Everything you ever wanted to know about “hello, world”*

(*but were afraid to ask)

Brooks Davis
SRI International

February 4th, 2017
FOSDEM 2017
Brussels, Belgium
Introduction

• Understanding the process ABI:
  • So I can change it!

• CHERI CPU is a MIPS64 compatible CPU
  • With C compatible memory safety extensions
  • We replace integer pointers with unforgeable capabilities
  • Prevent buffer overflows and make compartmentalization cheap
#include <stdio.h>

main()
{
    printf("hello, world\n");
}

K&R: The C Programming Language
K&R: The C Programming Language

#include <stdio.h>

void
main(void)
{
    printf("hello, world\n");
}
Today’s version

```c
int main(void)
{
    const char hello[] = "hello, world";

    printf("%s %d\n", hello, 123);

    return (0);
}
```
void main(void)
{
    const char hello[] = "hello, world 123\n";

    write(1, hello, sizeof(hello));

    exit(0);
}
Minimal (MIPS) assembly version

.text
.global __start
.ent __start
__start:
    li $a0, 1
dla $a1, hello
    li $a2, 17
    li $v0, 4
    syscall          # write(1, "hello, world 123\n", 17)
    li $a0, 0
    li $v0, 1
    syscall          # _exit(0)
.end __start

.data

hello:
    .ascii "hello, world 123\n"
Size comparison

• Assembly
  • Compiles to 9 instructions
  • Stripped binary less than 1K
    • Mostly ELF headers, MIPS ABI bits
• Minimal C
  • Stripped binary over 550K!
    • Mostly malloc() and localization
Program linkage

$ cc -static -o helloworld helloworld.o

$ ld -EB -melf64btsmip_fbsd -Bstatic \n-o helloworld /usr/lib/crt1.o \n/usr/lib/crti.o /usr/lib/crtbeginT.o \n-L/usr/lib helloworld.o \n--start-group -lgcc -lgcc_eh -lc \n--end-group \n/usr/lib/crtend.o /usr/lib/crtln.o
## Compiler runtime support

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| crt1.o | Contains `__start()` function which initializes process environment and calls `main()`.
| ctti.o | Entry points for old style `_init()` and `_fini()` functions.           |
| crtbegin.o | Declares `.ctor` and `.dtor` constructor and destructor sections.       |
| cttbeginS.o | Declares functions to call constructors and destructors.               |
| cttbeginT.o |                                        |
| cttend.o | NULL terminates `.ctor` and `.dtor` sections.                          |
| ctttn.o  | Trailers for `_init()` and `_fini()` functions.                        |

On FreeBSD: built in gnu/lib/csu and lib/csu/ARCH.
Execution lifecycle (notional)
Execution lifecycle (static)

fork() → waitpid() → execve() → readlink() → isetgid() → break() → mmap() → munmap() → mmap() → munmap() → munmap() → munmap() → munmap() → sysarch() → fstat() → ioctl() → write() → exit()
Code and images online

https://people.freebsd.org/~brooks/talks/fosdem2017-helloworld

or

execve()
exec_copyin_args()
sys_execve()
kern_execve()

namei() Resolve path

exec_check_permissions() Check that the file has the right permissions and open it.

exec_map_first_page() Map the header into kernel memory.
exec_elf64_imgact()

Defined with macros:

__CONCAT(exec_, __elfN(imgact))
(struct image_params *imgp)
exec_new_vmspace()

- `pmap_remove_pages()`
- `vm_map_remove()`
- Evict all page mappings from the address space

- `vm_map_stack()`
  Map a stack into the address space
exec_elf64_imgact()

elf_load_section() Map .text section into memory

elf_load_section() Map .data section into memory and create bss
kern_execve()

**exec_copyout_strings()**

**elf64_freebsd_fixup()**

Copy argv, envp, etc to the stack and adjust stack pointer.

**exec_setregs()**

Set initial register context to enter __start().

---

.text .data bss Stack
sys_execve()
Returning to userspace

- Stack is mapped into address space
- Program is mapped into address space
- Strings, argv, envp, signal handler, etc are on the top of the stack
- Register state is set up to call __start()
SCO i386 ABI stack

```
__start(char **ap, ...) {
    ...
    argc = *(long *) ap;
    argv = ap + 1;
    env = ap + 2 + argc;
    ...
```
Most cycles spent in malloc()
```c
void __start(char **ap)
{
    int argc;
    char **argv, **env;

    argc = * (long *) ap;
    argv = ap + 1;
    env = ap + 2 + argc;

    ...
}
```
__start__() 2/2

...  

handle_argv(argc, argv, env);
_init_tls();
handle_static_init_init(argc, argv, env);

exit(main(argc, argv, env));
}
_init_tls()

Most cycles spent in malloc()
_init_tls()

• Find the ELF auxargs vector

Elf.Addr *sp;
sp = (Elf.Addr *) environ;
while (*sp++ != 0) ;
aux = (Elf_Auxinfo *) sp;
function _init_tls()

- Find the ELF auxargs vector
- Use that to find the program headers
- Use those to find the PT_TLS section (initial values)
- Call \texttt{__libc_allocate_tls()} (as \texttt{rtld_allocate_tls()})
  - Allocates space
  - Copies initial values
- Set the TLS pointer

Uses JEMalloc, but JEMalloc uses TLS!
__start() 2/2

...

handle_argv(argc, argv, env);
_init_tls();
handle_static_init(argc, argv, env);

exit(main(argc, argv, env));

}
main()
vfprintf()
vfprintf()
__vfprintf()
Look up decimal point string.

`vfprintf()`

(`"%s", hello)`

(`" %d", 123)`

(`"\n"`)
__sprint()

New-line character found.
hello, world 123
__start()
exit()

- Call destructors registered with atexit()
- Flush any unflushed FILEs
- Call _exit()
Dynamic binary

_rtdl_relocate_nonplt_self(): RtlD relocates itself

Load and relocate libc

__start()
__start()
printf()
_mips_rtld_bind()
printf()
QUESTIONS?
Feedback requested

• Was the talk interesting and/or helpful?
• What didn’t make sense?
• What would you like have learned more (or less) about?

• brooks.davis@sri.com