Everything you ever wanted to know about “hello, world”*
(*but were afraid to ask)

Brooks Davis
SRI International

June 10, 2016
BSDCan 2016
K&R: *The C Programming Language*

```c
#include <stdio.h>

main()
{
    printf("hello, world\n");
}
```
K&R: The C Programming Language

#include <stdio.h>

void
main(void)
{
    printf("hello, world\n");
}
int main(void)
{
    const char hello[] = "Hello World!"
;
    printf("%s %d\n", hello, 123);

    return (0);
}

Today's version
Minimal C version

```c
void main(void)
{
    const char *hello[] = "hello, world\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, 12
li $v0, 4
syscall     # write(1, "hello, world\n", 13)
li $a0, 0
li $v0, 1
syscall     # exit(0)
.end __start

.data
hello:
  .ascii "hello, world\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 -melf64bsmip_fbsd -Bstatic \
-o helloworld /usr/lib/crt1.o \ 
/usr/lib/crti.o /usr/lib/crtbeginT.o \ 
-L/usr/lib helloworld.o \ 
--start-group -lgcc -lgcc_eh -lc \
--end-group \ 
/usr/lib/crtend.o /usr/lib/crtin.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()`.
| crt1i.o  | Entry points for old style `_init()` and `_fini()` functions.          |
| crtbeg.o | Declares `.ctor` and `.dtor` constructor and destructor sections.     |
| crtbegS.o| Declares functions to call constructors and destructors.               |
| crtbegT.o|                                                                         |
| crten.o  | NULL terminates `.ctor` and `.dtor` sections.                          |
| crtn.o   | Trailers for `_init()` and `_fini()` functions.                        |

Built in gnu/lib/csu and lib/csu/ARCH.
Code and images online

https://people.freebsd.org/~brooks/talks/bsdcan2016-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()
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()

efunc_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 entry __start().
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()
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_static_init(argc, argv, env);

exit(main(argc, argv, env));
}
Most cycles spent in malloc()
_init_tls()

- Find the ELF auxargs vector

- Use that to find the program headers
  Elf_Addr *sp;
  sp = (Elf_Addr *) environ;

- Use those to find the PT_TLS section (initial values)
  while (*sp++ != 0);

- Call __libc_allocate_tls() (as _rtld_allocate_tls())
  aux = (Elf_Auxinfo *

- 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));
}

Calls constructors and registers destructors. Four types supported:
• .pre_init_array section
• _init() function
• .ctors section (via _init())
• .init_array section
main()
vfprintf()
__get_locale()
vfprintf()
__vfprintf()
Look up decimal point string.

```
Look up decimal point string.
```

```
__vfprintf()
```

```
("%s", hello)
```

```
("%d", 123)
```

```
(\n"
```
__sprint()
__flush()

The actual call to write()
Hello World! 123
__start()
exit()

Call destructors registered with atexit()

Flush any unflushed FILEs

Call _exit()
Dynamic binary

_rtld_relocate_nonplt_self()
Rtld relocates itself

__start()
Load and relocate libc
__start()
printf()
_mips_rtld_bind()
printf()
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