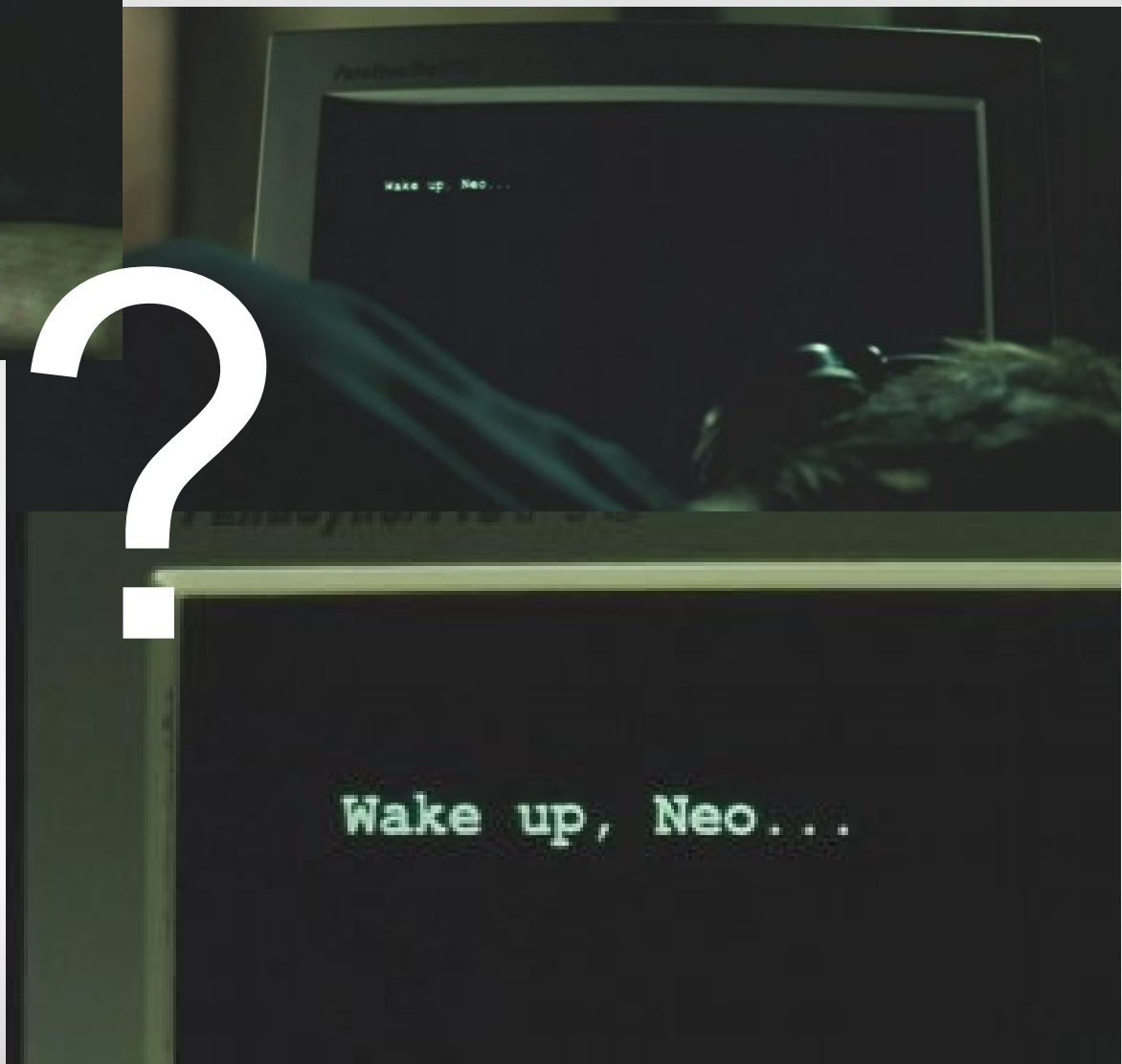
$ Wake up, Neo...

Confidence 2012
22 may
Łukasz Taczuk

1999 r. - The Matrix

?

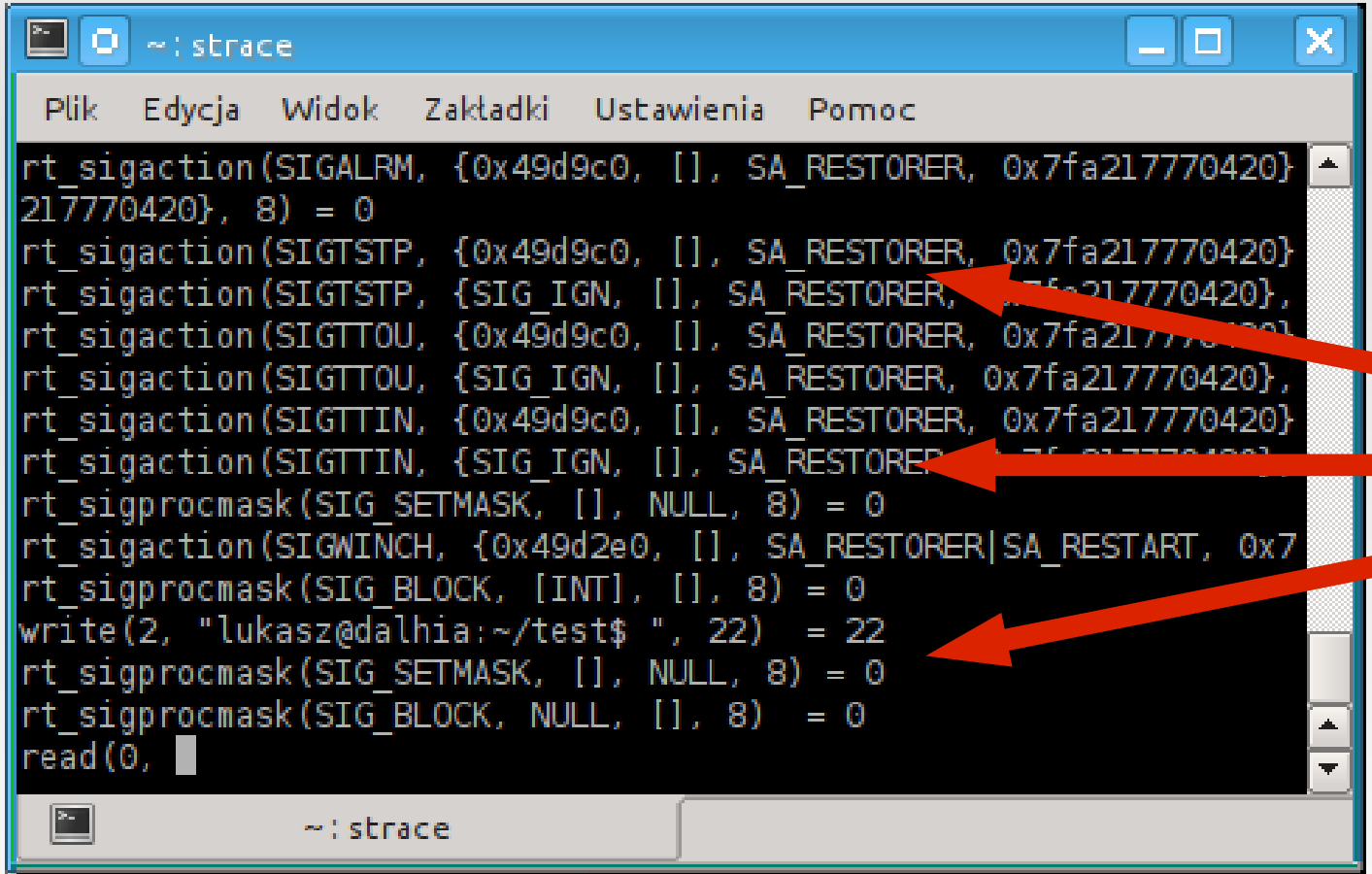


strace

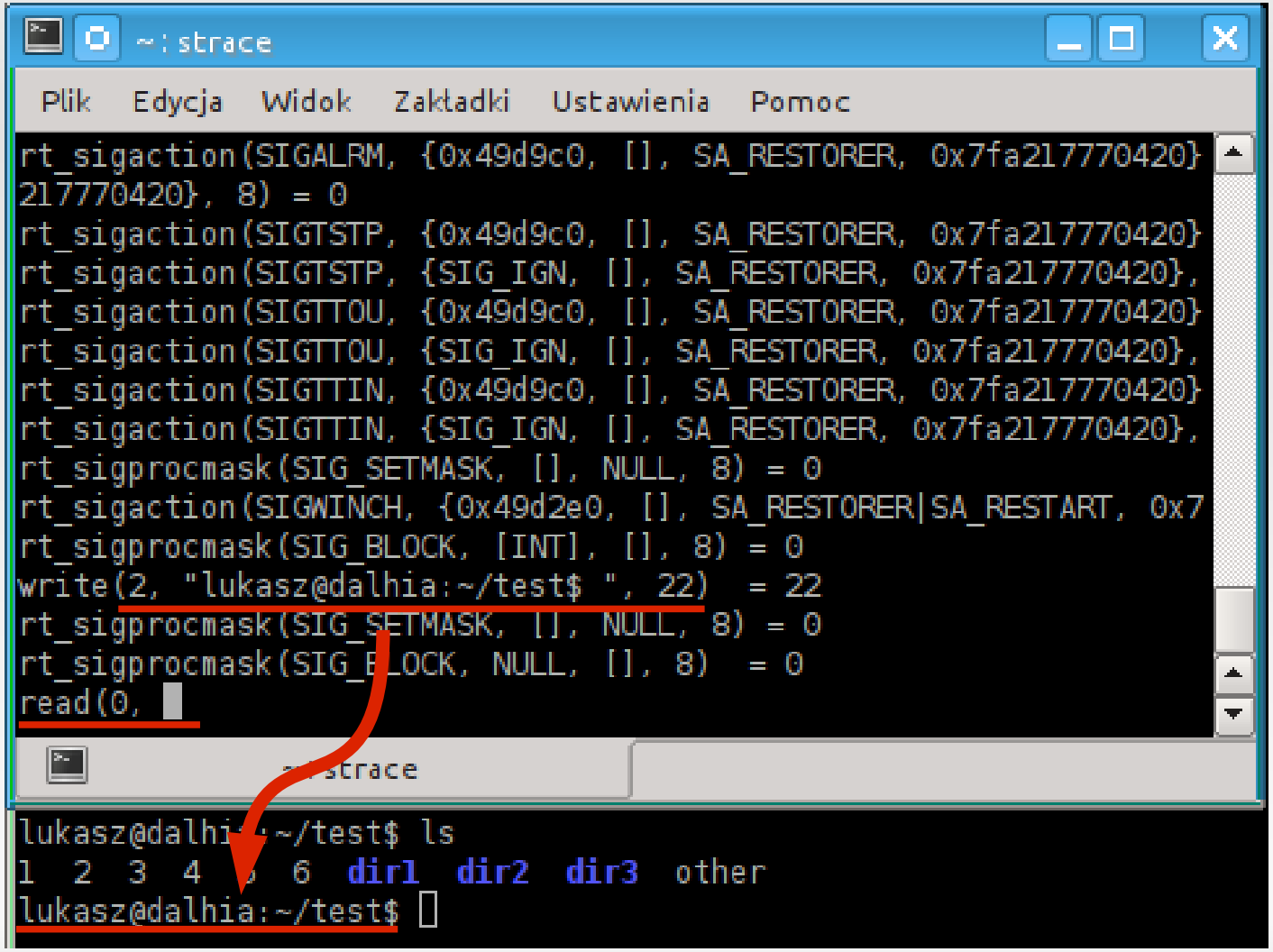
What Wikipedia has to say:

***strace** is a debugging utility for Linux (...) to monitor the system calls used by a program and all the signals it receives(...).*
*This is made possible by a kernel feature known as **ptrace**.*

strace


~: strace
Plik Edycja Widok Zakładki Ustawienia Pomoc
rt_sigaction(SIGALRM, {0x49d9c0, [], SA_RESTORER, 0x7fa217770420}
217770420}, 8) = 0
rt_sigaction(SIGTSTP, {0x49d9c0, [], SA_RESTORER, 0x7fa217770420}
rt_sigaction(SIGTSTP, {SIG_IGN, [], SA_RESTORER,
rt_sigaction(SIGTTOU, {0x49d9c0, [], SA_RESTORER, 0x7fa21
rt_sigaction(SIGTTOU, {SIG_IGN, [], SA_RESTORER, 0x7fa217770420},
rt_sigaction(SIGTTIN, {0x49d9c0, [], SA_RESTORER, 0x7fa217770420}
rt_sigaction(SIGTTIN, {SIG_IGN, [], SA_RESTORER
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigaction(SIGWINCH, {0x49d2e0, [], SA_RESTORER|SA_RESTART, 0x7
rt_sigprocmask(SIG_BLOCK, [INT], [], 8) = 0
write(2, "lukasz@dalhia:~/test$ ", 22) = 22
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0
read(0,
~: strace
lukasz@dalhia:~/test$ ls
1 2 3 4 5 6 dir1 dir2 dir3 other
lukasz@dalhia:~/test$
strace output
Straced bash

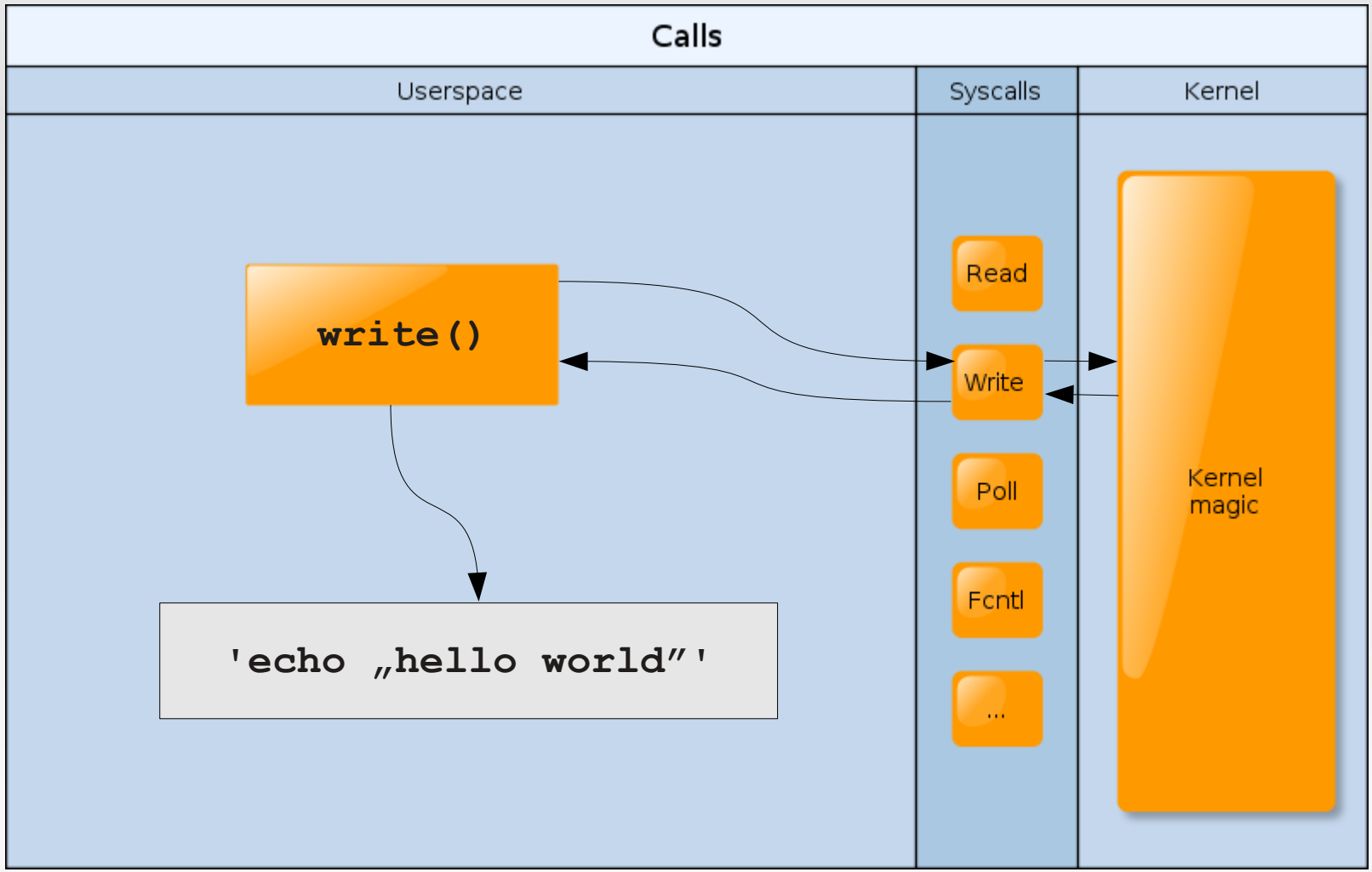
Once upon a time



Interesting...

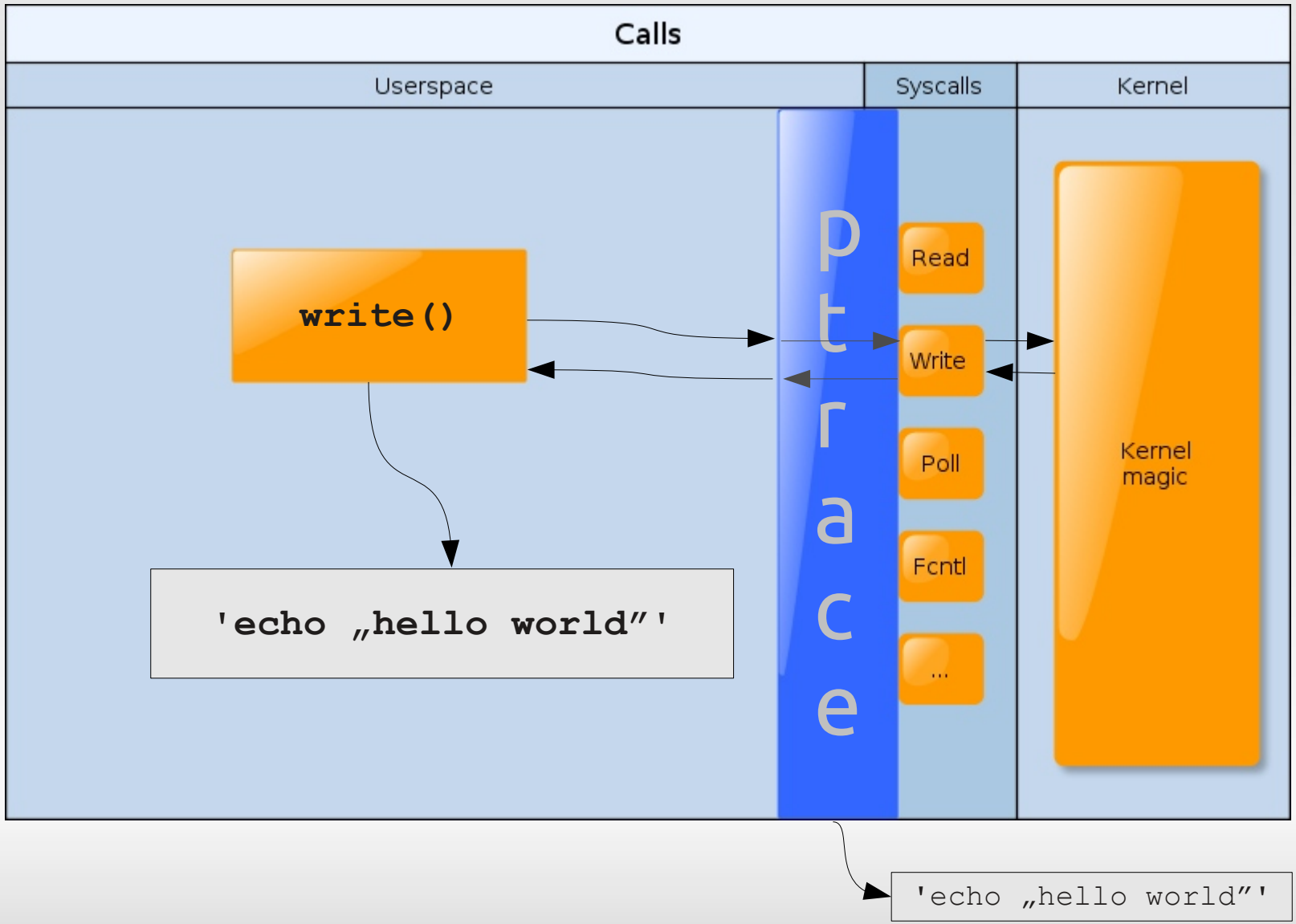
How does writing to stdout work?

```
ssize_t write(int fd, const void *buf, size_t count);
```



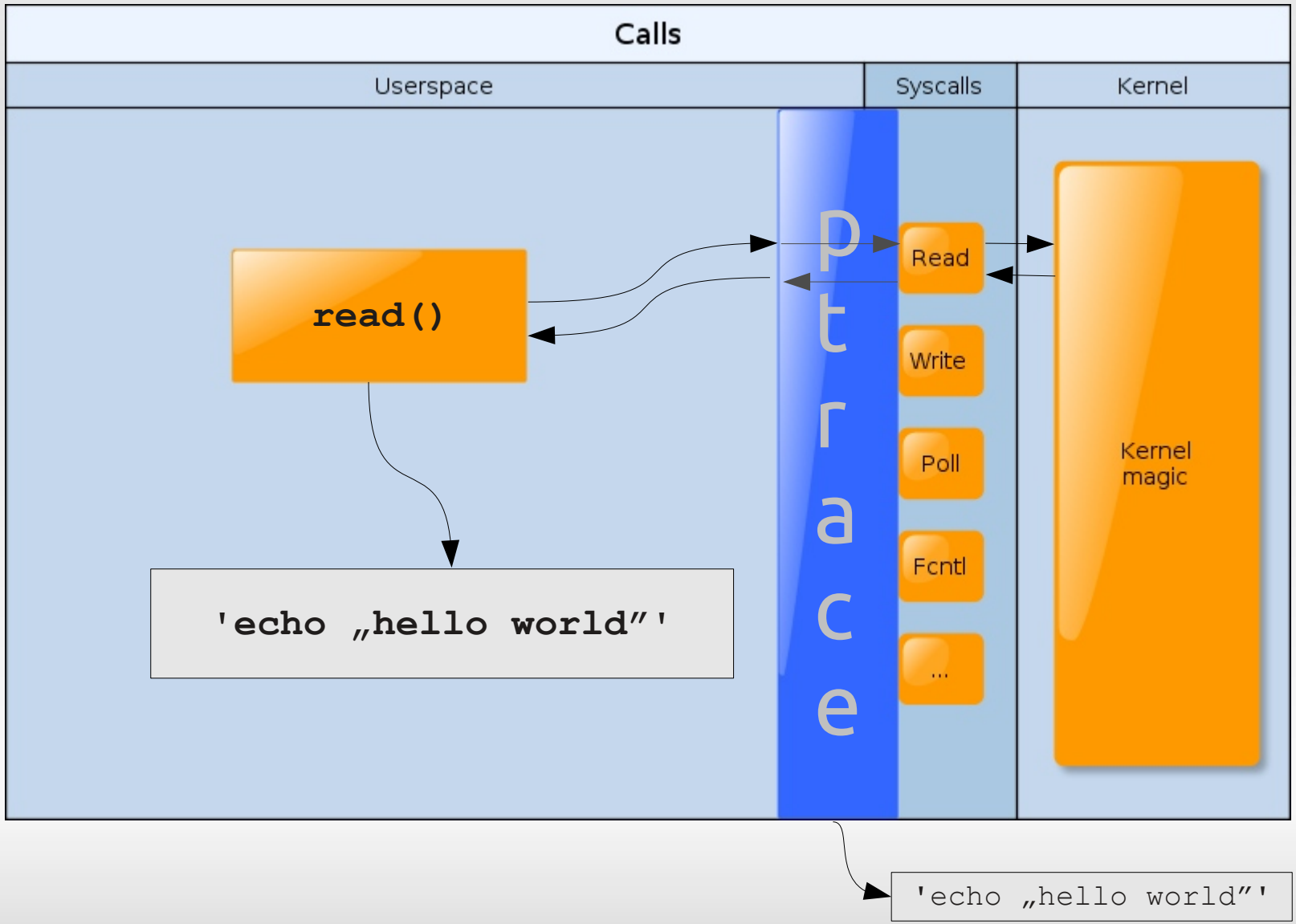
ptrace – the gate-crasher

```
ssize_t write(int fd, const void *buf, size_t count);
```



Intercepting read

```
ssize_t read(int fd, void *buf, size_t count);
```



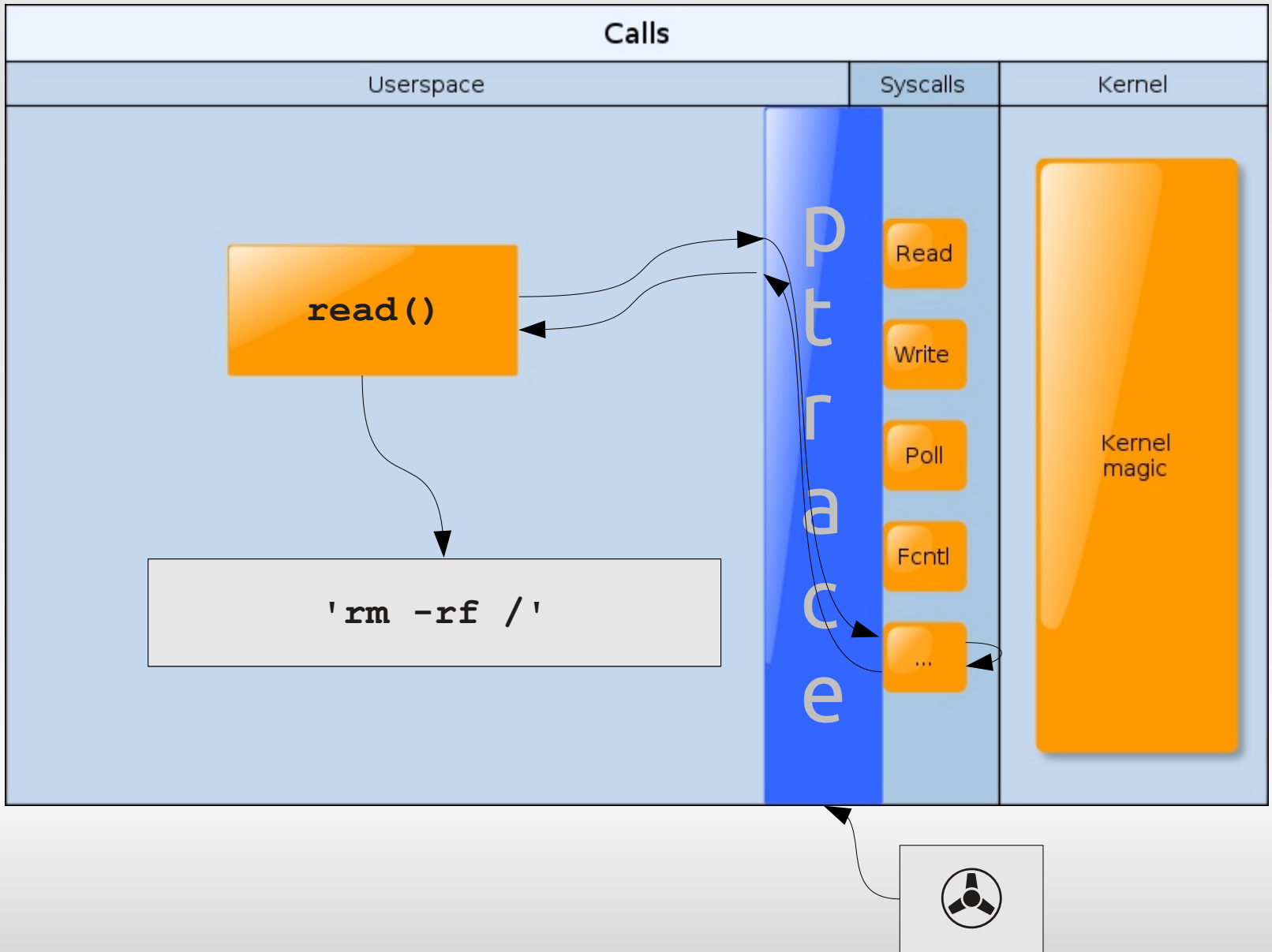
The idea

Debugging gives us both read and **write** access to the debuggee's memory/registers

What would happen if we **modified** read or write syscalls' parameters?

Modifying read!

```
ssize_t read(int fd, void *buf, size_t count);
```



...and that's all?

...and that's all?

Nope.

Asynchronous input

- read/write calls are usually blocking!
- ptrace can also intercept signals
- Send a sigstop signal to the traced process
- *Ignore* that signal with ptrace
- Continue execution
- The syscall is automagically restarted (at least it should be :)

Demo time!

Let's see how deep the rabbit hole goes

ptrace restrictions

(Un)fortunately, tracing the following is impossible:

- Already traced programs
- Programs with the suid bit set
- CAP_SYS_PTRACE = 0
- Generally, any untraceable program (need to be the same user)

How to protect yourself?

Patch the Kernel

- Grsecurity

Or you can use Ubuntu :)

…and don't let anybody ssh into your computer

TODO

A lot of ideas but I'm probably running out of time :)

Thank you for your attention.

`https://github.com/overfl0/iojack`

note: as of now, experimental output hiding is located in 'hideoutput' branch