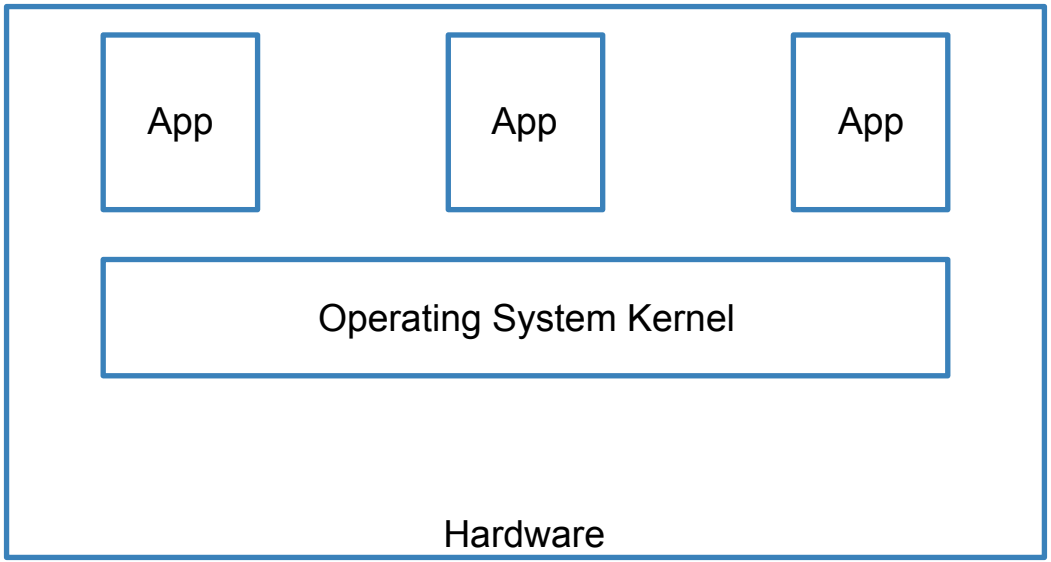
System Calls

Will Scott

Processes



System Calls

How do applications safely make requests to the Operating System?

Examples

read, write, open, close

fork, mount, mkdir, select,

init_module, sleep

Types of System Calls

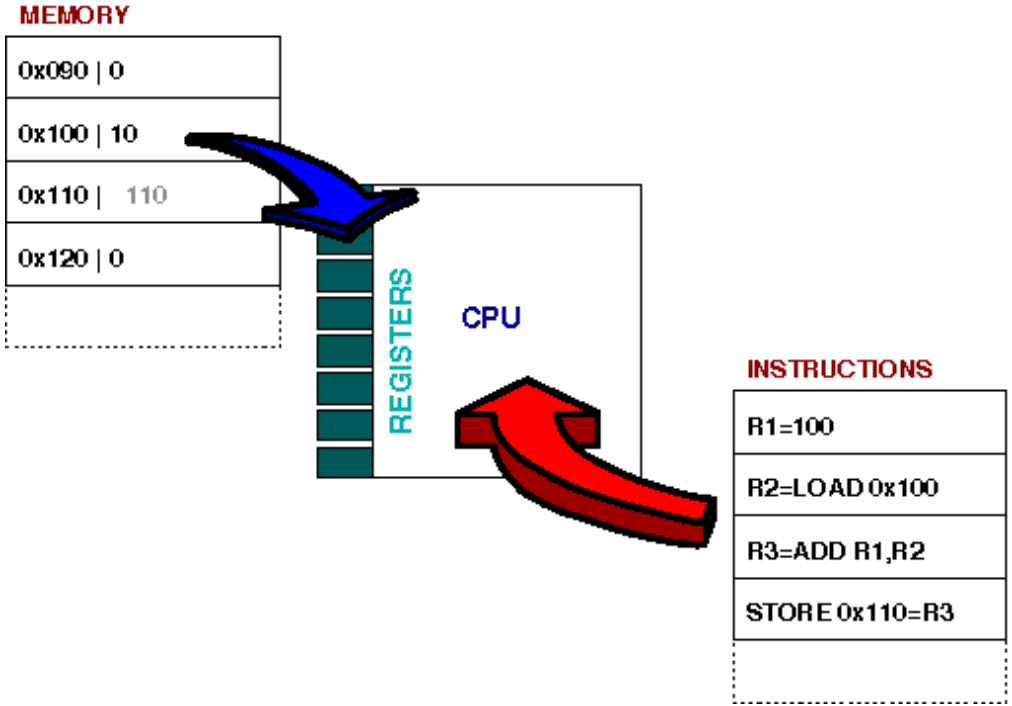
- Process Control
- Filesystem
- Device Manipulation
- Learn System Information
- Communication

System Calls

~350 system calls exist in linux.

http://syscalls.kernelgrok.com/

Computer Architecture



http://bottomupcs.sourceforge.net/csbu/c1453.htm

Hardware support

- Privileged instruction - potentially unsafe instructions are prohibited in user-mode
- Memory protection - memory outside the process can't be accessed
- Interrupts - the kernel needs some way to regain control

How is a system call made?

i386	int $0x80
x86_64	syscall
arm	swi 0x0

What does that do?

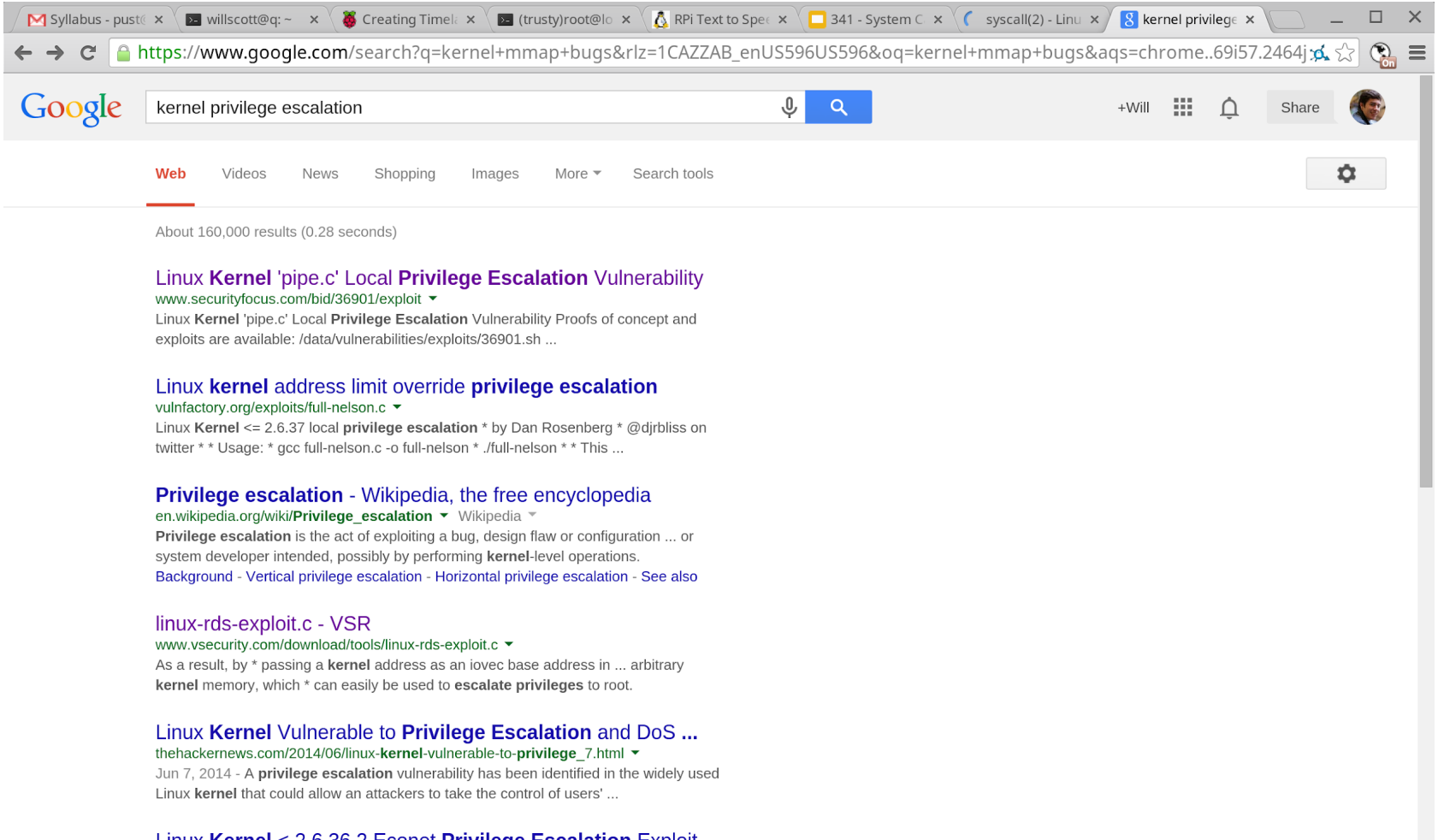
Acts like a hardware interrupt with code 80.

Jumps to interrupt handler.

1. Validates user input
2. Runs requested function
3. Returns to user

How do we validate input?

Is it possible for the kernel to implement 350 system calls without any bugs?


Syllabus - pust
willscott@q: ~
Creating Timel
(trusty)root@lo
RPi Text to Spe
341 - System C
syscall(2) - Linu
kernel privilege
https://www.google.com/search?q=kernel+mmap+bugs&rlz=1CAZZAB_enUS596US596&oq=kernel+mmap+bugs&aqs=chrome..69i57.2464j
Google
kernel privilege escalation
+Will
Share
Web
Videos
News
Shopping
Images
More
Search tools
About 160,000 results (0.28 seconds)
Linux Kernel 'pipe.c' Local Privilege Escalation Vulnerability
www.securityfocus.com/bid/36901/exploit
Linux Kernel 'pipe.c' Local Privilege Escalation Vulnerability Proofs of concept and exploits are available: /data/vulnerabilities/exploits/36901.sh ...
Linux kernel address limit override privilege escalation
vulnfactory.org/exploits/full-nelson.c
Linux Kernel <= 2.6.37 local privilege escalation * by Dan Rosenberg * @djrbliss on twitter * * Usage: * gcc full-nelson.c -o full-nelson * ./full-nelson * * This ...
Privilege escalation - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Privilege_escalation
Wikipedia
Privilege escalation is the act of exploiting a bug, design flaw or configuration ... or system developer intended, possibly by performing kernel-level operations.
Background - Vertical privilege escalation - Horizontal privilege escalation - See also
linux-rds-exploit.c - VSR
www.vsecurity.com/download/tools/linux-rds-exploit.c
As a result, by * passing a kernel address as an iovec base address in ... arbitrary kernel memory, which * can easily be used to escalate privileges to root.
Linux Kernel Vulnerable to Privilege Escalation and DoS ...
thehackernews.com/2014/06/linux-kernel-vulnerable-to-privilege_7.html
Jun 7, 2014 - A privilege escalation vulnerability has been identified in the widely used Linux kernel that could allow an attackers to take the control of users' ...

How do we return?

```
        popl %ebx;
        popl %ecx;
        popl %edx;
        popl %esi;
        popl %edi;
        popl %ebp;
        popl %eax;
1       popl %ds;
2       popl %es;
        addl $4,%esp;
3       iret;
```

Returns control to point of interruption by popping IP, CS and then the Flags from the stack and continues execution at this location. CPU exception interrupts will return to the instruction that cause the exception because the CS:IP placed on the stack during the interrupt is the address of the offending instruction.

System Call conventions

Where should work be done?

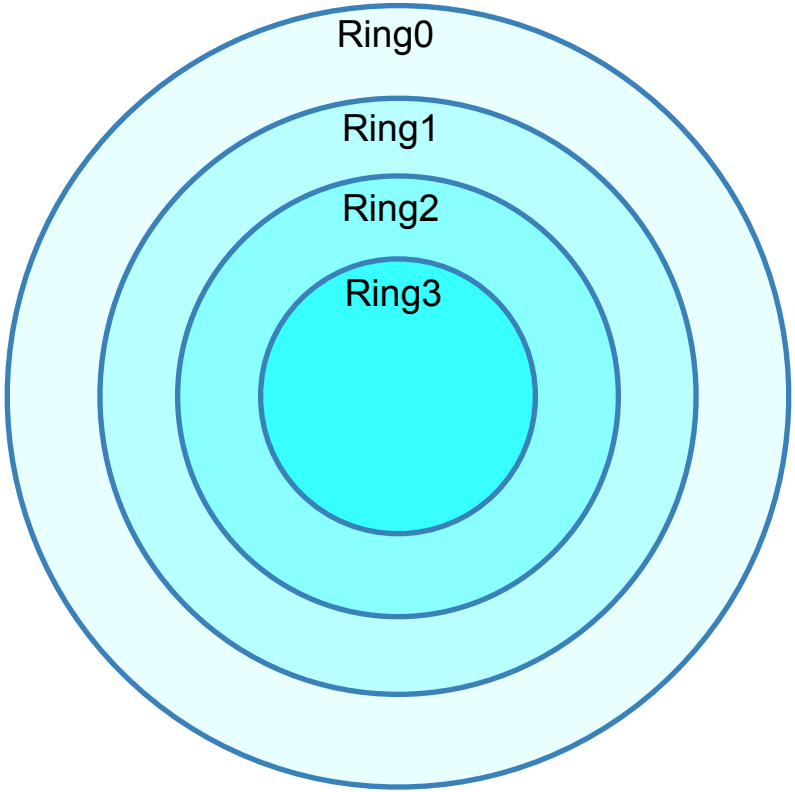
Layering permissions

- Principle of least privilege
- Units of execution privilege
- Performance vs Expressiveness

Layering permissions

- Alternatives
  - Microkernel
  - Unikernel
  - Exokernel

Layering Permissions



System Calls: Part 2

fopen(3)

- $ man fopen

```
FOPEN(3)                  Linux Programmer's Manual                 FOPEN(3)

NAME          top

       fopen, fdopen, freopen - stream open functions

SYNOPSIS          top

       #include <stdio.h>

       FILE *fopen(const char *pathname, const char *mode);

       FILE *fdopen(int fd, const char *mode);

       FILE *freopen(const char *pathname, const char *mode, FILE *stream);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fdopen(): _POSIX_C_SOURCE
```

fopen

```
42  FILE *
43  fopen(const char *file, const char *mode)
44  {
45          FILE *fp;
46          int f;
47          int flags, oflags;
48
49          if ((flags = __sflags(mode, &oflags)) == 0)
50                  return (NULL);
51          if ((fp = __sfp()) == NULL)
52                  return (NULL);
53          if ((f = open(file, oflags, DEFFILEMODE)) < 0) {
54                  fp->_flags = 0;                 /* release */
55                  return (NULL);
56          }
57          fp->_file = f;
58          fp->_flags = flags;
59          fp->_cookie = fp;
60          fp->_read = __sread;
61          fp->_write = __swrite;
62          fp->_seek = __sseek;
63          fp->_close = __sclose;
```

from BSD
libc/stdio/fopen.c v1.5

open

OPEN(2) Linux Programmer's Manual OPEN(2)

NAME top

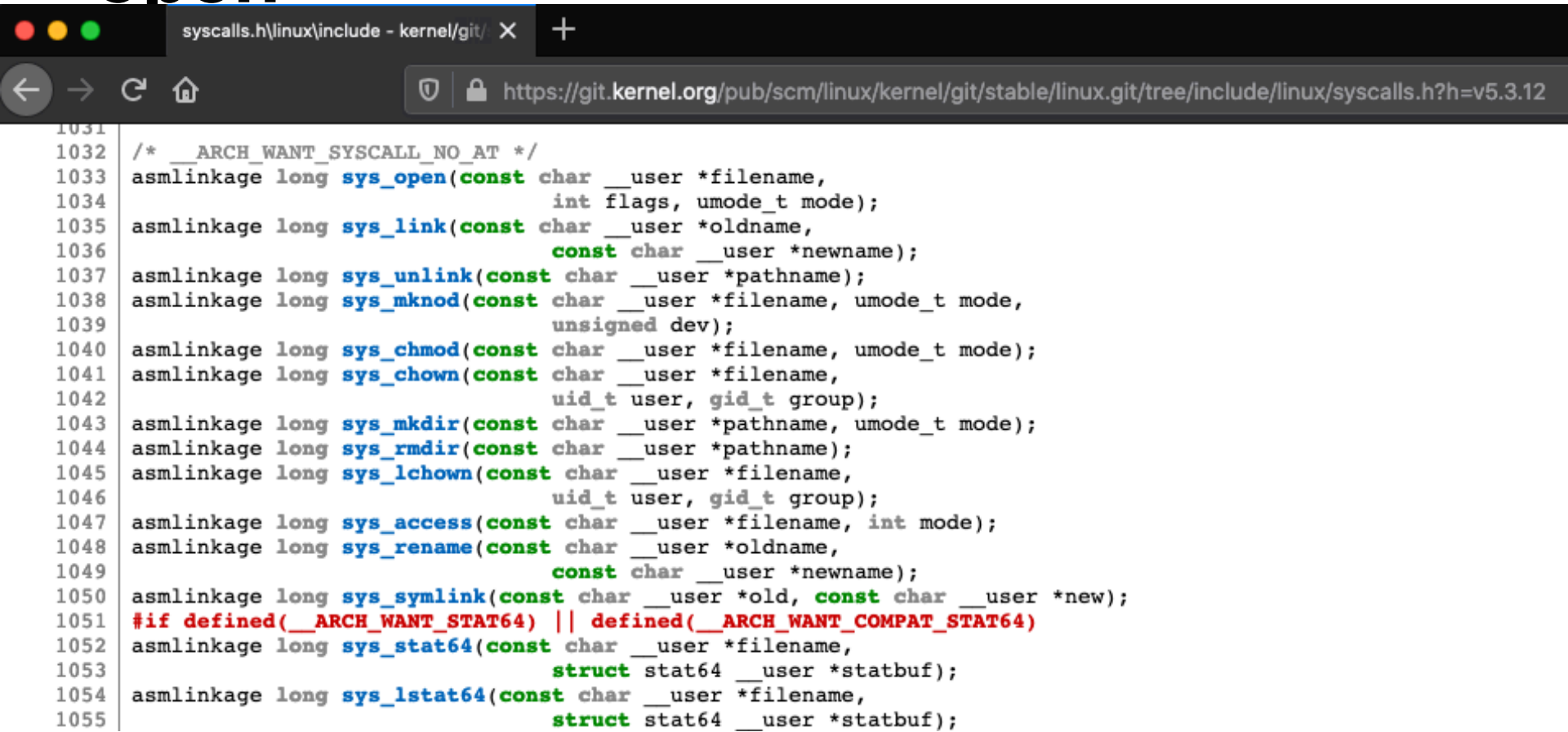
open, openat, creat - open and possibly create a file

SYNOPSIS top

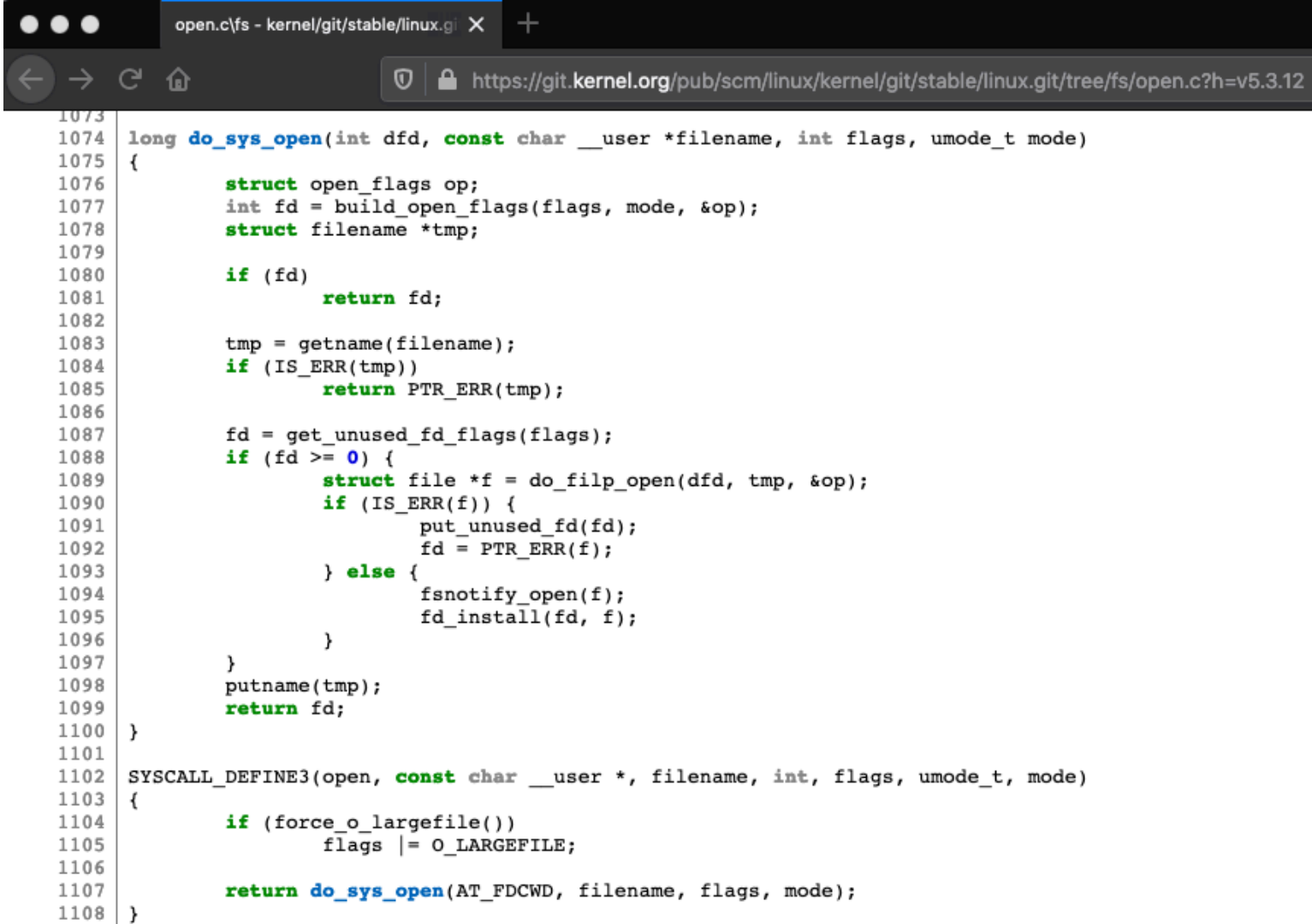
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

open


syscalls.h\linux\include - kernel/git/
https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/include/linux/syscalls.h?h=v5.3.12
1031
1032 /* __ARCH_WANT_SYSCALL_NO_AT */
1033 asmlinkage long sys_open(const char __user *filename,
1034 int flags, umode_t mode);
1035 asmlinkage long sys_link(const char __user *oldname,
1036 const char __user *newname);
1037 asmlinkage long sys_unlink(const char __user *pathname);
1038 asmlinkage long sys_mknod(const char __user *filename, umode_t mode,
1039 unsigned dev);
1040 asmlinkage long sys_chmod(const char __user *filename, umode_t mode);
1041 asmlinkage long sys_chown(const char __user *filename,
1042 uid_t user, gid_t group);
1043 asmlinkage long sys_mkdir(const char __user *pathname, umode_t mode);
1044 asmlinkage long sys_rmdir(const char __user *pathname);
1045 asmlinkage long sys_lchown(const char __user *filename,
1046 uid_t user, gid_t group);
1047 asmlinkage long sys_access(const char __user *filename, int mode);
1048 asmlinkage long sys_rename(const char __user *oldname,
1049 const char __user *newname);
1050 asmlinkage long sys_symlink(const char __user *old, const char __user *new);
1051 #if defined(__ARCH_WANT_STAT64) || defined(__ARCH_WANT_COMPAT_STAT64)
1052 asmlinkage long sys_stat64(const char __user *filename,
1053 struct stat64 __user *statbuf);
1054 asmlinkage long sys_lstat64(const char __user *filename,
1055 struct stat64 __user *statbuf);

open




```
1073
1074 long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
1075 {
1076         struct open_flags op;
1077         int fd = build_open_flags(flags, mode, &op);
1078         struct filename *tmp;
1079
1080         if (fd)
1081                 return fd;
1082
1083         tmp = getname(filename);
1084         if (IS_ERR(tmp))
1085                 return PTR_ERR(tmp);
1086
1087         fd = get_unused_fd_flags(flags);
1088         if (fd >= 0) {
1089                 struct file *f = do_filp_open(dfd, tmp, &op);
1090                 if (IS_ERR(f)) {
1091                         put_unused_fd(fd);
1092                         fd = PTR_ERR(f);
1093                 } else {
1094                         fsnotify_open(f);
1095                         fd_install(fd, f);
1096                 }
1097         }
1098         putname(tmp);
1099         return fd;
1100 }
1101
1102 SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
1103 {
1104         if (force_o_largefile())
1105                 flags |= O_LARGEFILE;
1106
1107         return do_sys_open(AT_FDCWD, filename, flags, mode);
1108 }
```

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/arch/x86/entry/syscalls/syscall_32.tbl?h=v5.3.12

blob: c00019abd0769d60d7a142db83b611d076e9ab0b (plain)

```
#
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The __ia32_sys and __ia32_compat_sys stubs are created on-the-fly for
# sys_*() system calls and compat_sys_*() compat system calls if
# IA32_EMULATION is defined, and expect struct pt_regs *regs as their only
# parameter.
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall             __ia32_sys_restart_syscall
1       i386    exit                    sys_exit                        __ia32_sys_exit
2       i386    fork                    sys_fork                        __ia32_sys_fork
3       i386    read                    sys_read                        __ia32_sys_read
4       i386    write                   sys_write                       __ia32_sys_write
5       i386    open                    sys_open                        __ia32_compat_sys_open
6       i386    close                   sys_close                       __ia32_sys_close
7       i386    waitpid                 sys_waitpid                     __ia32_sys_waitpid
8       i386    creat                   sys_creat                       __ia32_sys_creat
9       i386    link                    sys_link                        __ia32_sys_link
10      i386    unlink                  sys_unlink                      __ia32_sys_unlink
11      i386    execve                  sys_execve                      __ia32_compat_sys_execve
12      i386    chdir                   sys_chdir                       __ia32_sys_chdir
13      i386    time                    sys_time32                      __ia32_sys_time32
14      i386    mknod                   sys_mknod                       __ia32_sys_mknod
15      i386    chmod                   sys_chmod                       __ia32_sys_chmod
```

Cooperative Multitasking

- Most Operating Systems to day **preempt** processes to switch contexts.
- Cooperative models ask processes to periodically **yield control** back to the OS.
- Examples:
  - Mac OS 9. Windows 16bit apps

Linux

- ps
- ls /sys, /proc
- top
- kill
- *ctrl-z*, bg, fg

Homework for next lecture

A network packet arrives.

Later, an application attempts to 'recv' (receive) the packet.

When should code run to:

a. Validate the packet checksum?

b. Copy the packet to user memory?