System Calls

W4118 Operating Systems I

columbia-os.github.io

Credits to Jae

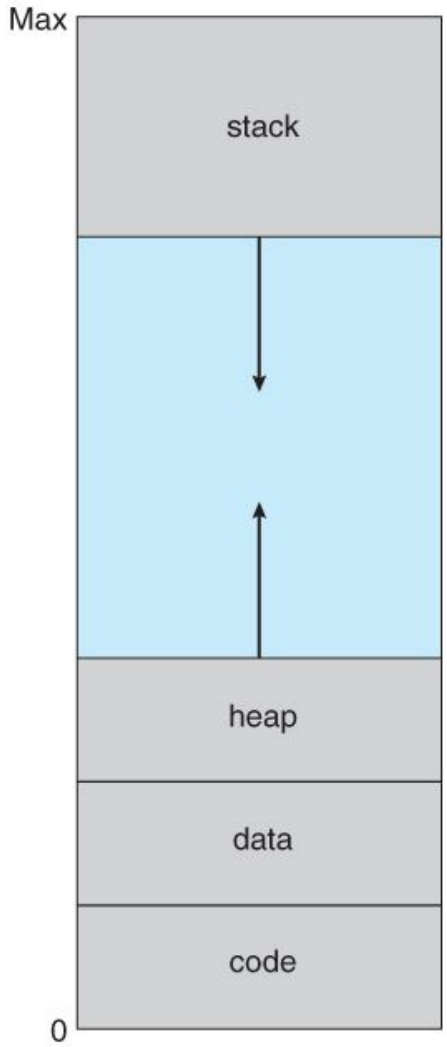
Allocating Memory

Run **`malloc.c`**, malloc() does not appear in the strace, why?

`brk()`, changes the location of the program break, which define the end of the process's data segment (i.e., the top of the heap)

`brk(NULL)` gets the current process break

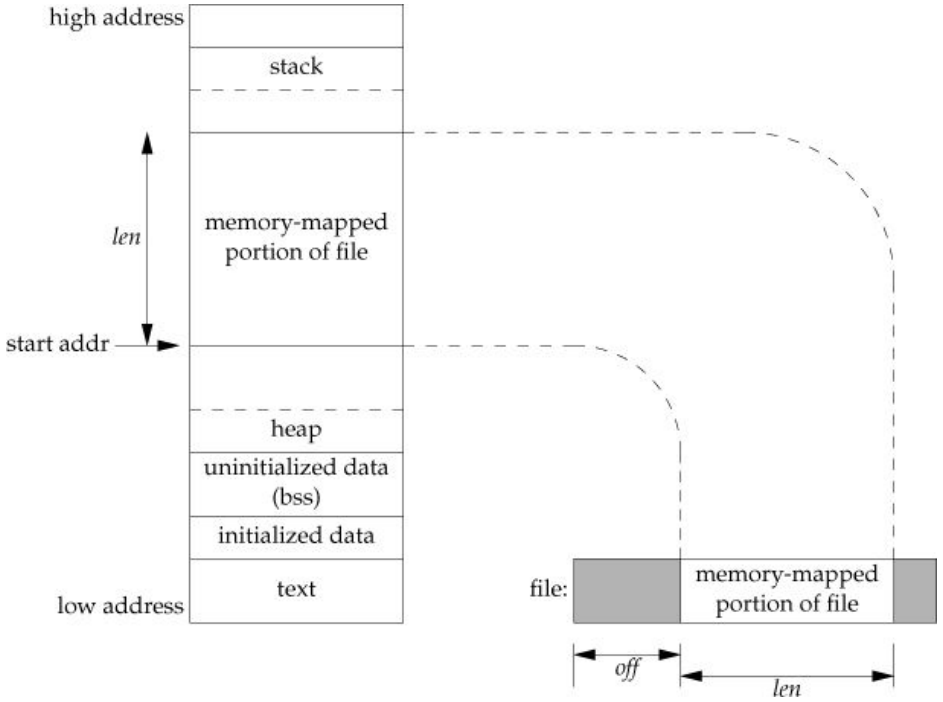
`brk(addr)` sets the break to **`addr`**



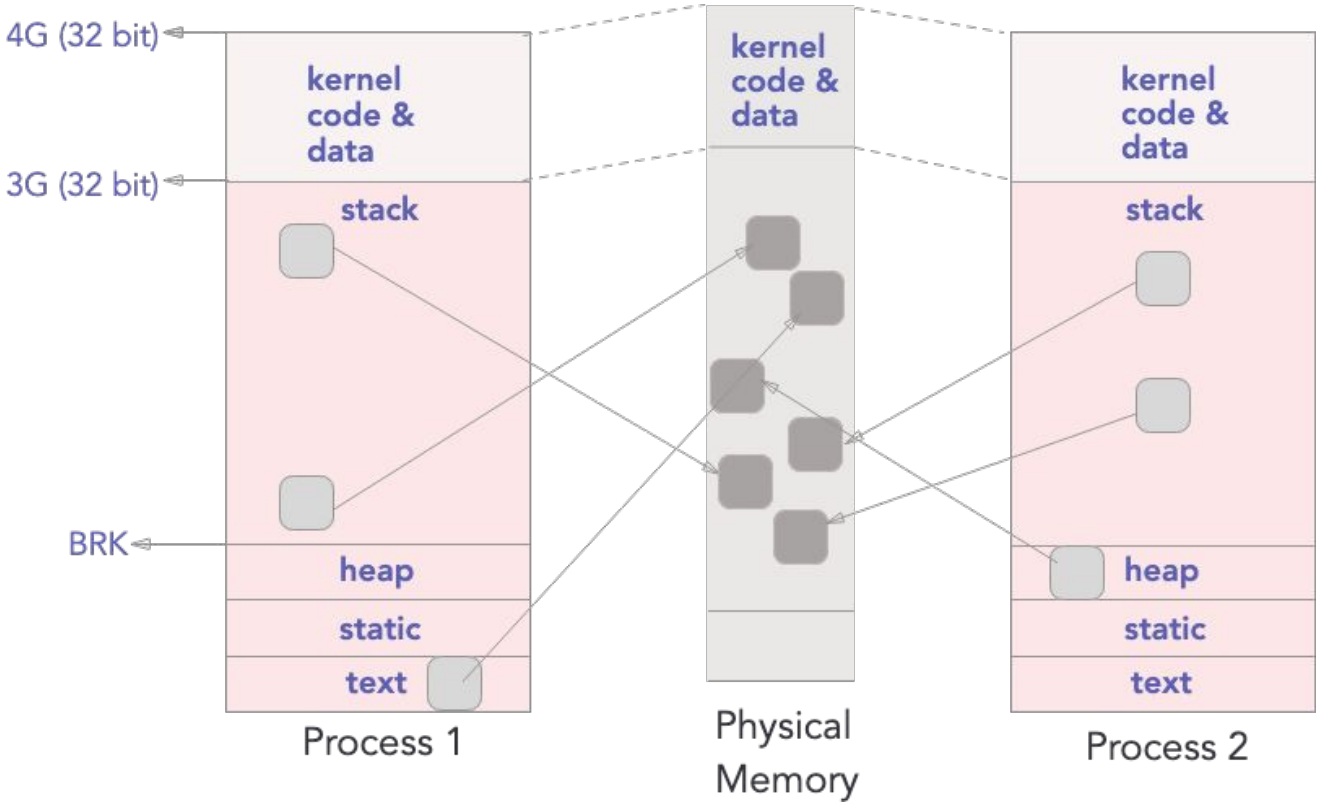
File-backed mappings

Program setup involved mapping in the C standard library:

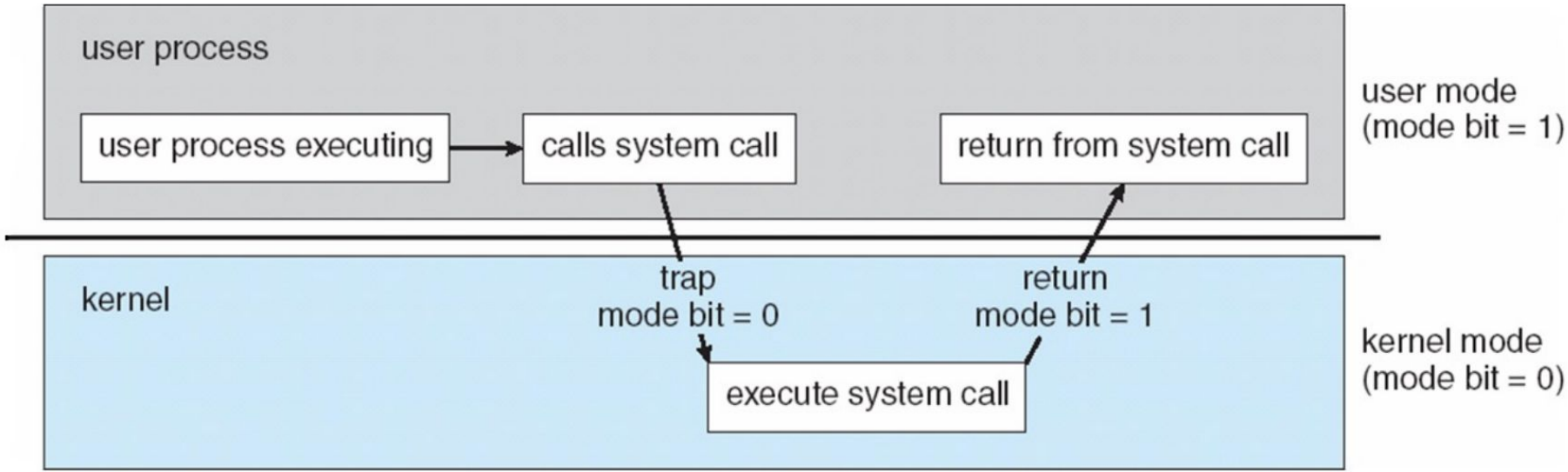
```
openat() = fd

mmap(..., fd, …)

close(fd)
```



Full Virtual Address Space



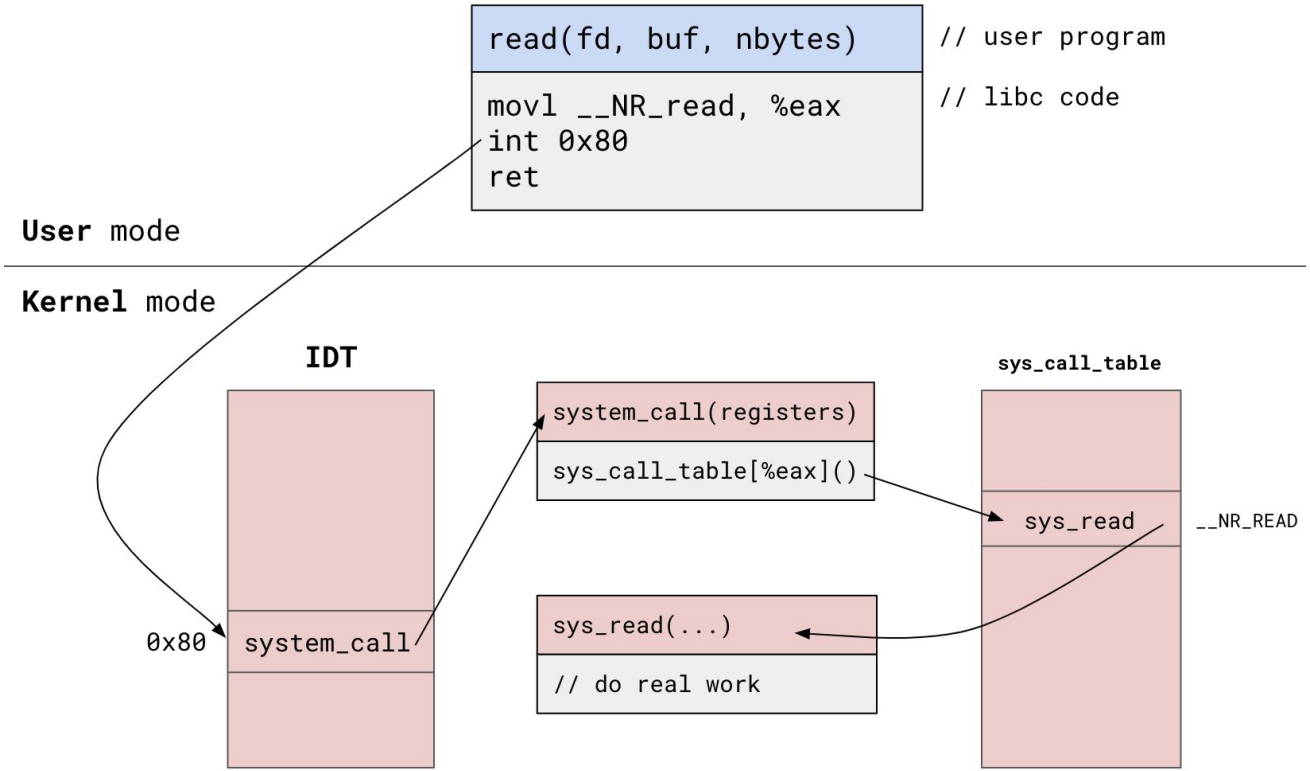
Processor Modes



Interrupts

- Hardware interrupts
  - asynchronous
  - e.g. network packet arrival, timer, key press, mouse click
- Exceptions/Faults
  - synchronous
  - e.g. dividing by zero, page fault
- Software interrupts
  - synchronous
  - x86 assembly int: raise software interrupt
  - e.g. syscall (int 0x80), debugger

Linux System Call Dispatch



Linux System Call Dispatch Notes

- `int 0x80` is how syscalls were invoked in 32-bit x86, e.g., x86-64 has a `syscall` instruction
- See system call handler and syscall dispatch under /arch/x86/entry
- See system call table here
- See `sys_read()` implementation in /fs/read_write.c
- Check vdso for even faster “system calls”

System Call Parameters

- Syscall parameters are passed via registers
  - Max arg size is the register size
  - Use struct pointer to pass in more/larger arguments (e.g. struct sigaction)

Need to validate memory! Why?

System Call Parameters

- Syscall parameters are passed via registers
  - Max arg size is the register size
  - Use struct pointer to pass in more/larger arguments (e.g. struct sigaction)

Need to validate memory!

Example, `read()/write()`: What if the buffer actually points to kernel memory?

- Pointer points to a region of memory in user-space, not kernel-space.
- If reading/writing/executing, memory must be marked readable/writable/executable accordingly

`copy_to_user()`
`copy_from_user()`

Why not to write a system call?

Why not to write a system call?

- You use a syscall number
- You need to maintain it forever
- You need to register and support it for each architecture
- Not easily usable from scripts or the filesystem
- You cannot maintain it easily outside the kernel tree
- It’s an overkill!

Alternatives:

- Use a special file and manipulate it instead