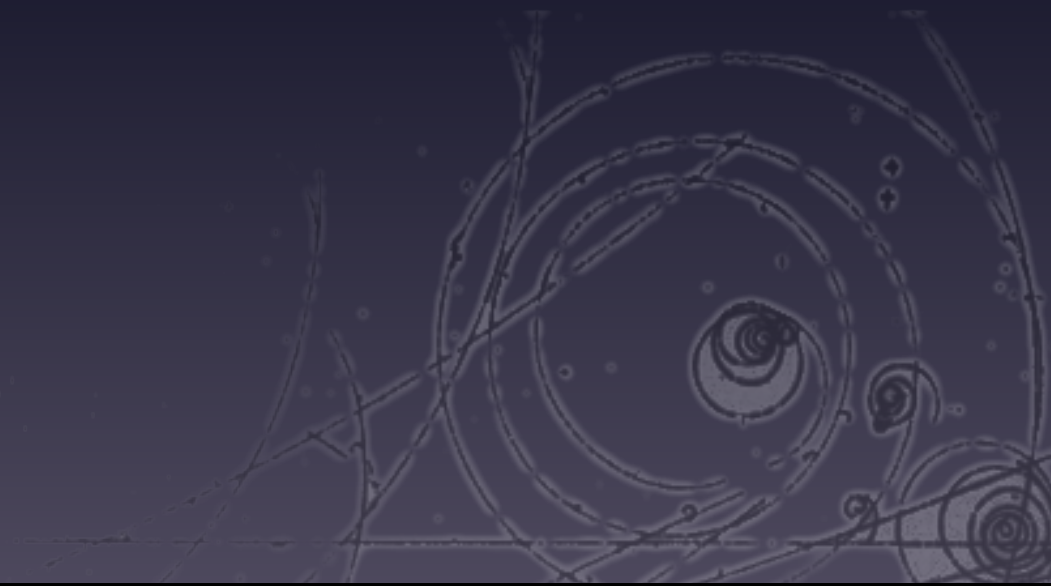
Today's slides are here:

git clone `https://github.com/Nils-TUD/nolibc.git`

(I'll update that now and then.)



Advanced Systems Programming
Living Without a Runtime

POSIX, libc, libstdc++, ...

Nils Asmussen Marcus Hähnel
Julian Stecklina Torsten Frenzel

October 13, 2020

Motivation

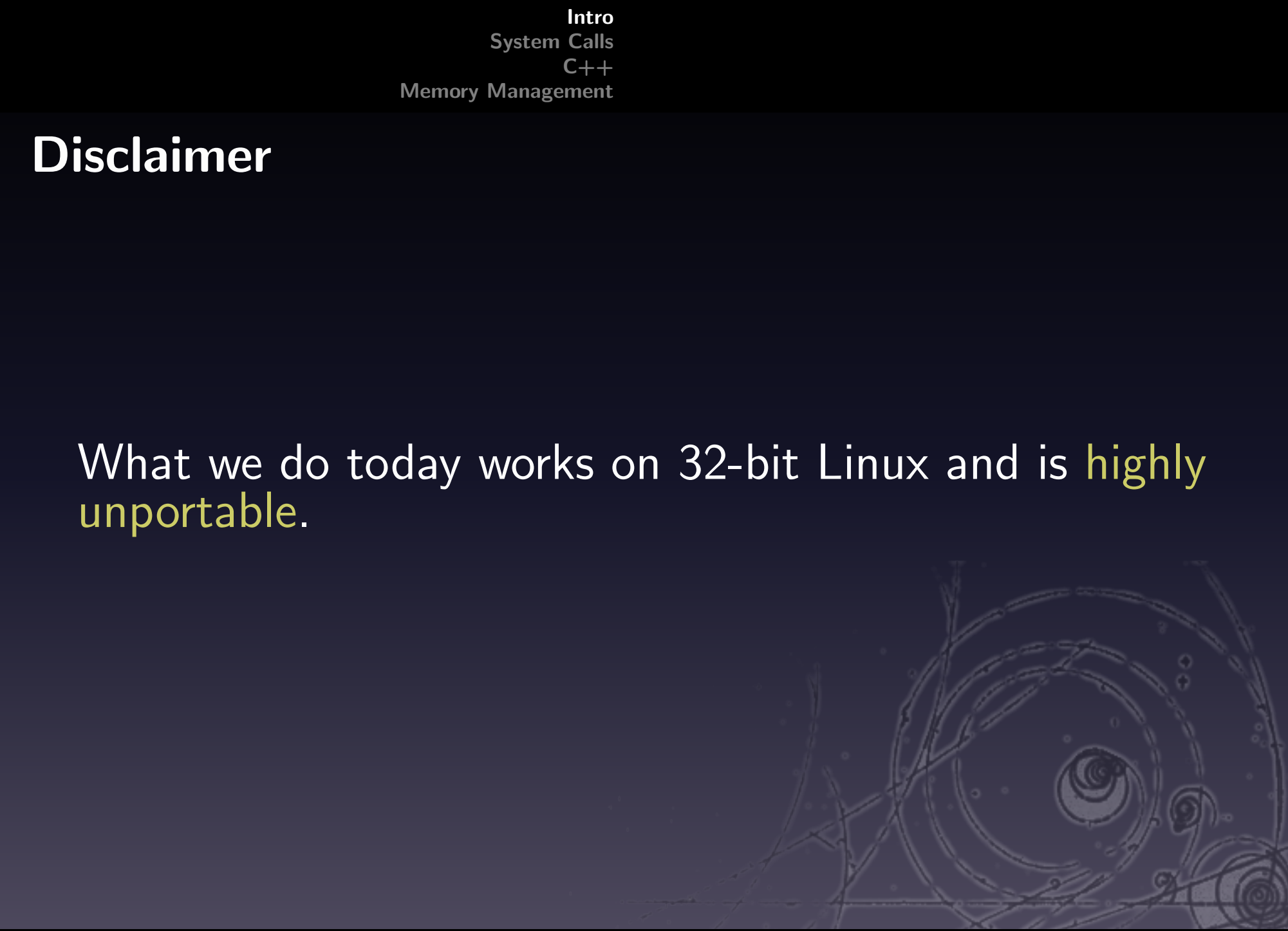
The system programmer sometimes operates in restricted environments without runtime support:

- boot code,
- kernel code,
- runtime library,
- ...

But what needs runtime support in C/C++?

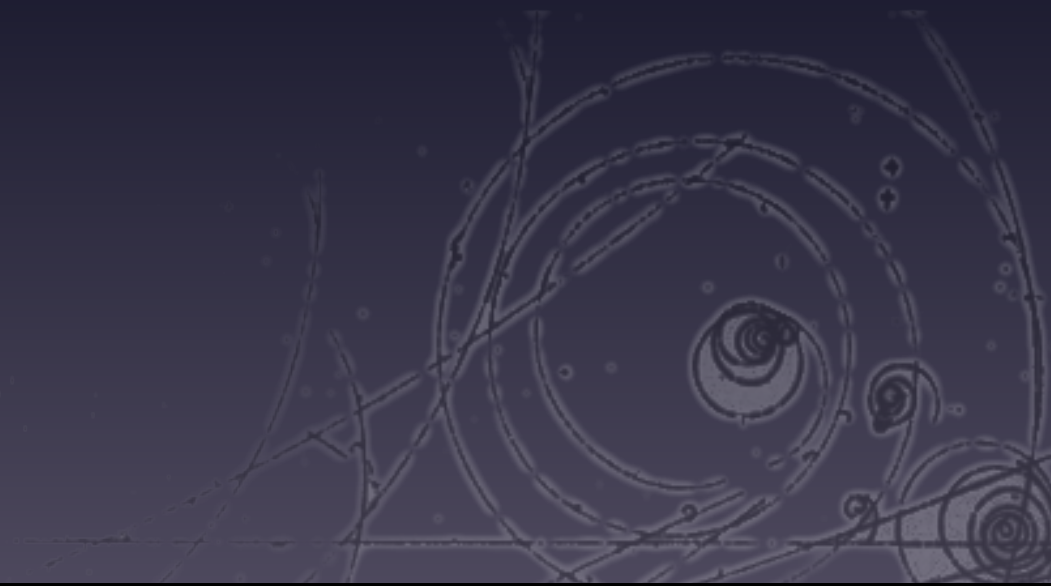
Disclaimer

What we do today works on 32-bit Linux and is highly unportable.

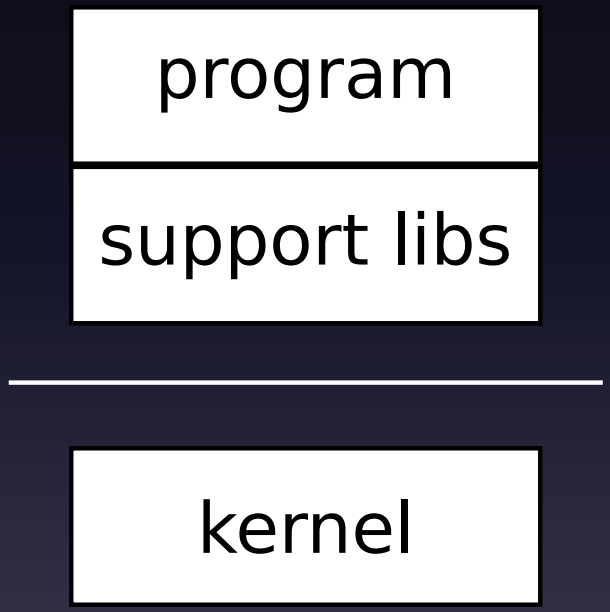


Plan

1. Hello World!
2. `wc -l`
3. `sort`
4. `malloc`



C Program Environment



- The C/C++ program expects POSIX interface.
- The kernel provides specific system calls.
- Libraries (libc, libstdc++, ...) bridge the gap.

Program Startup

Exercise

Write a C/C++ program `empty/main.cc` that does nothing.
Compile and link it with `-nostdlib`. Make it link!
What happens when you run it?

Try to output "Hello World" with `puts` or `printf`. Why does it fail?

- Use the prepared `Makefile`!

git clone `https://github.com/Nils-TUD/nolibc.git`
(I'll update that now and then.)

Program Startup

Exercise

Write a C/C++ program `empty/main.cc` that does nothing.
Compile and link it with `-nostdlib`. Make it link!
What happens when you run it?

Try to output "Hello World" with `puts` or `printf`. Why does it fail?

- Use the prepared `Makefile`!

git clone `https://github.com/Nils-TUD/nolibc.git`
(I'll update that now and then.)

You have to provide a function `_start` (`extern "C"`)

What system calls are interesting?

Exercise

Write a normal C++ program `hello/main.cc` that prints "Hello World" to `stdout`.
Using `strace` find out what system calls are used for output and program shutdown.

System Calls

System calls are wrapped by the libc into C functions. Linux programs can use `int $0x80` to trap into the kernel[1]. The syscall number is placed in EAX.
The parameters are in EBX, ECX, EDX, ESI, EDI, EBP (in this order). The result is in EAX.

[1] There is also `sysenter/syscall`.

Inline Assembler Recap

```
asm volatile ( "instr1 \n"
               "instr2 \n"
               : /* output, modify */
               : /* input */
               : /* clobber */
             );
```

Output Constraints

"+a"(v1), "=b"(v2)
+ modify, = output

Input Constraints

"a" "b"

(see GCC documentation "Machine Constraints")

Your First Syscall

Exercise / Recap

Call the `getpid` system call in `getpid/main.cc` directly and print the result.

- include files are in `/usr/include`
- **#include** <sys/syscall.h>

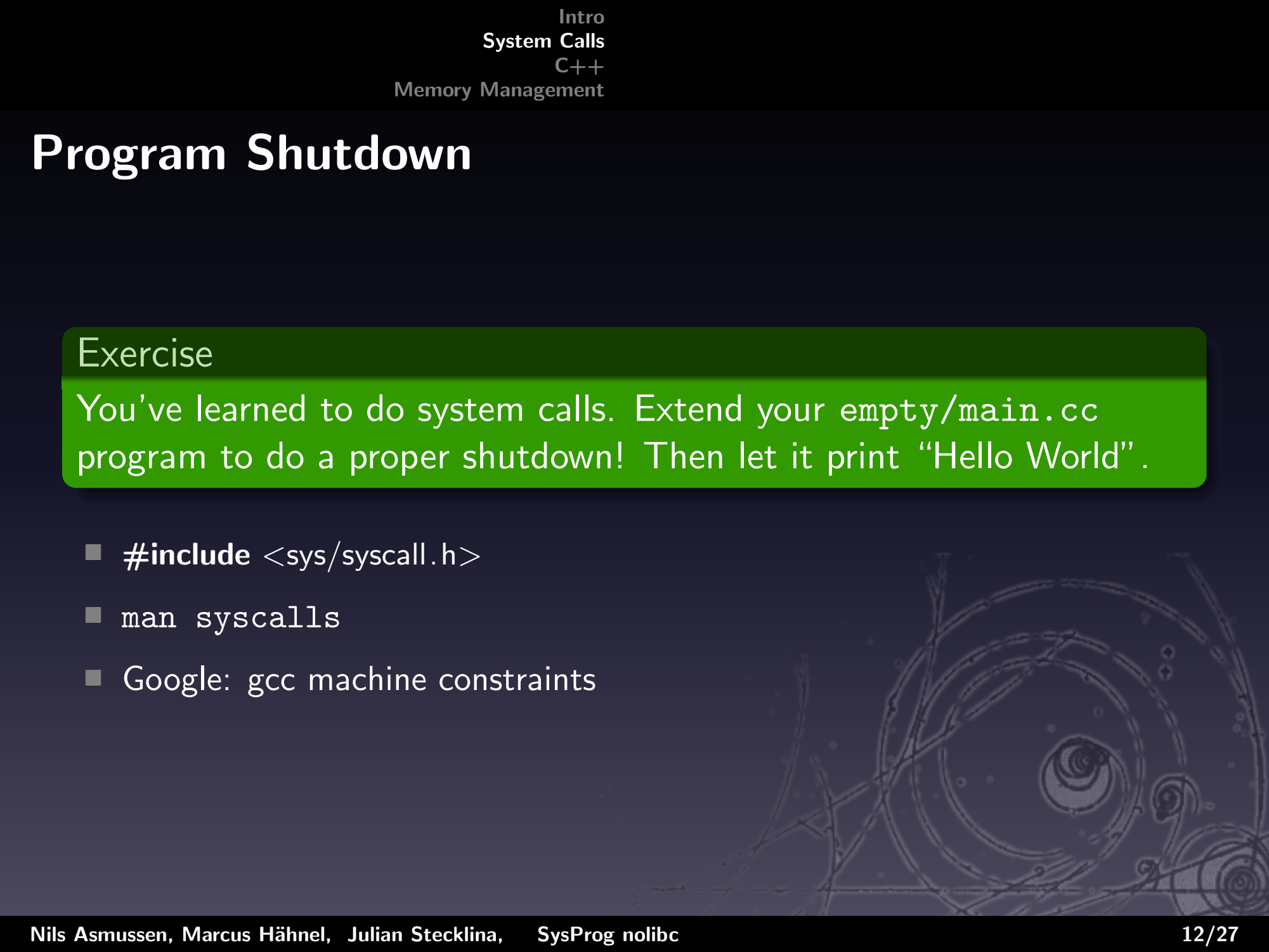
```
asm volatile ( "int $0x80"
               : "+a" (v)
               : /* input */
               );
```

Program Shutdown

Exercise

You've learned to do system calls. Extend your `empty/main.cc` program to do a proper shutdown! Then let it print "Hello World".

- **#include** <sys/syscall.h>
- `man syscalls`
- Google: gcc machine constraints



Program Startup - Done Right

start.S

C-Functions usually set up a new stack frame. This is not expected for _start which is just jumped to and not called. Write a file `start.S` that provides a simple function calling main (without parameters) and then exit

Program Startup - Done Right

start.S

C-Functions usually set up a new stack frame. This is not expected for _start which is just jumped to and not called. Write a file `start.S` that provides a simple function calling main (without parameters) and then exit

Return values of c functions are in eax, arguments are passed via stack

C++ Constructors

Exercise

Check if your `empty` program executes constructors for global instances, e.g. by writing a class Foo with a constructor that prints "Hello World" and a global instance of it.
What is a good workaround?

C++ Constructors

Constructors of global instances are called by the runtime prior to `main()` in *undefined* order. The "construct on first use" idiom can help:

```
Foo &get_foo()
{
  static Foo x;
  return x;
}
```

Constructor will be executed on first call. Might need `-fno-threadsafe-statics`.

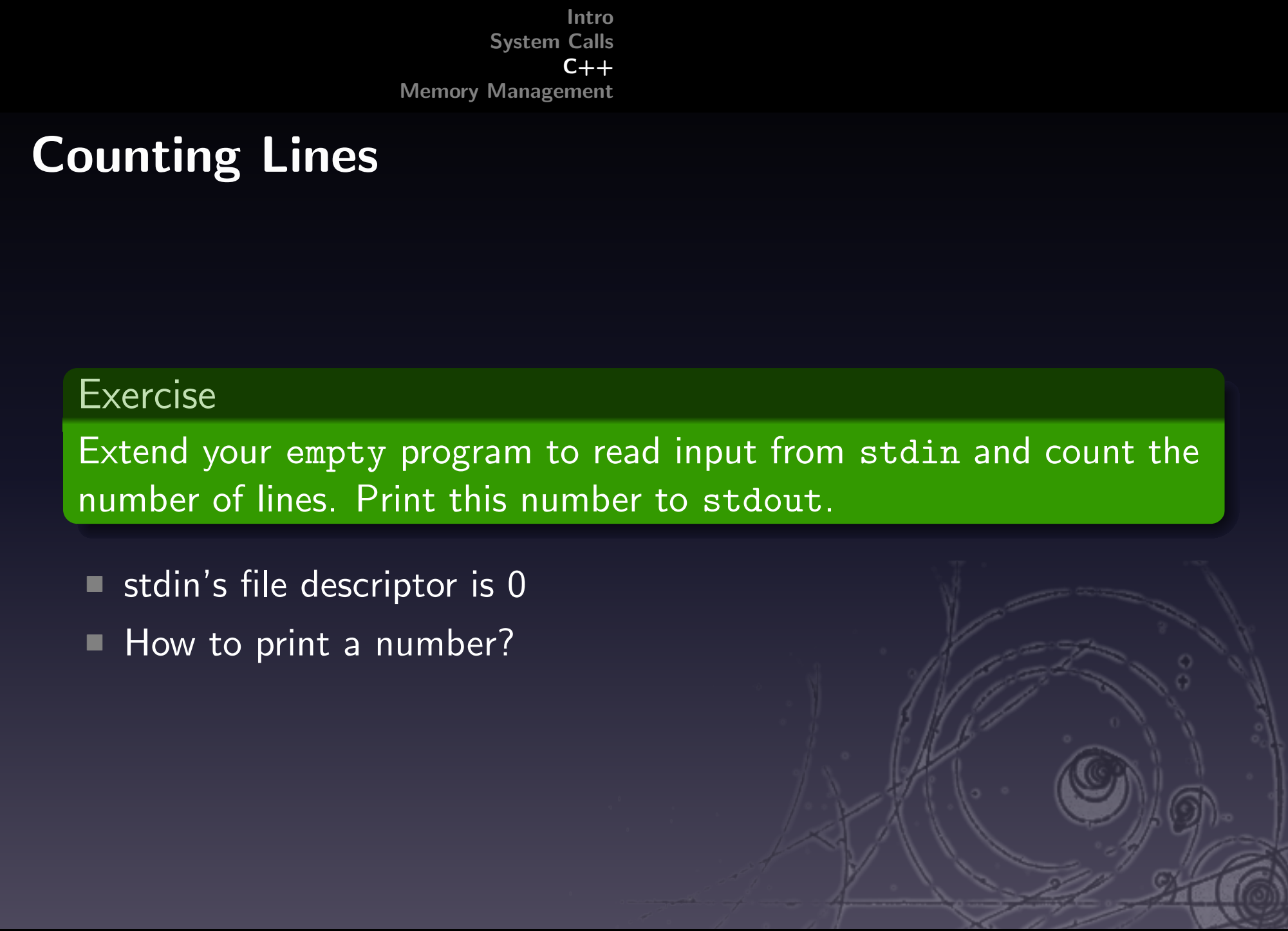
The real deal:
`http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html`

Counting Lines

Exercise

Extend your `empty` program to read input from `stdin` and count the number of lines. Print this number to `stdout`.

- stdin's file descriptor is 0
- How to print a number?



Counting Lines

We don't have `gets`. Use `read` system call to read blocks of memory and find line endings (`'\n'`) yourself. We are done reading when `read` returns an error or zero.

Print numbers by:

1. Divide repeatedly by the base (10). Store remainders.
2. The remainders in reverse order are your number as string.

We might need some memory management to turn this into a `sort`. Let's see what we can do about that.

Memory Management

C++'s `new` does two things:

- allocates memory using the standard C backend (`malloc`),
- initializes the object using its constructor.

`delete` does the reverse:

- calls the destructor of the object,
- frees the memory using the standard C backend (`free`).

Part is done by the compiler, part by the runtime.

Memory Management

C++'s `new` does two things:

- allocates memory using the standard C backend (`malloc`),
- initializes the object using its constructor.

`delete` does the reverse:

- calls the destructor of the object,
- frees the memory using the standard C backend (`free`).

Part is done by the compiler, part by the runtime.

Memory Management

C++'s `new` does two things:

- allocates memory using the standard C backend (`malloc`),
- initializes the object using its constructor.

`delete` does the reverse:

- calls the destructor of the object,
- frees the memory using the standard C backend (`free`).

Part is done by the compiler, part by the runtime.

Overloading new

The standard definition has one argument:

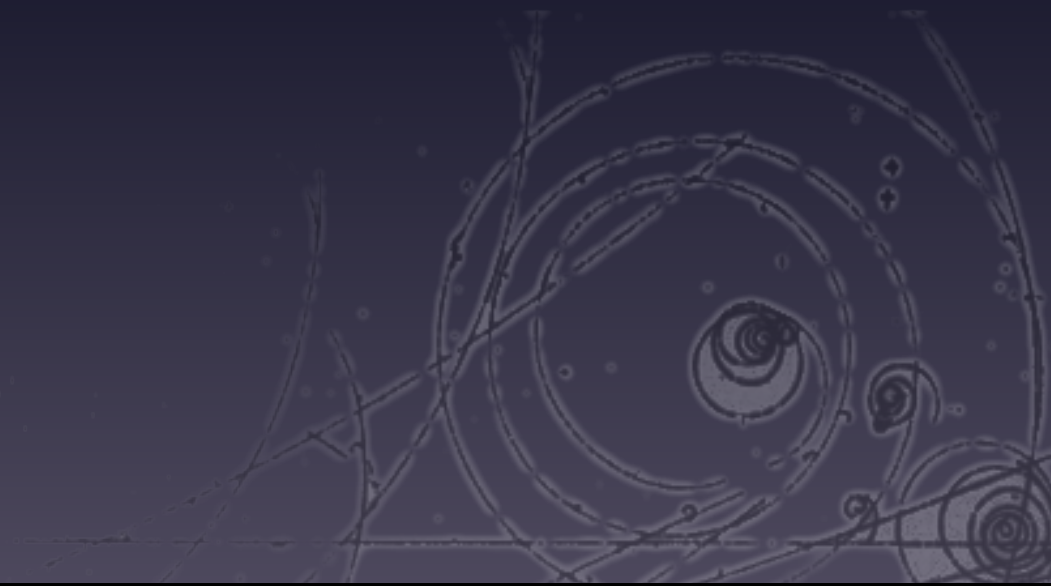
```
void* operator new (size_t size)
{
    /* Do something */
    return aPtr;
}
```

Versions with multiple parameters are also possible:

```
void* operator new (size_t size, void *p)
{
    /* This is the so-called placement new */
    return p;
}
```

Overloading delete

```
void operator delete (void *p)
{
    /* Do something */
}
```



Dynamic Memory Management

Need to maintain a pool of free memory to satisfy allocation requests:

- bitmap
- free list

How to determine the memory block size on deallocation?

- extra data structure indexed by pointer (list, tree, hashtable)
- colocate information with data block
- fixed block size

How to handle exceptional situations (OOM, double free, corrupted pointer)?

Memory Management

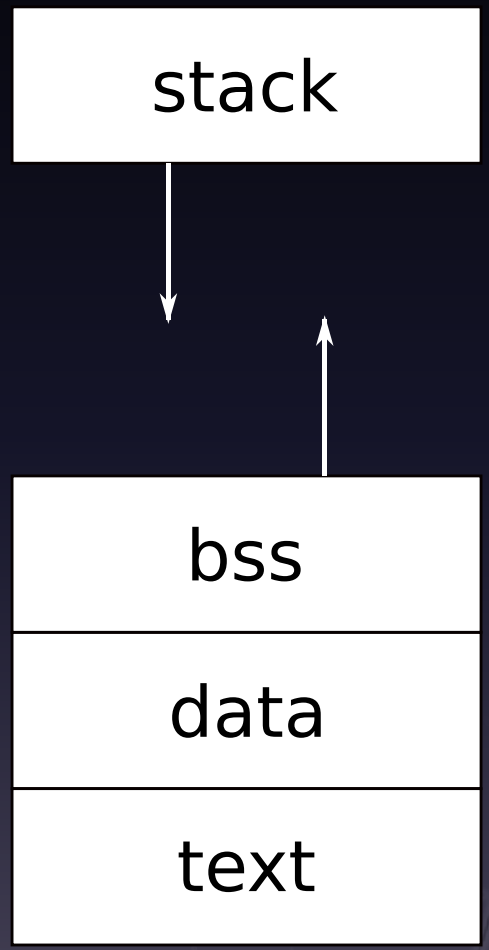
Perhaps, you noticed that there is no `malloc`/`new`. We have to use lower-level functions:

sbrk

- the interface from way back
- extend your “break” (end of bss)

mmap

- the modern way from the introduction of virtual memory in UNIX
- allocate memory where you want


stack
bss
data
text

Allocating Memory

Exercise

Figure out in `teststuff.cc` how to allocate memory using `sbrk` and `mmap`.
Which system calls are called by the libc functions (`strace`)?
Extend your `empty` program with a trivial `malloc`/`new` using one of those.
`delete`/`free` can be a no-op for now.

- don't miss "NOTES" in the man pages
- MAP_ANONYMOUS
- page size is 4096 bytes

Sorting Lines

Exercise

Extend `empty` to read lines from `stdin` and print them sorted to `stdout`.

- How to read lines instead of data blocks?
- Use an idiot-proof sorting algorithm!
- You still have your list implementation, if you need one.

Proper Memory Management

Extend your memory management to properly handle delete/free!

- use a bitmap (array of bool) to handle free space
- store size of block in-place

```
void *new(size_t size) {
    /* ... get a free memory block ... */
    size_t *h = reinterpret_cast<size_t *>(p);
    h[0] = size_of_p;
    return h+1;
}
```

We didn't cover . . .

- Run-time type information (**dynamic_cast**<>)
- Exceptions
- . . .