Request from Architecture WG

- Dig into the features needed to run an application that does nothing.
  - execve
  - context switch
- is the proposed executable minimal?

```
int main() {
    int a,b=1,c=2;


    a = b + c;
    while (1);
    return a;
}
```

Minimal application

Build the c app, effectively produces something different from expected.

- dynamic executable: loads interpreter aka dynamic loader.
  - dynamic loader loads up shared objects
  - maps shared objects into the process address space
- libc initialization aka **`crt0.o`**:
  - despite common believes, any given executable does not start at **`main()`**, it start at **`_start`**.
  - malloc initialization (even if not used)
  - infrastructure for **`.init`**, **`.ctors`**, and **`.init_array`**

Rewrite the minimal

First attempt, rewrite in assembly

- + Complete control of the code
- + solves both problems
- - Difficult port to other platforms (is it needed?)
- - Difficult to understand
- - Difficult to use, it needs an additional linker configuration file to be built.

```
.global _start
.section .text
_start:
        sub sp, sp, 0x10
        ldr w1, [sp, #0x04]
        ldr w0, [sp, #0x08]
        add w0, w1, w0
        str w0, [sp, #0x0c]
loop:
        b loop
        mov x8, 0x5d
        mov x0, #0
        svc #0
```

ELISA WORKSHOPS

Keep C, but make it static

- `--static` builds the file static
- Build the C file as static, save the unnecessary dynamic linker complexities:
  - shared objects load from file
  - shared objects mapping into the process address space.
- library crt0.o file is still linked to the executable, ant it continues to provide unnecessary complexities.

Keep C, and get rid of crt0.o

- Typically is done **`-nostartfiles`**
- It produces same executable characteristics as the assembly, but the source needs to be slightly modified.

```
int _start() {
    int a,b,c;


    a = b + c;
    while (1);
    return a;
}
```

Feature analysis: execve syscall

- Formally executed by the process we are analyzing, but it is not part of the user written code.
- This syscall can be generally analyzed, no arch specific contents.

Feature analysis: Context Switch

- context switch happens for userspace:
  - at syscall return
  - at interrupts return
- usecase has no syscall, except the execve that starts it.
- interrupt code is highly architecture dependent