Debugging with strace

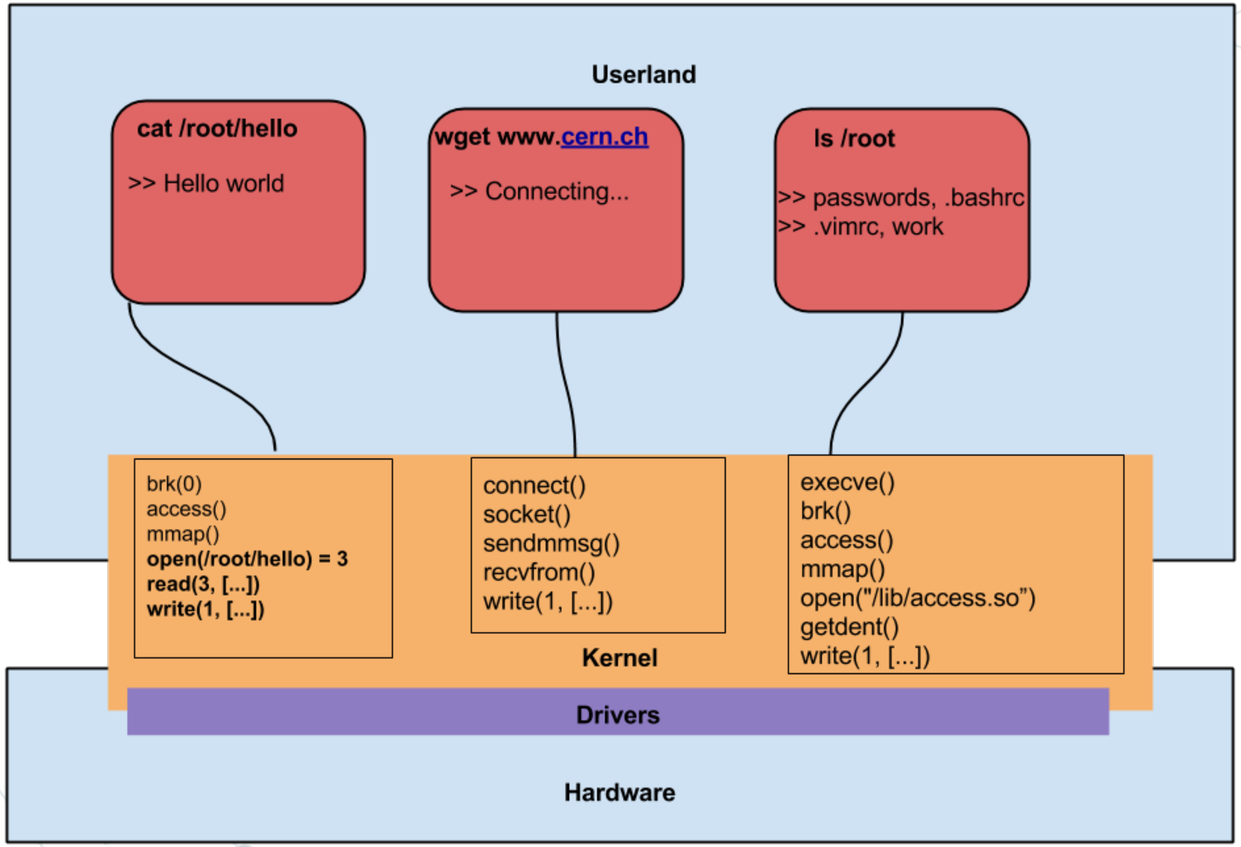
a.k.a. what are my applications doing underneath?

Marek Denis
marek.denis@cern.ch

System call

- In Linux, whenever your application wants to
    - open a file
    - read/write the file
    - send a packet over the wire
    - print calculation result to the user
    - allocate memory
    - create another process
- ...it does this through kernel by using **system calls**

System call



Userland
cat /root/hello
>> Hello world
wget www.cern.ch
>> Connecting...
ls /root
>> passwords, .bashrc
>> .vimrc, work
brk(0)
access()
mmap()
open(/root/hello) = 3
read(3, [...])
write(1, [...])
connect()
socket()
sendmmsg()
recvfrom()
write(1, [...])
execve()
brk()
access()
mmap()
open("/lib/access.so")
getdent()
write(1, [...])
Kernel
Drivers
Hardware

strace - trace system calls (and more)

- Lists all the system calls launched while running your application
- Great for profiling your applications
- Great for debugging, especially blackbox debugging
- ...even greater for learning what is really happening when you type **cat /etc/passwd**

strace - sample usage

```
$ strace <command>
    # strace cat /etc/passwd

$ strace -p <pid of the running process>
    # strace -p `pidof myapp`
```

strace - output

- All executed syscalls (may be filtered)
- Shows return values of syscalls and errno (if set)

  $ strace -e trace=open *cat .bashrc*

  >> open(“.bashrc”, O_RDONLY) = 3 (file descriptor number)

- Shows (truncated) arguments for syscalls:

  $ strace -e trace=write *echo “Hello CERN!”*

  >> write(1, “Hello CERN!\n”, 12) = 12

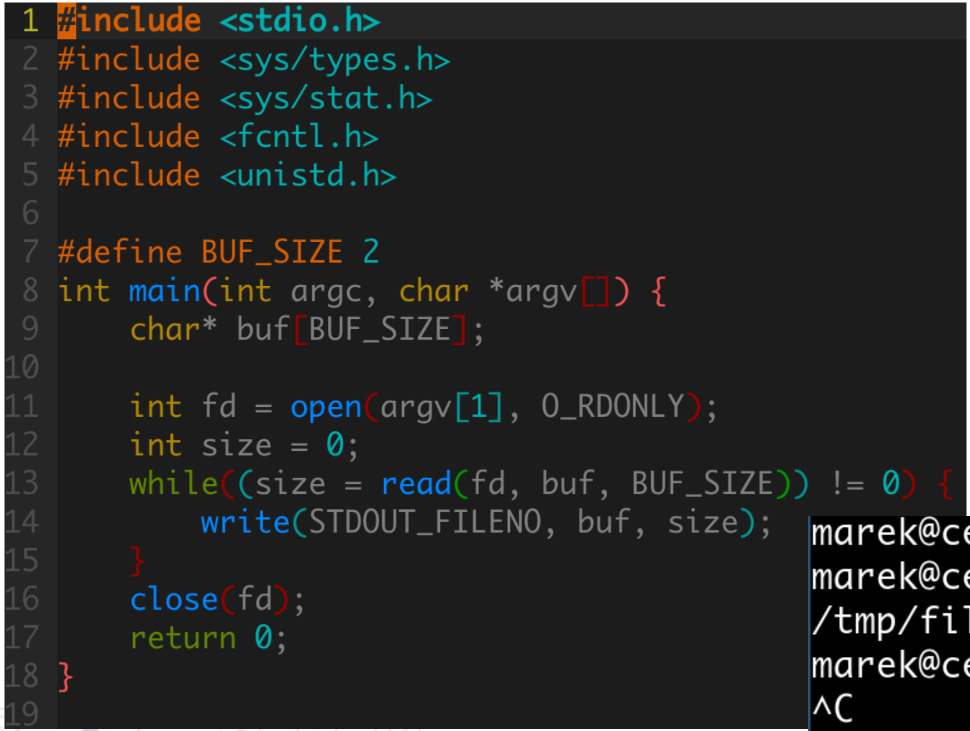
strace - useful options

strace -o <file>	- save output to a file
strace -c	- prints statistics of numbers of syscalls used
strace -T	- show time spent in syscalls
strace -t[tt]	- prefix each syscall with time of the day
strace -f	- trace also child processes

strace -e trace={open,write,read,fstat,ioctl,connect,bind} - filter specific syscalls

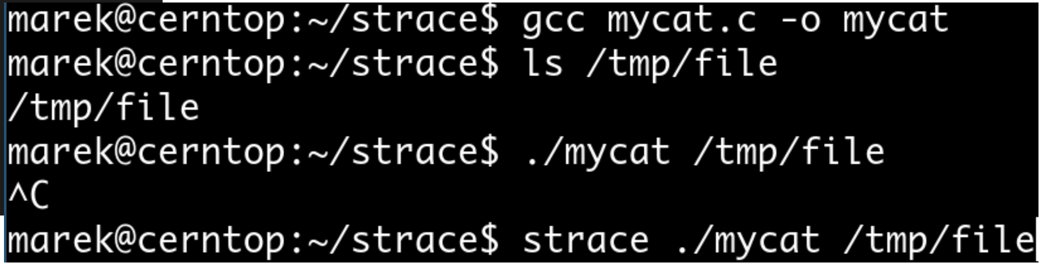
strace -e trace={process,network,signal,memory} - filter groups of syscalls

Debugging with strace



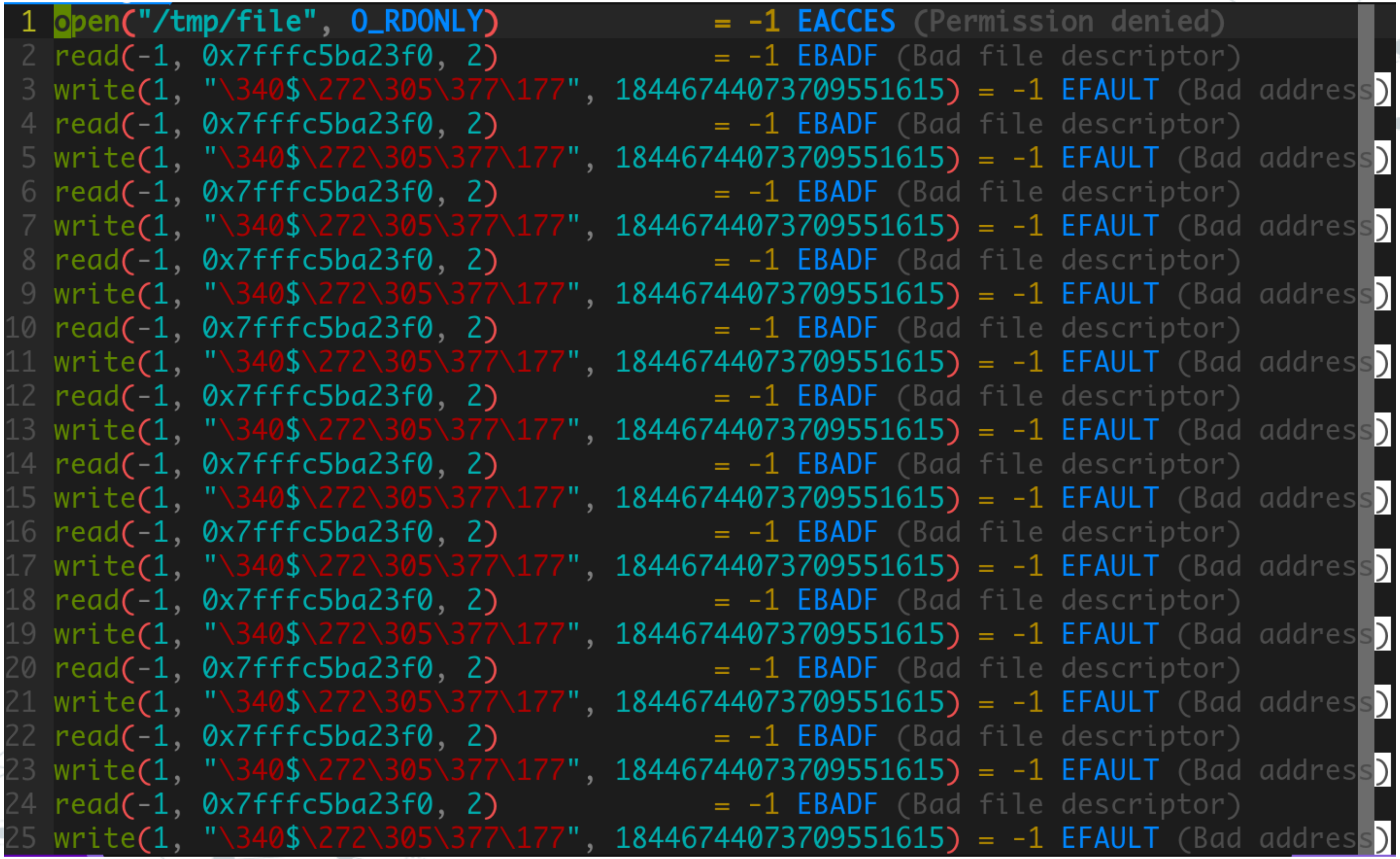
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define BUF_SIZE 2
int main(int argc, char *argv[]) {
    char* buf[BUF_SIZE];

    int fd = open(argv[1], O_RDONLY);
    int size = 0;
    while((size = read(fd, buf, BUF_SIZE)) != 0) {
        write(STDOUT_FILENO, buf, size);
    }
    close(fd);
    return 0;
}

```

- The code looks fine at the first glance
  a. open a file
  b. read chunks of data until the file ends
  c. write text chunks to stdout
  d. close file descriptor
- This code has at least 4 bugs - can you spot them?
- **It compiles and runs but may hang without a “reason”**



```
marek@cerntop:~/strace$ gcc mycat.c -o mycat
marek@cerntop:~/strace$ ls /tmp/file
/tmp/file
marek@cerntop:~/strace$ ./mycat /tmp/file
^C
marek@cerntop:~/strace$ strace ./mycat /tmp/file
```

Debugging with strace



```
open("/tmp/file", O_RDONLY)                    = -1 EACCES (Permission denied)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
read(-1, 0x7fffc5ba23f0, 2)                    = -1 EBADF (Bad file descriptor)
write(1, "\340$\272\305\377\177", 18446744073709551615) = -1 EFAULT (Bad address)
```

Simple echo turns out to be not that simple….

```
marek@cerntop:~$ strace echo "That's all folks!"
execve("/bin/echo", ["echo", "That's all folks!"], [/* 59 vars */]) = 0
brk(0)                                  = 0x1a4b000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f647e905000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=97882, ...}) = 0
mmap(NULL, 97882, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f647e8ed000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\37\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1840928, ...}) = 0
mmap(NULL, 3949248, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f647e320000
mprotect(0x7f647e4db000, 2093056, PROT_NONE) = 0
mmap(0x7f647e6da000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7f647e6da000
mmap(0x7f647e6e0000, 17088, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f647e6e0000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f647e8ec000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f647e8ea000
arch_prctl(ARCH_SET_FS, 0x7f647e8ea740) = 0
mprotect(0x7f647e6da000, 16384, PROT_READ) = 0
mprotect(0x606000, 4096, PROT_READ)     = 0
mprotect(0x7f647e907000, 4096, PROT_READ) = 0
munmap(0x7f647e8ed000, 97882)           = 0
brk(0)                                  = 0x1a4b000
brk(0x1a6c000)                          = 0x1a6c000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=8740368, ...}) = 0
mmap(NULL, 8740368, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f647daca000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f647e904000
write(1, "That's all folks!\n", 18That's all folks!)     = 18
close(1)                                = 0
munmap(0x7f647e904000, 4096)            = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
marek@cerntop:~$
```

We barely scratched the surface

```
$ man 8 strace
$ man 8 ltrace
```

or see strace on steroids

http://www.sysdig.org/

Questions

…and hopefully answers