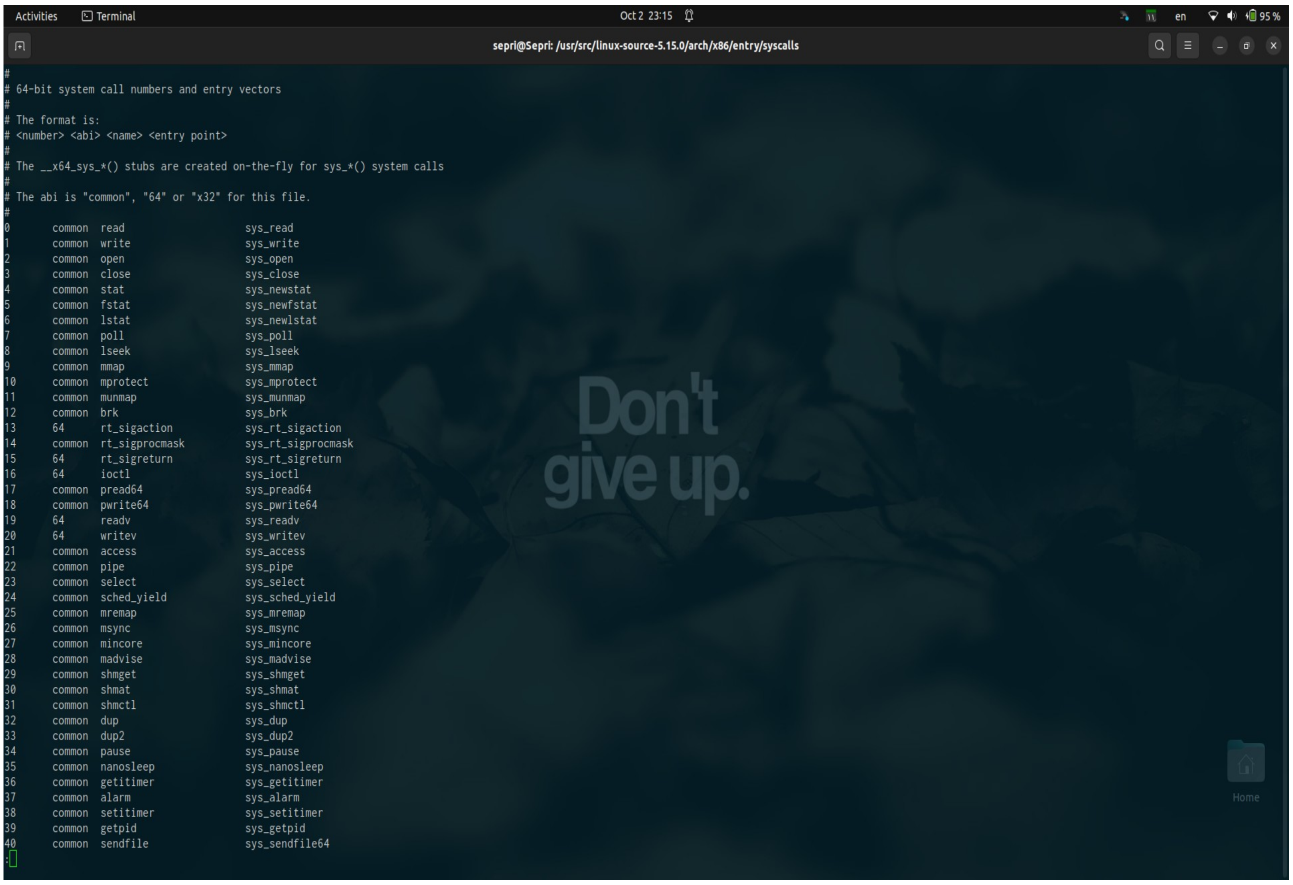
Sepehr Ebadi 9933243

```
sudo apt update
sudo apt install linux-source
cd /usr/src
sudo tar -xjf linux-source-*.tar.bz2
cd linux-source-5.15.0/
cd arch/x86/entry/syscalls/
cat syscall_64.tbl | less
```


Activities
Terminal
Oct 2 23:15
sepri@Sepri: /usr/src/linux-source-5.15.0/arch/x86/entry/syscalls
Don't give up.
Home

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
8       common  lseek                   sys_lseek
9       common  mmap                    sys_mmap
10      common  mprotect                sys_mprotect
11      common  munmap                  sys_munmap
12      common  brk                     sys_brk
13      64      rt_sigaction            sys_rt_sigaction
14      common  rt_sigprocmask          sys_rt_sigprocmask
15      64      rt_sigreturn            sys_rt_sigreturn
16      64      ioctl                   sys_ioctl
17      common  pread64                 sys_pread64
18      common  pwrite64                sys_pwrite64
19      64      readv                   sys_readv
20      64      writev                  sys_writev
21      common  access                  sys_access
22      common  pipe                    sys_pipe
23      common  select                  sys_select
24      common  sched_yield             sys_sched_yield
25      common  mremap                  sys_mremap
26      common  msync                   sys_msync
27      common  mincore                 sys_mincore
28      common  madvise                 sys_madvise
29      common  shmget                  sys_shmget
30      common  shmat                   sys_shmat
31      common  shmctl                  sys_shmctl
32      common  dup                     sys_dup
33      common  dup2                    sys_dup2
34      common  pause                   sys_pause
35      common  nanosleep               sys_nanosleep
36      common  getitimer               sys_getitimer
37      common  alarm                   sys_alarm
38      common  setitimer               sys_setitimer
39      common  getpid                  sys_getpid
40      common  sendfile                sys_sendfile64
:
```

Example 1: Interface of the `open` System Call

The `open` system call serves the purpose of opening a file.

User-Space Code Example:

```
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("/path/to/file", O_RDONLY);
    if (fd == -1) {
        // Handle the error appropriately
    }
    // Utilize the file descriptor 'fd' as needed
    close(fd);
    return 0;
}
```

```
Kernel-Side Implementation of the System Call:
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
    int fd = do_sys_open(AT_FDCWD, filename, flags, mode);
    return fd;
}
```

In this implementation, `SYSCALL_DEFINE3` is a macro that sets up a system call with three parameters.

Example 2: Interface of the `write` System Call

The `write` system call is used to send data to a specified file descriptor.

User-Space Code Example:

```
#include <unistd.h>

int main() {
    const char *message = "Hello, world!";
    int bytes_written = write(1, message, 13); // Writing to standard output (file descriptor 1)
    if (bytes_written == -1) {
        // Handle the error appropriately
    }
    return 0;
}
```

```
Kernel-Side Implementation of the System Call:
SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf, size_t, count)
{
    return ksys_write(fd, buf, count);
}
```

This implementation of the `write` system call is also defined with `SYSCALL_DEFINE3`, which indicates it takes three arguments: the file descriptor (`fd`), a buffer from user space (`buf`), and the number of bytes to write (`count`).