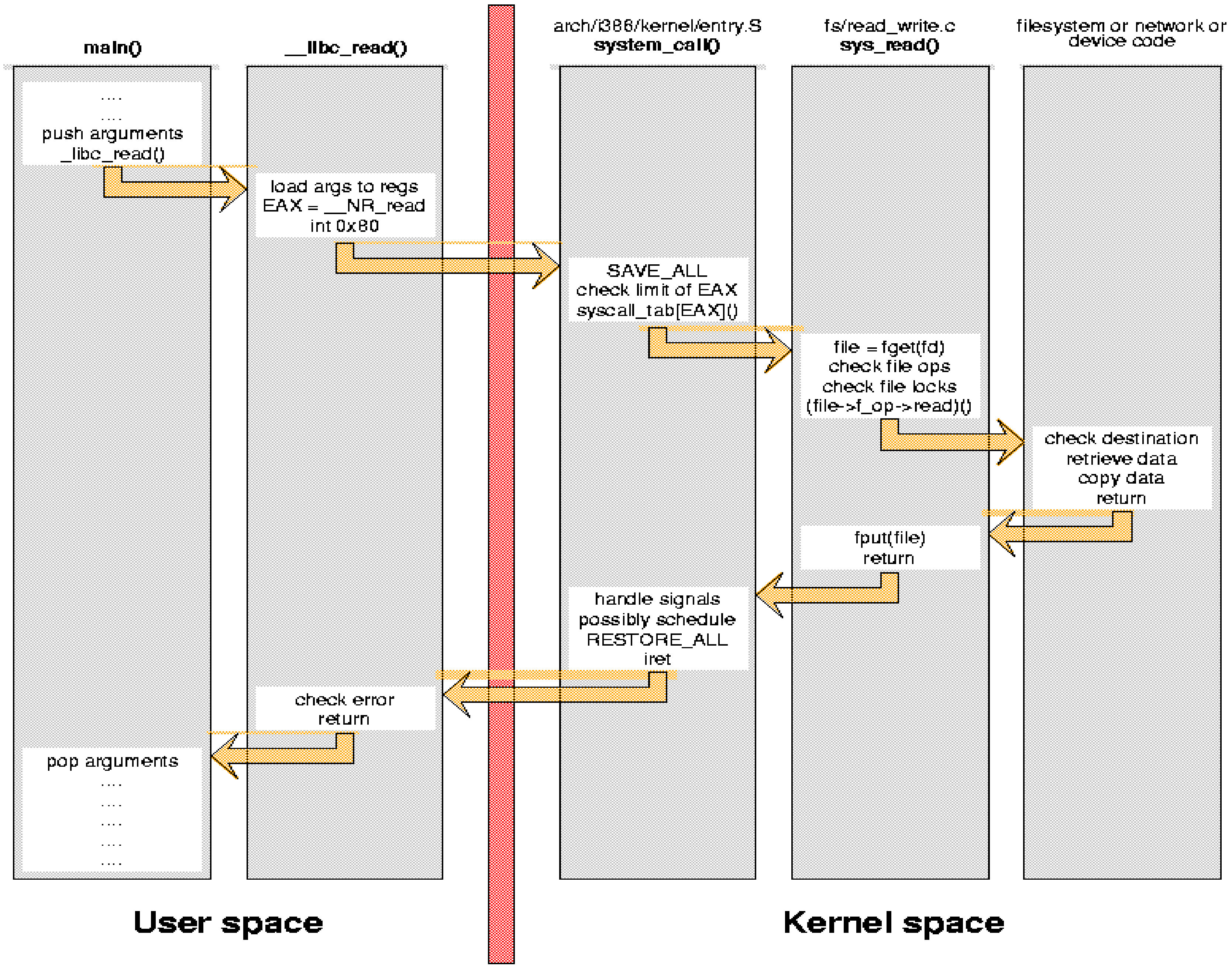
Ausgewählte Betriebssysteme

Anatomy of a system call

User view

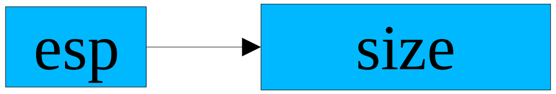
-
  ```
  #include <stdio.h>

  int
  main(void)
  {
      printf(“Hello World!\n”);
      return 0;
  }
  ```


main()
__libc_read()
arch/i386/kernel/entry.S
system_call()
fs/read_write.c
sys_read()
filesystem or network or device code
....
....
push arguments
_libc_read()
load args to regs
EAX = __NR_read
int 0x80
SAVE_ALL
check limit of EAX
syscall_tab[EAX]()
file = fget(fd)
check file ops
check file locks
(file->f_op->read)()
check destination
retrieve data
copy data
return
fput(file)
return
handle signals
possibly schedule
RESTORE_ALL
iret
check error
return
pop arguments
....
....
....
....
....
User space
Kernel space

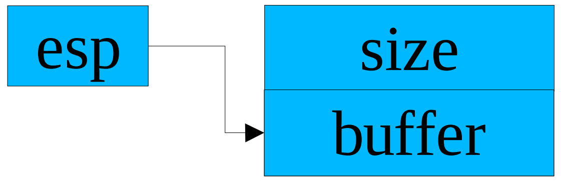
Syscall (1)

User: `write(fd, buffer, sizeof(buffer));`



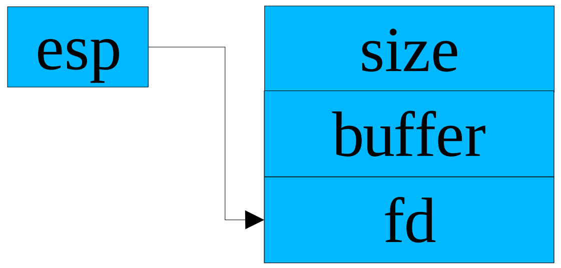
Syscall (2)

User: `write(fd, buffer, size);`



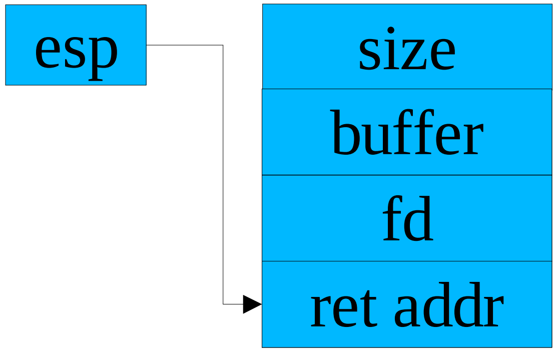
Syscall (3)

User: `write(fd, buffer, size);`



Syscall (4)

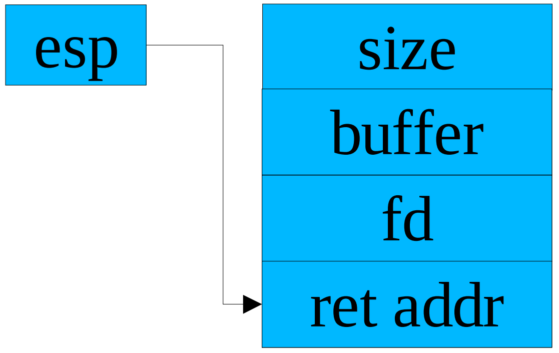
User: `write(fd, buffer, size);` write:



?

Syscall (5)

User: `write(fd, buffer, size);` write:

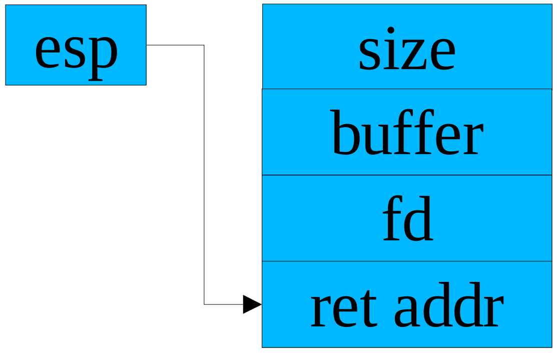


?

Kernel: `sys_write(fd, buffer, size);` (linux/fs/read_write.c)

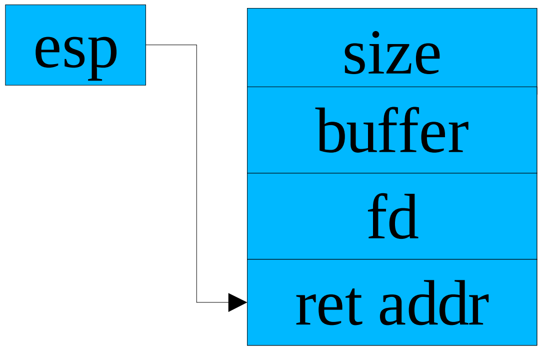
Syscall (6)

User: `write(fd, buffer, size);` write:

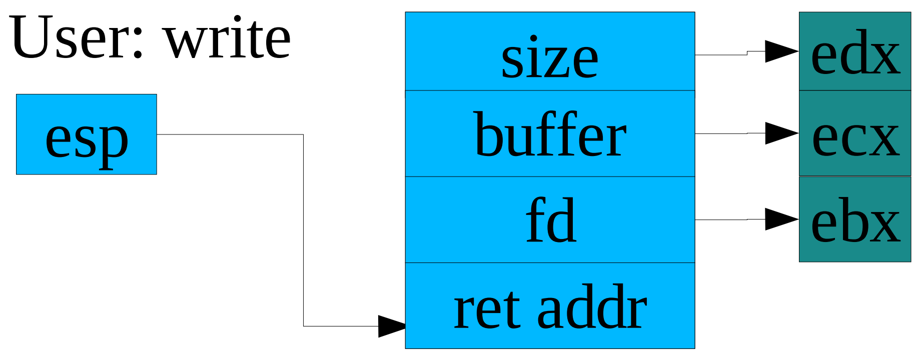


?

Kernel: `sys_write(fd, buffer, size);`

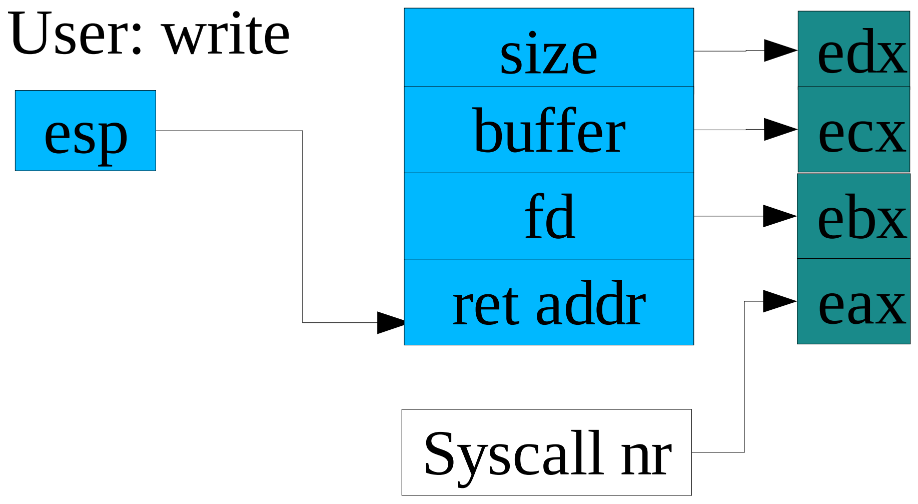


Syscall (7)



```
movl <size>, %edx
movl <buffer>, %ecx
movl <fd>, %ebx
```

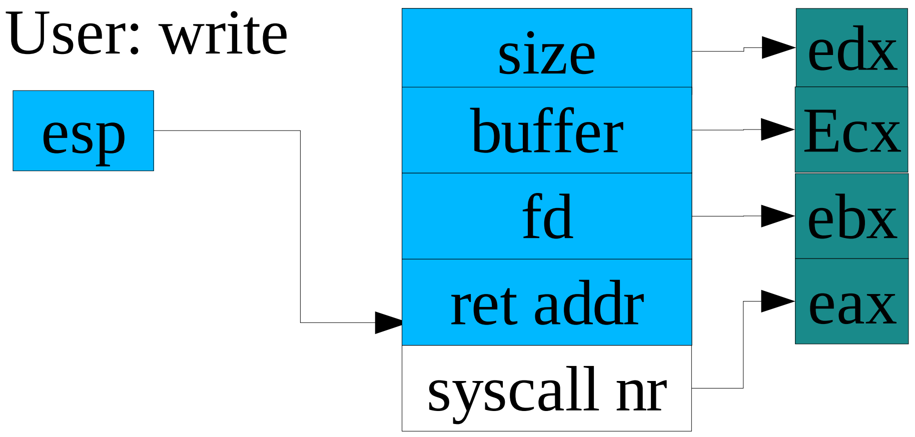
Syscall (8)



```
movl <size>, %edx
movl <buffer>, %ecx
movl <fd>, %ebx
movl $__NR_write, %eax
```

User
Kernel

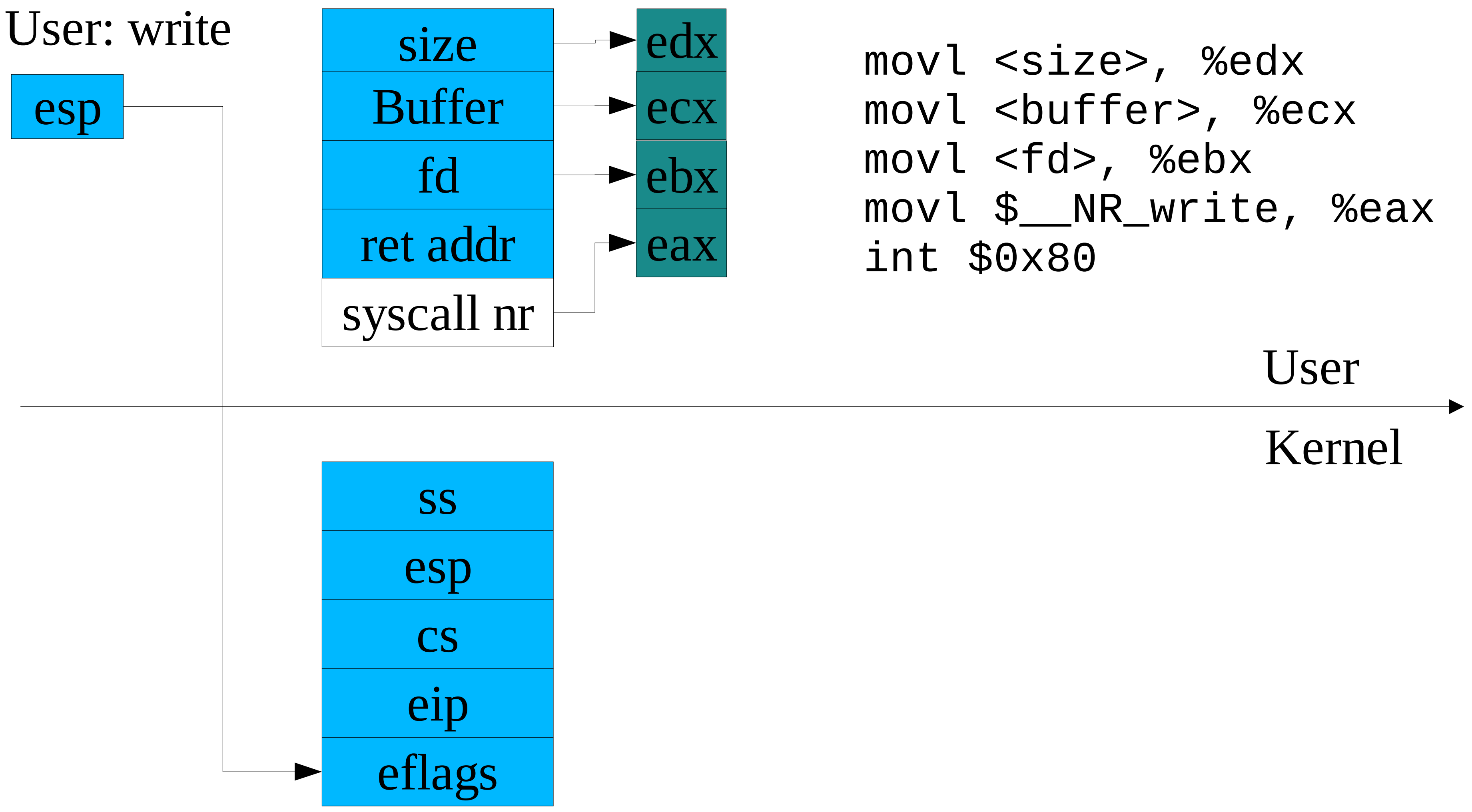
Syscall (9)



```
movl <size>, %edx
movl <buffer>, %ecx
movl <fd>, %ebx
movl $__NR_write, %eax
int $0x80
```

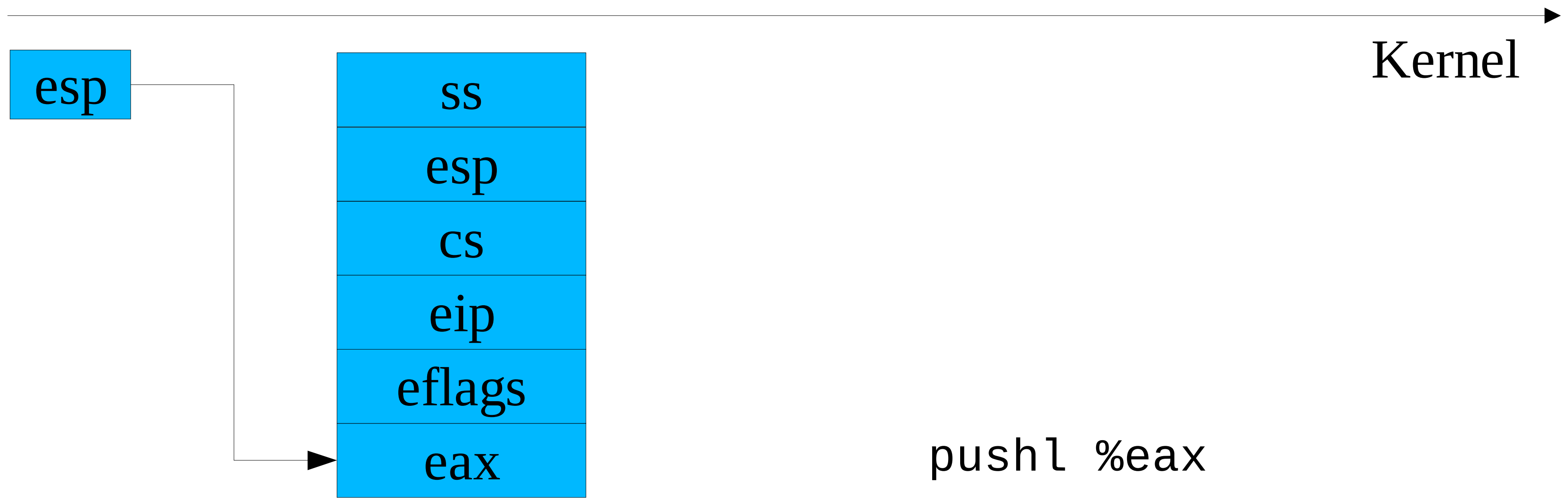
User
Kernel

Syscall (10)



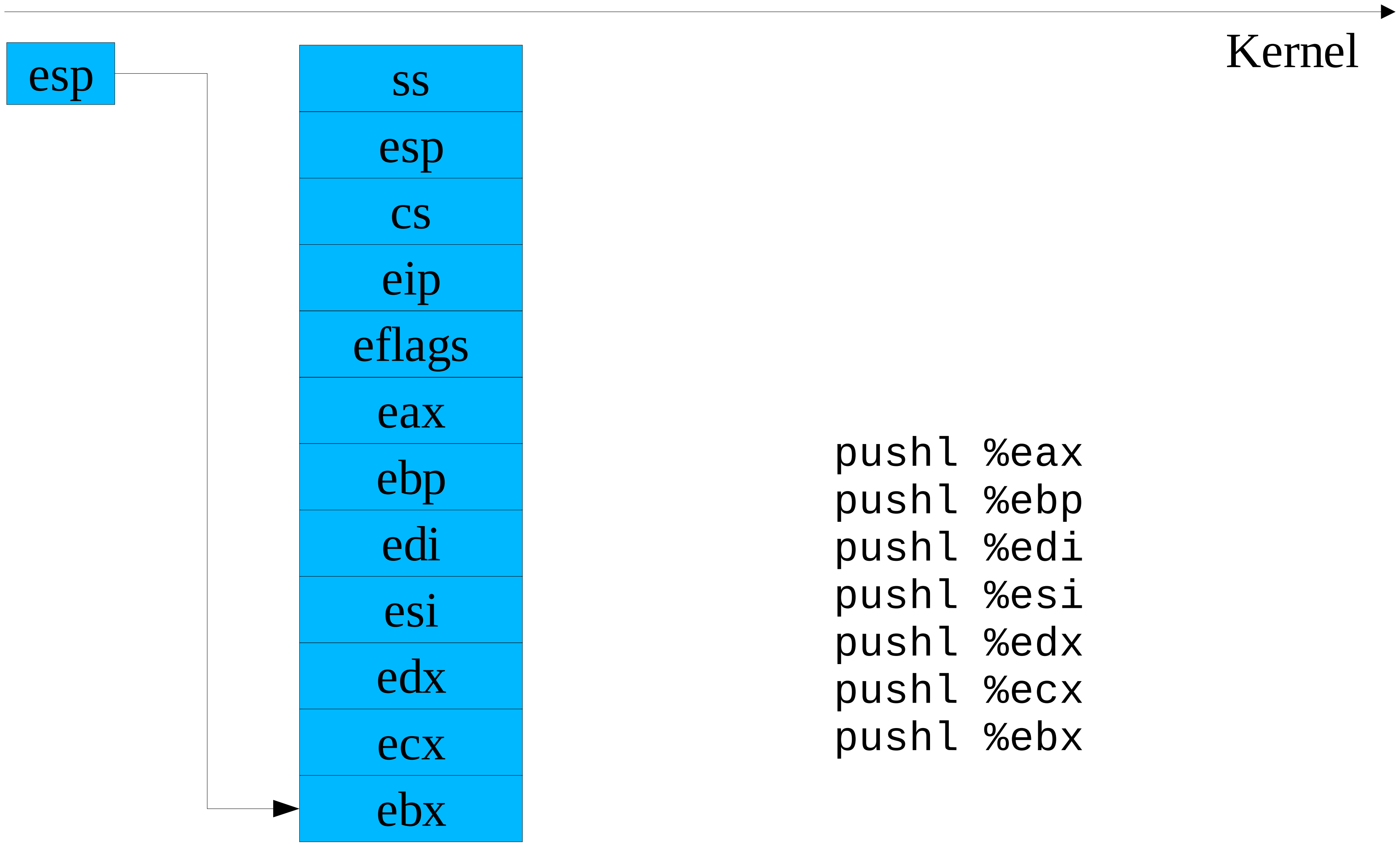
```
movl <size>, %edx
movl <buffer>, %ecx
movl <fd>, %ebx
movl $__NR_write, %eax
int $0x80
```

Syscall (11)



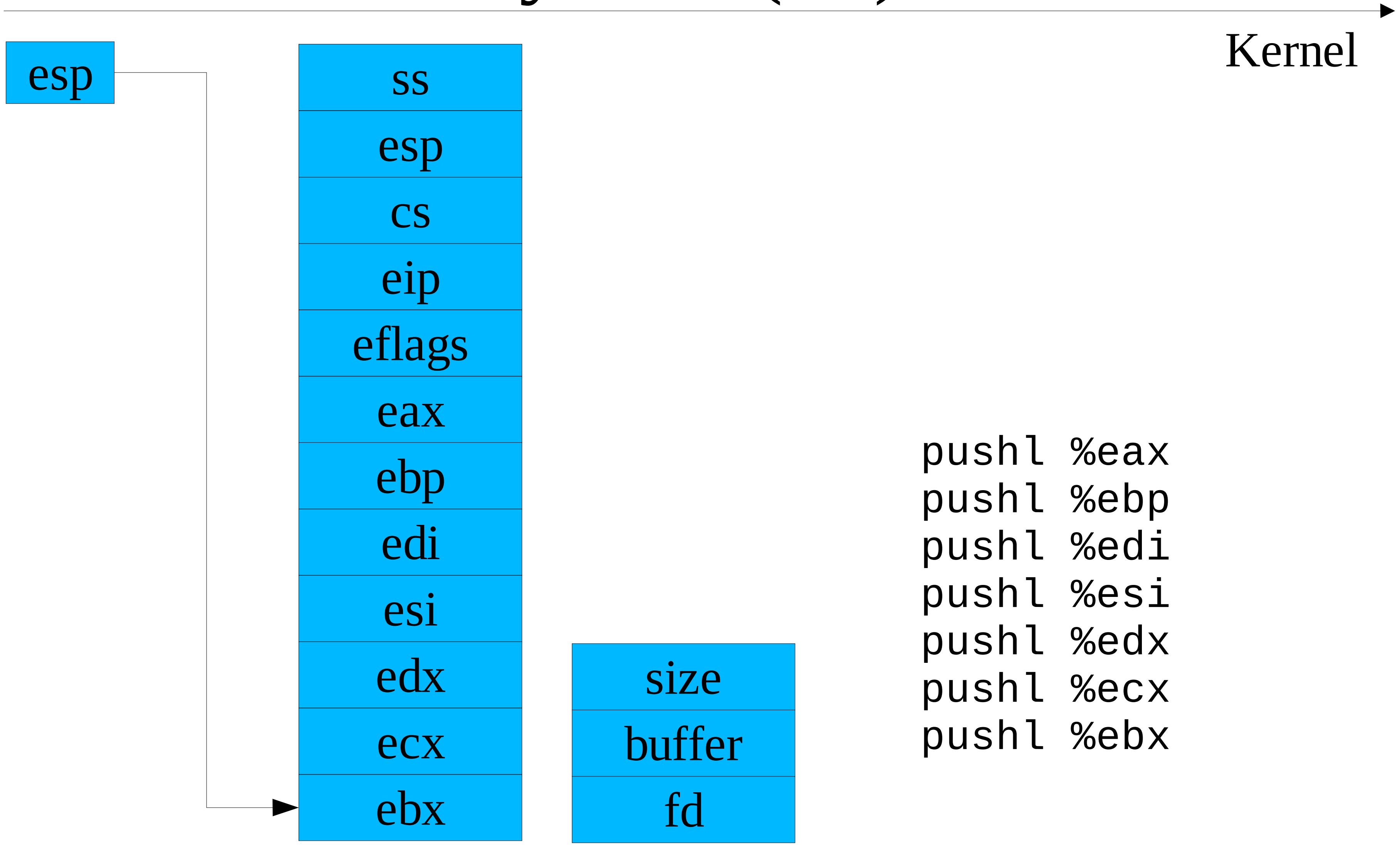
```
pushl %eax
```

Syscall (12)



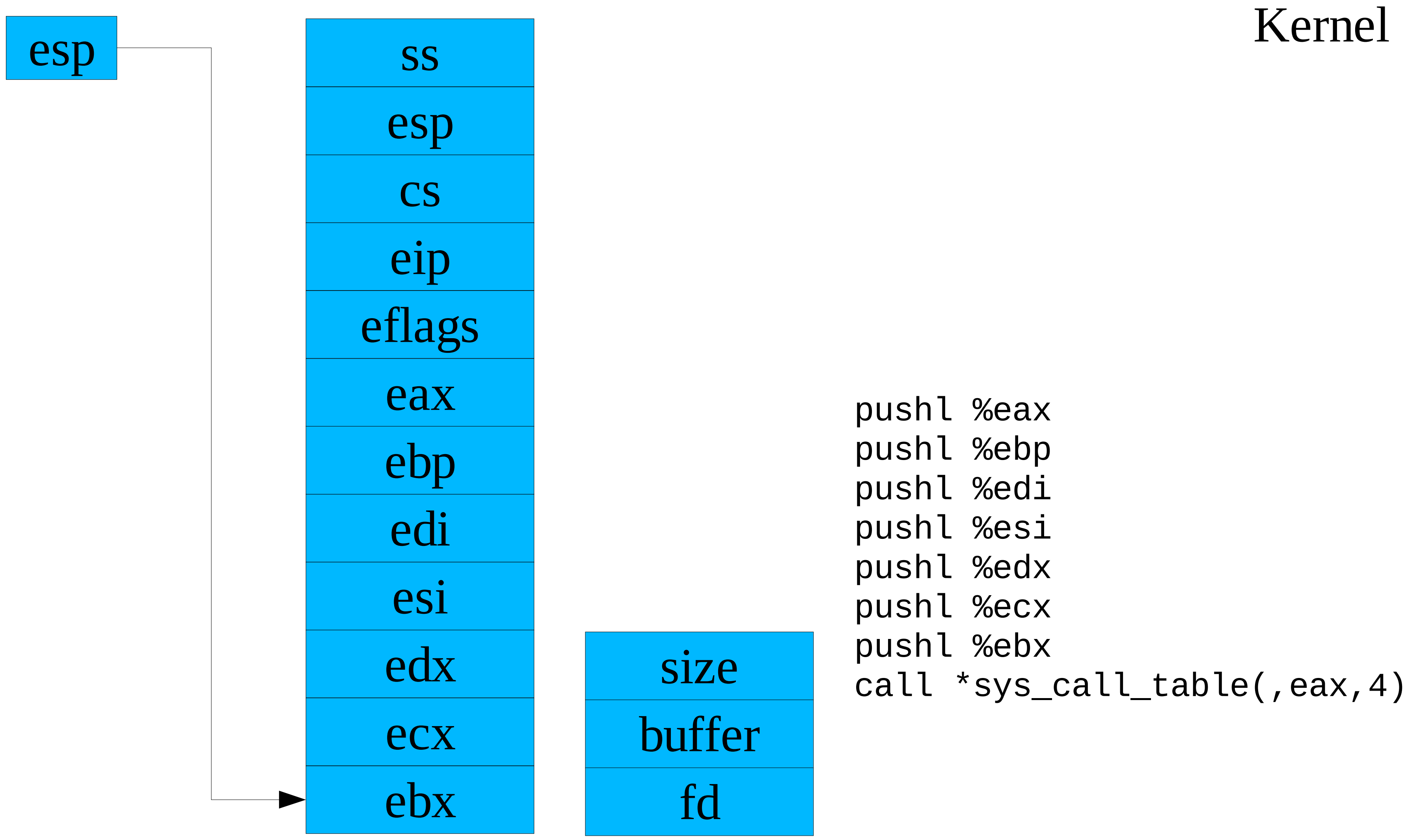
```
pushl %eax
pushl %ebp
pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
```

Syscall (13)



```
pushl %eax
pushl %ebp
pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
```

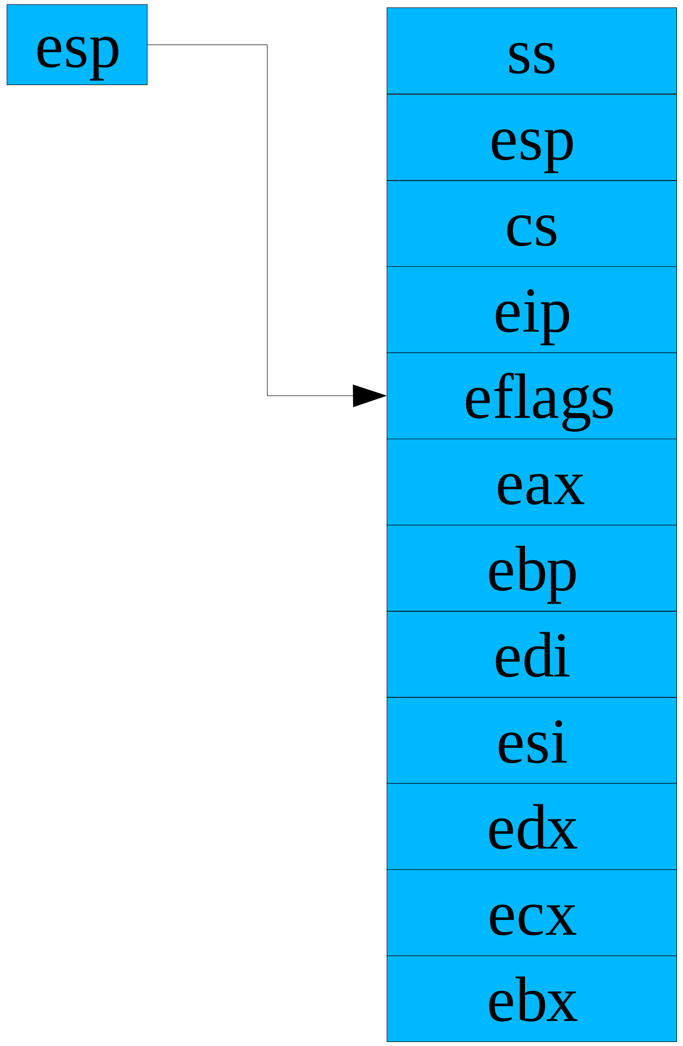
Syscall (14)



```
pushl %eax
pushl %ebp
pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
call *sys_call_table(,eax,4)
```

Syscall (15)

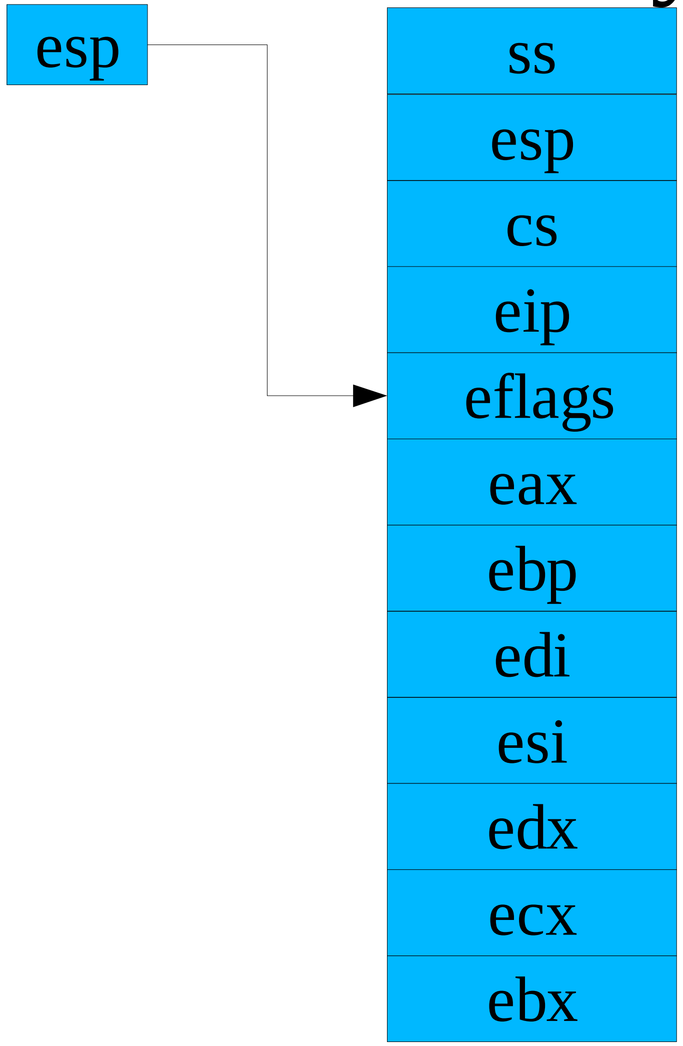
Kernel



```
pushl %eax
pushl %ebp
pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
call *sys_call_table(,eax,4)
movl %eax, EAX(%esp)
popl %ebx
popl %ecx
popl %edx
popl %esi
popl %edi
popl %ebp
popl %eax
```

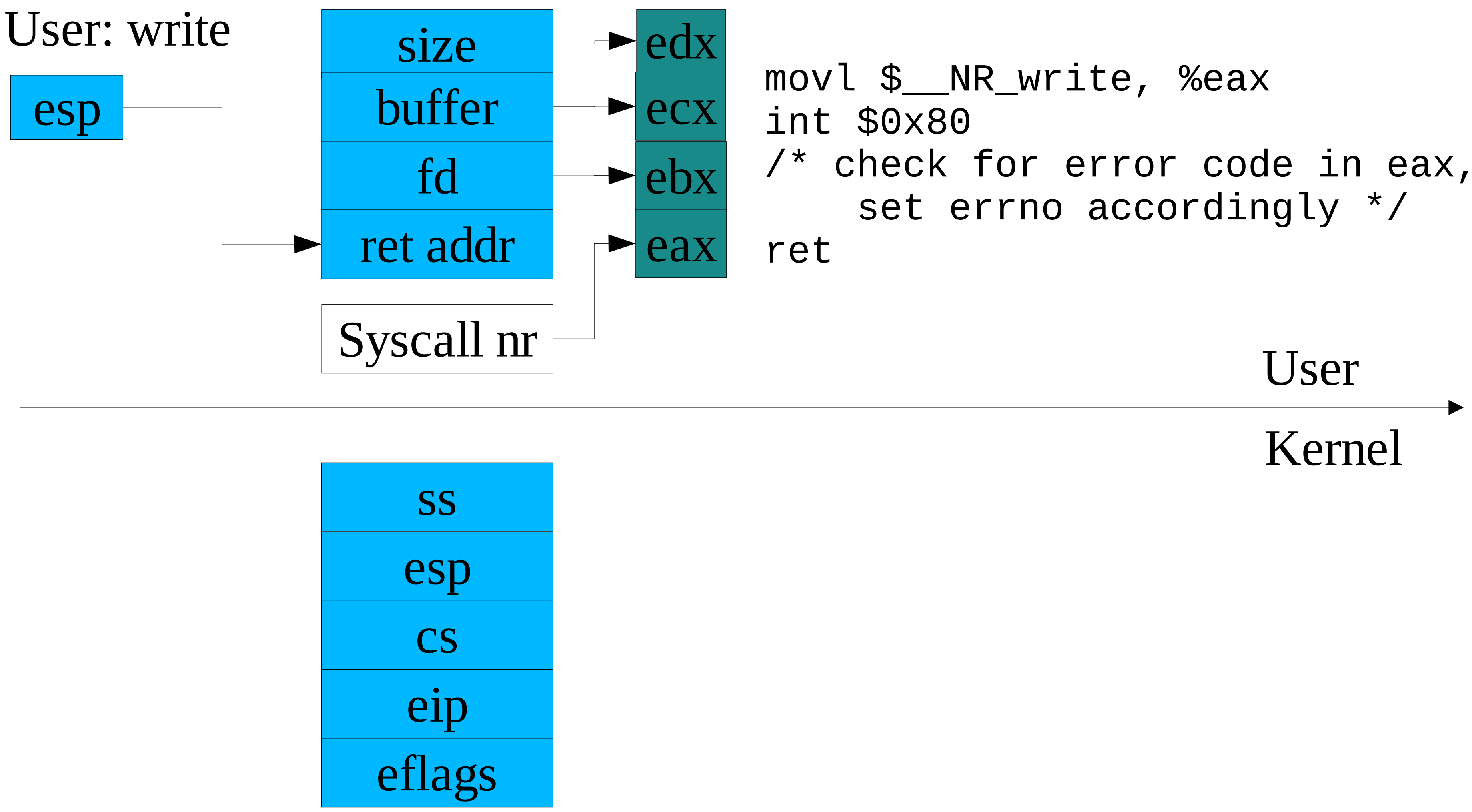
Syscall (16)

Kernel



```
pushl %eax
pushl %ebp
pushl %edi
pushl %esi
pushl %edx
pushl %ecx
pushl %ebx
call *sys_call_table(,eax,4)
movl %eax, EAX(%esp)
popl %ebx
popl %ecx
popl %edx
popl %esi
popl %edi
popl %ebp
popl %eax
iret
```

Syscall (17)



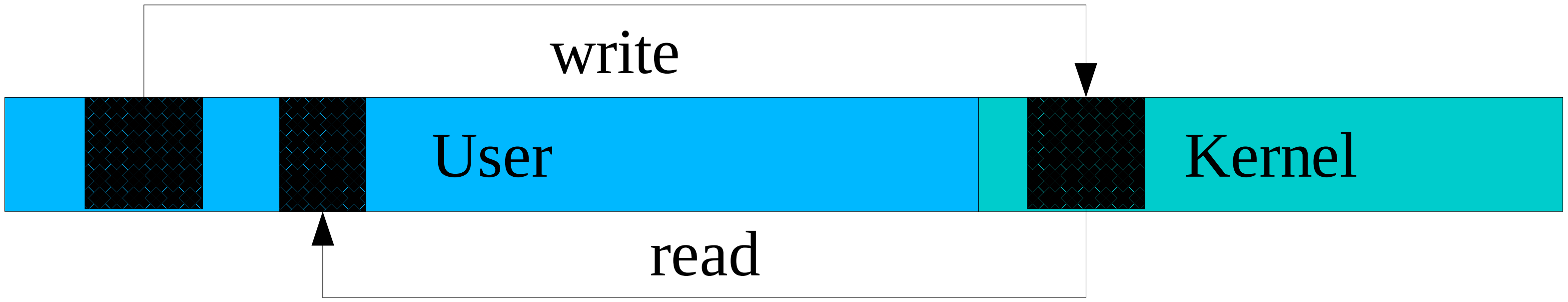
```
movl $__NR_write, %eax
int $0x80
/* check for error code in eax,
    set errno accordingly */
ret
```

What is missing

- check for valid sycall number (label `badsys`)
- system call tracing (label `tracesys`)
  - Send signal to tracing process before and after system call execution
- bottom half handling (label `handle_bottom_half`)
- scheduling (label `reschedule`)
- signal delivery (label `signal_return`)
- copy in/out (access to user space)

Copy in/out

- Kernel needs access to user space
  - Copy data into the kernel (e.g. `write` copies data from user space into kernel buffers
  - copy data to user space (e.g. `read` copies data from kernel buffers to user space)

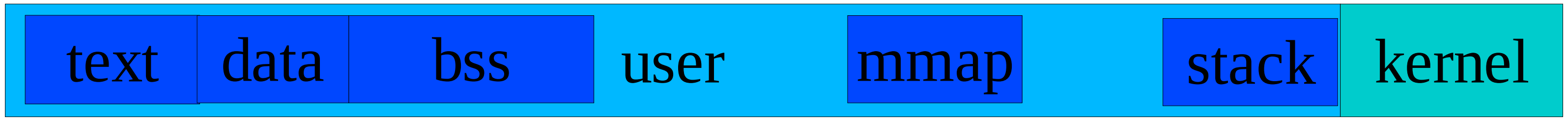


Copy in/out (2)

- Problem:
  - kernel can't trust the user
    - buffer argument of '`write`' could point to an invalid memory area (leading to a kernel crash) or into kernel space allowing the user to access privileged information
  - kernel has to check parameters
    - for pointers into the kernel (a simple compare)
    - To ensure the provided buffer is valid

Buffer validation

- validate buffer before accessing it
  - simply walk vma list of current process
  - 99,9999...% of all buffer addresses are valid
  - Unnecessary overhead



Buffer validation (2)

- validate buffer range before accessing it
  - Check if boundaries are in user space
  - Do NOT check if region is backed
- access the buffer and handle invalid accesses
  - kernel contains exception table (pair of exception instruction address and fixup address)
  - exception handler checks exception table and jumps to fixup address if kernel raises an (unhandled) exception

Copy in/out code

```
__get_user_1:
movl %esp,%edx
andl $0xffffe000,%edx                # get tcb of current process
cmpl addr_limit(%edx),%eax           # check for pointer into kernel space
jae bad_get_user                     # bad pointer
1:    movzbl (%eax),%edx              # access buffer, we get a pagefault
xorl %eax,%eax                   # if something is wrong
ret

.section __ex_table,"a"
.long 1b,bad_get_user                # exception table entry
.previous
bad_get_user:                        # we will return here after an invalid
xorl %edx,%edx                   # access
movl $-14,%eax                   # set eax to _EFAULT
ret                                  # return
```