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

Nils Asmussen

September 22, 2026

Motivation

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

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

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++?

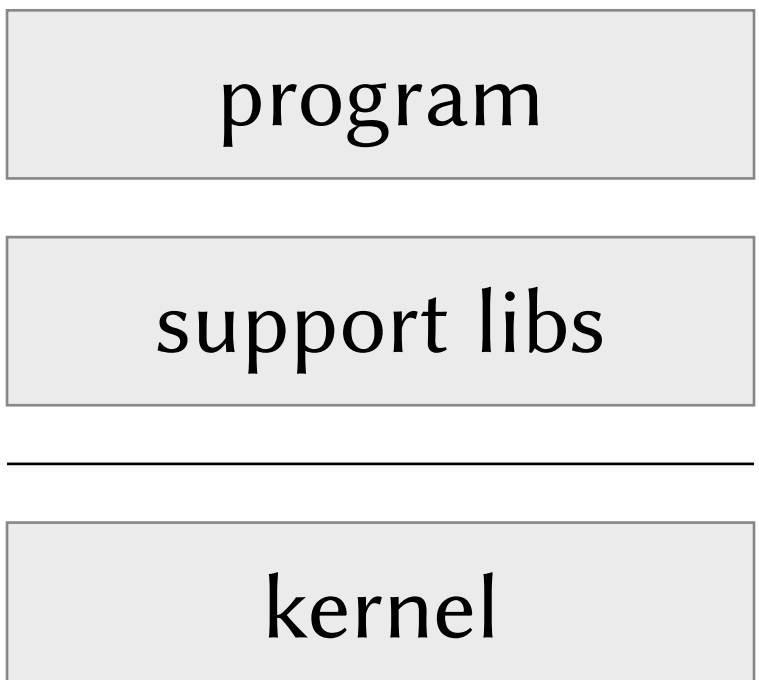
What we do today works on **x86-64 Linux** and is highly unportable!

Plan for Today

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

Hello World Without Runtime

C Program Environment



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

Program Startup

You have not cloned the repo yet?

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

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 `printf`. Why does it fail?

What System Calls are Interesting?

Exercise

- Write a normal C/C++ program `hello/main.cc` that prints `"Hello World"` to `stdout`.
- Use `strace` to 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 use the `syscall` instruction to trap into the kernel
- The syscall number is expected in `RAX`
- Parameters are passed in `RDI`, `RSI`, `RDX`, `R10`, `R8`, and `R9` (in this order)
- The result is written to `RAX`

Inline Assembler Recap

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

- Output/modify constraints: `"+a"(v1), "=b"(v2)`
- Input contraints: `"a"(v1), "b"(v2)`
- Dedicated letters for some registers (a = RAX, b = RBX, etc.)
- See GCC documentation “machine constraints”

Your First Syscall

Exercise

- Call the `getpid` system call manually in `getpid/main.cc`
- Print the result

- include files are in `/usr/include`
- `#include <syscall.h>`
- Defines constants such as `SYS_read`
- `asm volatile ("syscall" : "+a"(v));`

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"`

- `man syscalls`
- https://syscalls.mebeim.net

Program Startup – Done Right

- C functions set up a new stack frame
- This is not expected for `_start` (kernel performs jump, no call)

Program Startup – Done Right

- C functions set up a new stack frame
- This is not expected for `_start` (kernel performs jump, no call)

Exercise

- Create new file `start.S`
- Provide `_start` and call `main` (without arguments)
- Call `exit` with return value from `main`
- Note: return values are in `RAX`, arguments are passed in register `RDI`, `RSI`, `RDX`, `R10`, etc. (same ABI as for syscalls)

Short Detour: C++ Constructors

C++ Constructors

Exercise

- Check if your `empty` program (`-nostdlib`) executes constructors for global instances
- For example, create `class Foo` with constructor that prints `"Hello World"` and create a global instance of it
- What is a good workaround?

C++ Constructors

- Constructors of global instances are called by the runtime before `main`
- The order is `undefined` with compilation units!

C++ Constructors

- Constructors of global instances are called by the runtime before `main`
- The order is `undefined` with compilation units!
- The “construct on first use” idiom can help:

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

C++ Constructors

- Constructors of global instances are called by the runtime before `main`
- The order is `undefined` with compilation units!
- The “construct on first use” idiom can help:

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

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

C++ Constructors

- Constructors of global instances are called by the runtime before `main`
- The order is `undefined` with compilation units!
- The “construct on first use” idiom can help:

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

- Constructor will be executed on first call
- Might require `-fno-threadsafe-statics`
- The real deal:
  http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html

Let’s Build a Simple `wc -l`

Counting Lines

Exercise

- Extend your `empty` program to read input from `stdin`
- Count the number of lines
- Print result to stdout

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

Counting Lines

- We don't have `gets` to read a line
- Use `read` syscall to read blocks of memory (or single characters)
- Find line endings (`'\n'`) yourself
- We are done reading when `read` returns an error or 0

Counting Lines

- We don't have `gets` to read a line
- Use `read` syscall to read blocks of memory (or single characters)
- Find line endings (`'\n'`) yourself
- We are done reading when `read` returns an error or 0

Print numbers by:

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

Let’s build a simple sort

Memory Management

- C++‘s new does two things
  1. allocates memory using the standard C backend (`malloc`).
  2. initializes the object using its constructor.
- `delete` does the reverse
  1. calls the destructor of the object.
  2. frees the memory using the standard C backend (`free`).

Memory Management

- C++'s `new` does two things
  1. allocates memory using the standard C backend (`malloc`).
  2. initializes the object using its constructor.
- `delete` does the reverse
  1. calls the destructor of the object.
  2. 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) {
  void *addr = /* allocate memory */
  return addr;
}
```

Overloading new

The standard definition has one argument:

```
void *operator new(size_t size) {
  void *addr = /* allocate memory */
  return addr;
}
```

Versions with multiple arguments are possible:

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

Overloading delete

```
void operator delete(void *addr) {
  /* free memory */
}
```

Dynamic Memory Management

- Need to maintain pool of free memory to satisfy allocation requests:
  - bitmap
  - free list

Dynamic Memory Management

- Need to maintain 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, hashmap)
  - colocate information with data block
  - fixed block size

Dynamic Memory Management

- Need to maintain 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, hashmap)
  - colocate information with data block
  - fixed block size
- How to handle exceptional situations (OOM, double free, corrupt pointer)?

Allocating Memory

Where do we get additional memory from?

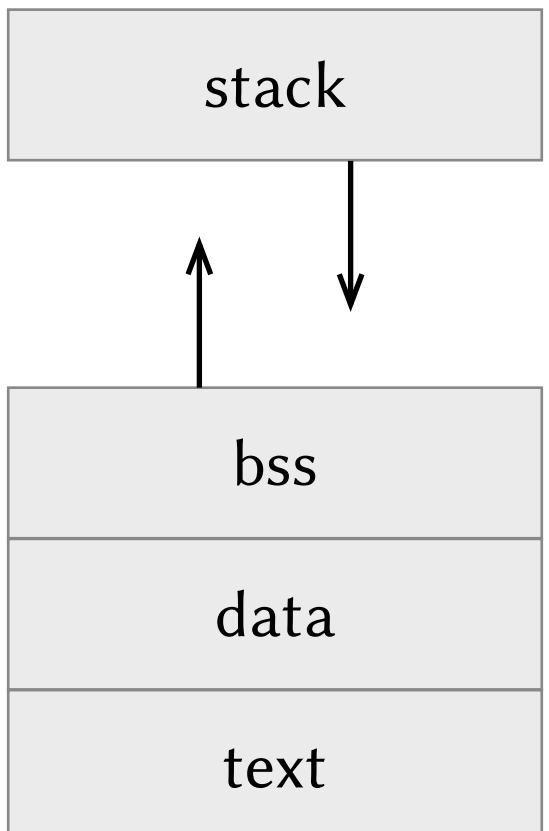
`sbrk`

- The interface from way back.
- Extends your “break” (end of BSS).

`mmap`

- The modern way from the introduction of virtual memory in UNIX.
- Allocate memory anywhere.

Memory Layout



Allocating Memory

Exercise

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

- Use `SYS_mmap2`, not `SYS_mmap`
- Page size is 4096 bytes on x86-64

Sorting Lines

Exercise

- Extend `empty` to read lines from `stdin`
- Print them sorted to `stdout`

- Use an idiot-proof sorting algorithm!
- How to read lines instead of data blocks?
- How can we support arbitrary line counts and lengths?

Proper Memory Management

Exercise

- Extend your memory management to implement `delete`/`free`

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

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

More Topics

We did not cover:

- Runtime type information (`dynamic_cast<>`)
- Exceptions
- ...