient, 2 steps were taken which are wasteful if the application doesn't perform processing of the file contents before I transmits them. And the send file() system call is designed to eradicate this inefficiency.

E: errors

- ✓ [EAGAIN] The socket is marked for non-blocking I/O and not all data was sent due
  - to the socket buffer being filled. If specified, the number of bytes
  - successfully sent will be returned in *sbytes.
- ✓ [EBADF] The fd argument is not a valid file descriptor.
- ✓ [EBADF] The s argument is not a valid socket descriptor.
- ✓ [EBUSY] Completing the entire transfer would have required disk I/O, so it was
  - aborted. Partial data may have been sent. (This error can only occur
  - when SF_NODISKIO is specified.)
- ✓ [EFAULT] An invalid address was specified for an argument.
- ✓ [EINTR] A signal interrupted **sendfile**() before it could be completed. If
  - specified, the number of bytes successfully sent will be returned in
  - *sbytes.
- ✓ [EINVAL] The fd argument is not a regular file.
- ✓ [EINVAL] The s argument is not a SOCK_STREAM type socket.
- ✓ [EINVAL] The offset argument is negative.
- ✓ [EIO] An error occurred while reading from fd.
- ✓ [ENOTCONN] The s argument points to an unconnected socket.
- ✓ [ENOTSOCK] The s argument is not a socket.
- ✓ [EOPNOTSUPP] The file system for descriptor fd does not support **sendfile**().
- ✓ [EPIPE] The socket peer has closed the connection.

What are the parameters and flags

```
int sendfile(int fd, int s, off_t offset, size_t nbytes, struct sf_hdtr *hdtr, off_t *sbytes,int flags);
```

The sendfile() system call transfers bytes from the file referred to by the descriptor in_fd to the file referred to by the descriptor out_fd. The out_fd descriptor must refer to a socket. The in_fd argument must refer to a file to which mmap() can be applied; in practice, this usually means a regular file. This somewhat restricts the use of sendfile(). We can use it to pass data from a file to a socket, but not vice versa. And we can't use sendfile() to pass data directly from one socket to another. Performance benefits could also be obtained if sendfile() could be used to transfer bytes between two regular files. On Linux 2.4 and earlier, out_fd could refer to a regular file. Some reworking of the underlying implementation meant that this possibility disappeared in the 2.6 kernel. However, this feature may be reinstated in a future kernel version. If offset is not NULL, then it should point to an off_t value that specifies the starting file offset from which bytes should be transferred from in_fd. This is a value-result argument. On return, it contains the offset of the next byte following the last byte that was transferred from in_fd. In this case, sendfile() doesn't change the file offset for in_fd. If offset is NULL, then bytes are transferred from in_fd starting at the current file offset, and the file offset is updated to reflect the number of bytes transferred.
The count argument specifies the number of bytes to be transferred. If end-offile is encountered before count bytes are transferred, only the available bytes are transferred. On success, sendfile() returns the number of bytes actually transferred. SUSv3 doesn't specify sendfile(). Versions of sendfile() are available on some other UNIX implementations, but the argument list is typically different from the version on Linux. Starting with kernel 2.6.17, Linux provides three new (nonstandard) system calls—splice(), vmsplice(), and tee()—that provide a superset of the functionality of sendfile(). See the manual pages for details.

On Linux, file descriptors can be true files or devices, such as a network socket. The sendfile implementation currently requires that the input file descriptor correspond to a true file or some device which supports mmap. This means, for example, it cannot be a network socket. The output file descriptor can correspond to a socket, and this is usually the case when it is used.

Code implementation of the system call, its parameters and flags

Source code #1

```
SYSCALL_DEFINE4(sendfile, int, out_fd, int, in_fd, off_t __user *, offset, size_t, count)
{
loff_t pos;
```

```
off_t off;
ssize_t ret;
if (offset) {
if (unlikely(get_user(off, offset)))
return -EFAULT;
pos = off;
ret = do_sendfile(out_fd, in_fd, &pos, count, MAX_NON_LFS);
if (unlikely(put_user(pos, offset)))
return -EFAULT;
return ret;
}
return do_sendfile(out_fd, in_fd, NULL, count, 0);
}
```

Source code #2

```
#include <sys/sendfile.h>

bool WriteFileDescriptor(int fd, span<const uint8_t> data) { .

ssize_t bytes_written_total = 0; // Allow for partial writes
ssize_t size = checked_cast<ssize_t>(data.size());
for (ssize_t bytes_written_partial = 0; bytes_written_total < size;
bytes_written_total += bytes_written_partial) {
bytes_written_partial = HANDLE_EINTR(write(
fd, data.data() + bytes_written_total, size - bytes_written_total));
if (bytes_written_partial < 0)
return false;
}
```

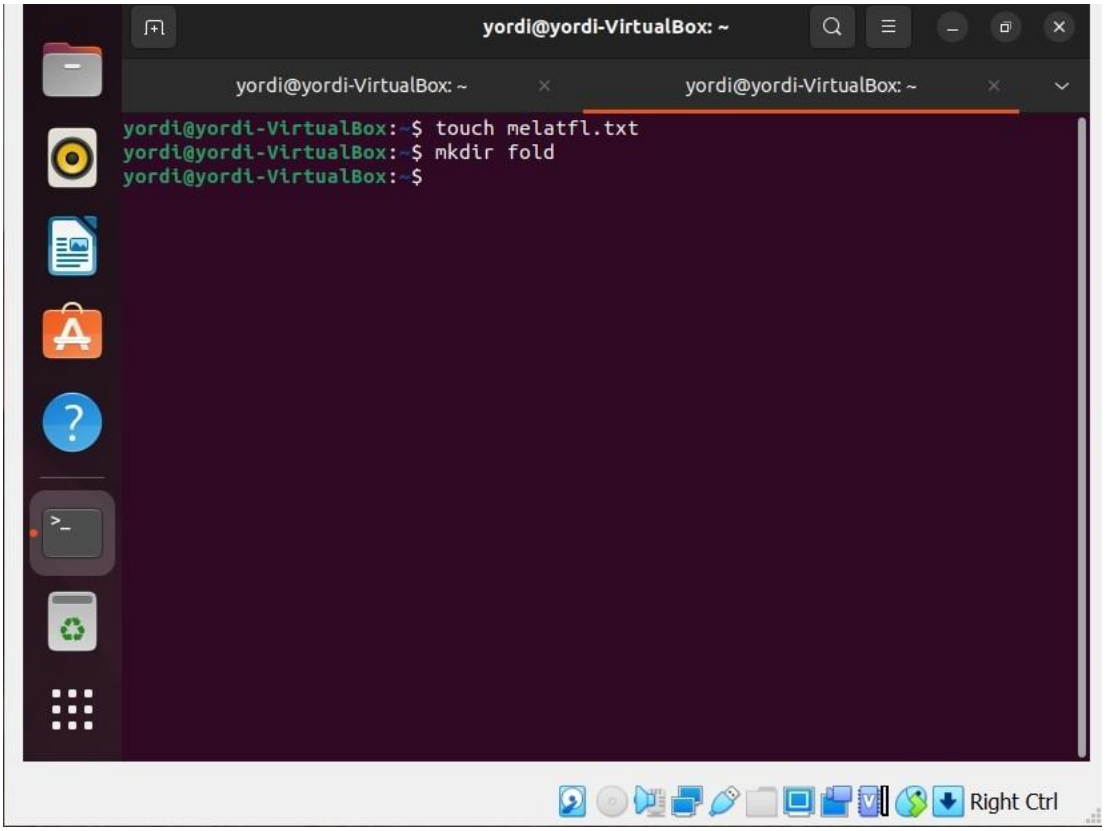
Source code #3

```
#include <sys/cdefs.h>
#include <sys/types.h>
#if defined(__USE_FILE_OFFSET64)
#if __ANDROID_API__ >= 21
ssize_t sendfile(int __out_fd, int __in_fd, off_t* __offset, size_t __count)
__RENAME(sendfile64) __INTRODUCED_IN(21);
#endif /* __ANDROID_API__ >= 21 */
#else
ssize_t sendfile(int __out_fd, int __in_fd, off_t* __offset, size_t __count);
#endif
#if __ANDROID_API__ >= 21
ssize_t sendfile64(int __out_fd, int __in_fd, off64_t* __offset, size_t __count)
__INTRODUCED_IN(21);
#endif
```

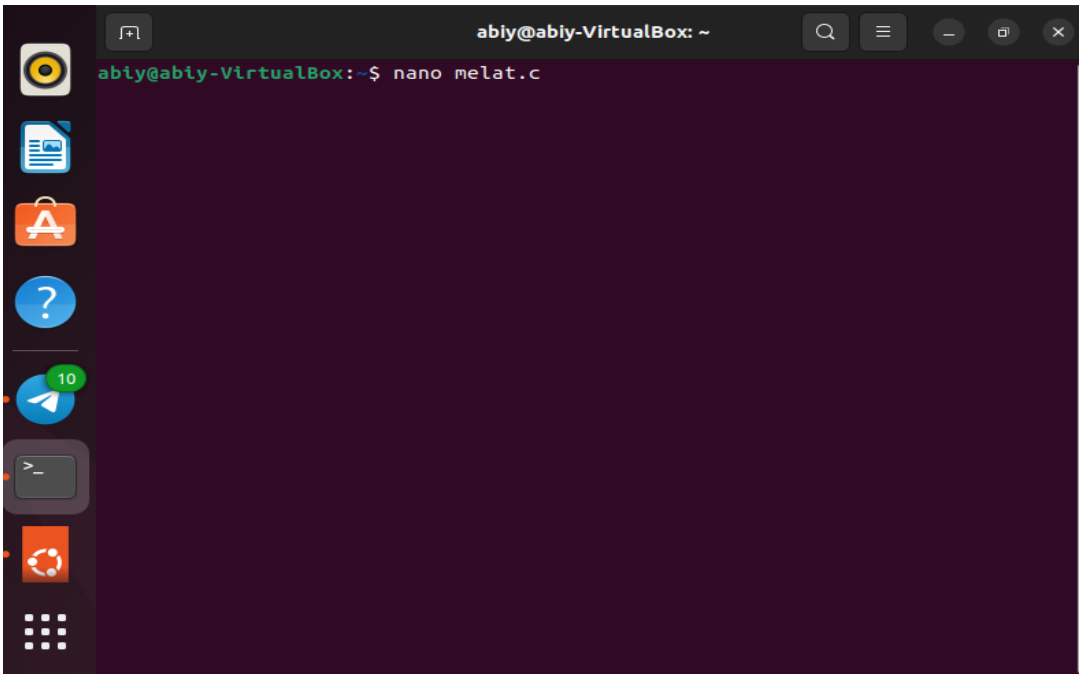
Example

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
int main(int argc, char **argv) {
    int src;                /* file descriptor for source file */
    int dest;               /* file descriptor for destination file */
    struct stat stat_buf;   /* hold information about input file */
    off_t offset = 0;       /* byte offset used by sendfile */

    /* check that source file exists and can be opened */
    src = open(argv[1], O_RDONLY);

    /* get size and permissions of the source file */
    fstat(src, &stat_buf);

    /* open destination file */
    dest = open(argv[2], O_WRONLY|O_CREAT, stat_buf.st_mode);

    /* copy file using sendfile */
    sendfile (dest, src, &offset, stat_buf.st_size);

    /* clean up and exit */
    close(dest);
    close(src);
}
```

/**flags used O_RDONLY, O_WRONLY, O_CREAT**/



File must be created in order to implement the sendfile system call

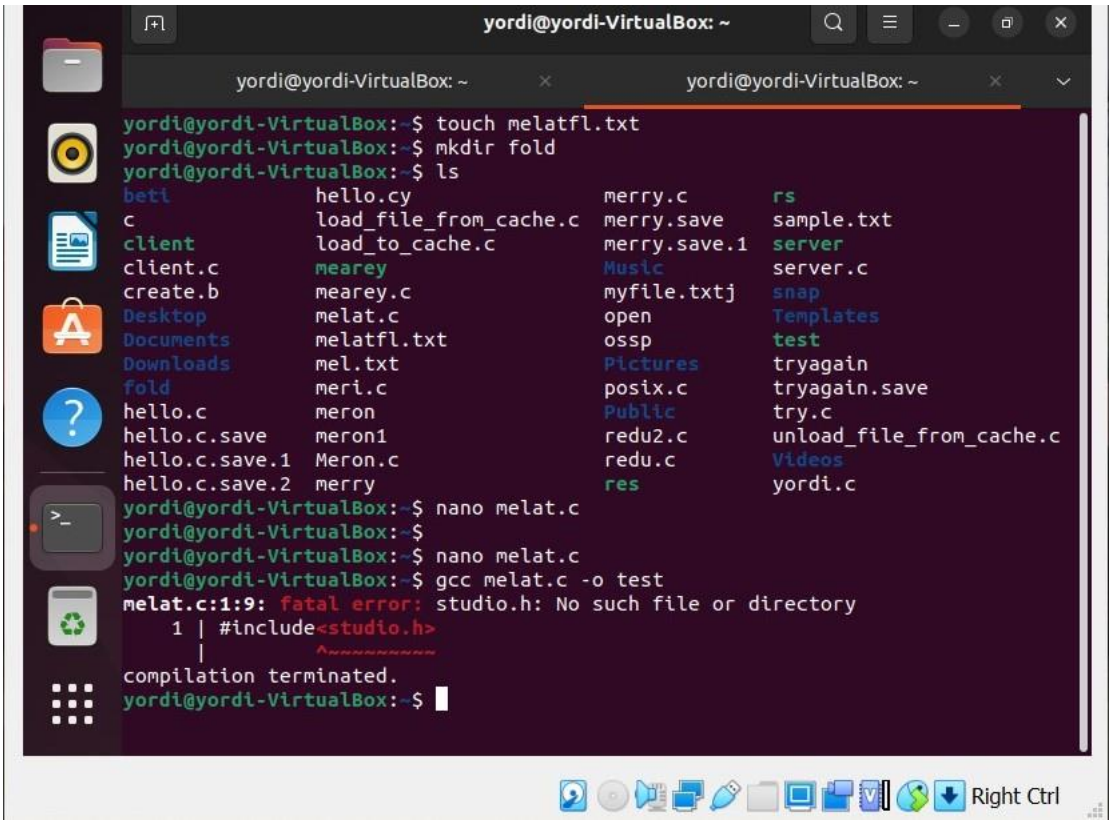


```
yordi@yordi-VirtualBox: ~
yordi@yordi-VirtualBox: ~
yordi@yordi-VirtualBox: ~
yordi@yordi-VirtualBox:~$ touch melatfl.txt
yordi@yordi-VirtualBox:~$ mkdir fold
yordi@yordi-VirtualBox:~$ ls
beti                hello.cy                  merry.c        rs
c                   load_file_from_cache.c    merry.save     sample.txt
client              load_to_cache.c           merry.save.1   server
client.c            mearey                    Music          server.c
create.b            mearey.c                  myfile.txtj    snap
Desktop             melat.c                   open           Templates
Documents           melatfl.txt               ossp           test
Downloads           mel.txt                   Pictures       tryagain
fold                meri.c                    posix.c        tryagain.save
hello.c             meron                     Public         try.c
hello.c.save        meron1                    redu2.c        unload_file_from_cache.c
hello.c.save.1      Meron.c                   redu.c         Videos
hello.c.save.2      merry                     res            yordi.c
yordi@yordi-VirtualBox:~$ nano melat.c
yordi@yordi-VirtualBox:~$
yordi@yordi-VirtualBox:~$
```

Right Ctrl

```
yordi@yordi-VirtualBox: ~
yordi@yordi-VirtualBox: ~
yordi@yordi-VirtualBox: ~
  GNU nano 6.2                    melat.c *
#include<studio.h>
#include<errno.h>
#include<stdlib.h>
#include<fcntl.h>
int main(int argc, char**argv)
{
int src;//file descriptor for source file
int dest;//file descriptor for destination file
struct stat melatfl;//hold information about the file
off_t offset=0;//byte offset used by sendfile

//check that the source file exists and can be opened
src = open(argv[1], O_RDONLY);
//gettin size and permission of the source file
fstat(src,&melatfl);
//open destination file
dest = open(argv[2], O_WRONLY|O_CREAT,melatfl.st_mode);
//copy file using sendfile()
sendfile(dest,src,&offset,mel.st_size);
//cleanup and exit
close(dest);
close(src);

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify
```

Right Ctrl



REFFERENCE

↑ Michaelkerrisk 2010 william Pollock no starch press, SanFransisco

↑ Vaughan, G.V., Elliston, B., Tromey, T., and Taylor, I.L. 2000. GNU Autoconf, Automake, and Libtool. New Riders, Indianapolis, Indiana.
↑ http://man.freetechsecrets.com/netstat.1.html
↑ https://source.chromium.org/chromium/chromium/src/+/main:third_party/android_ndk/toolchains/llvm/prebuilt/linux-x86_64/lib64/clang/12.0.5/include/sanitizer/linux_syscall_hooks.h;l=2177?q=sendfile%20(int%20out_fd)&ss=chromium
↑ https://man7.org/linux/man-pages/man2/sendfile.2.html
↑ https://techterms.com/definition/flag
↑ https://linuxhint.com/c-language-o_donly-o_wrongly-and-o_rdwr-flags/