A Simple C Runtime by Example

Yajin Zhou (http://yajin.org)

Zhejiang University

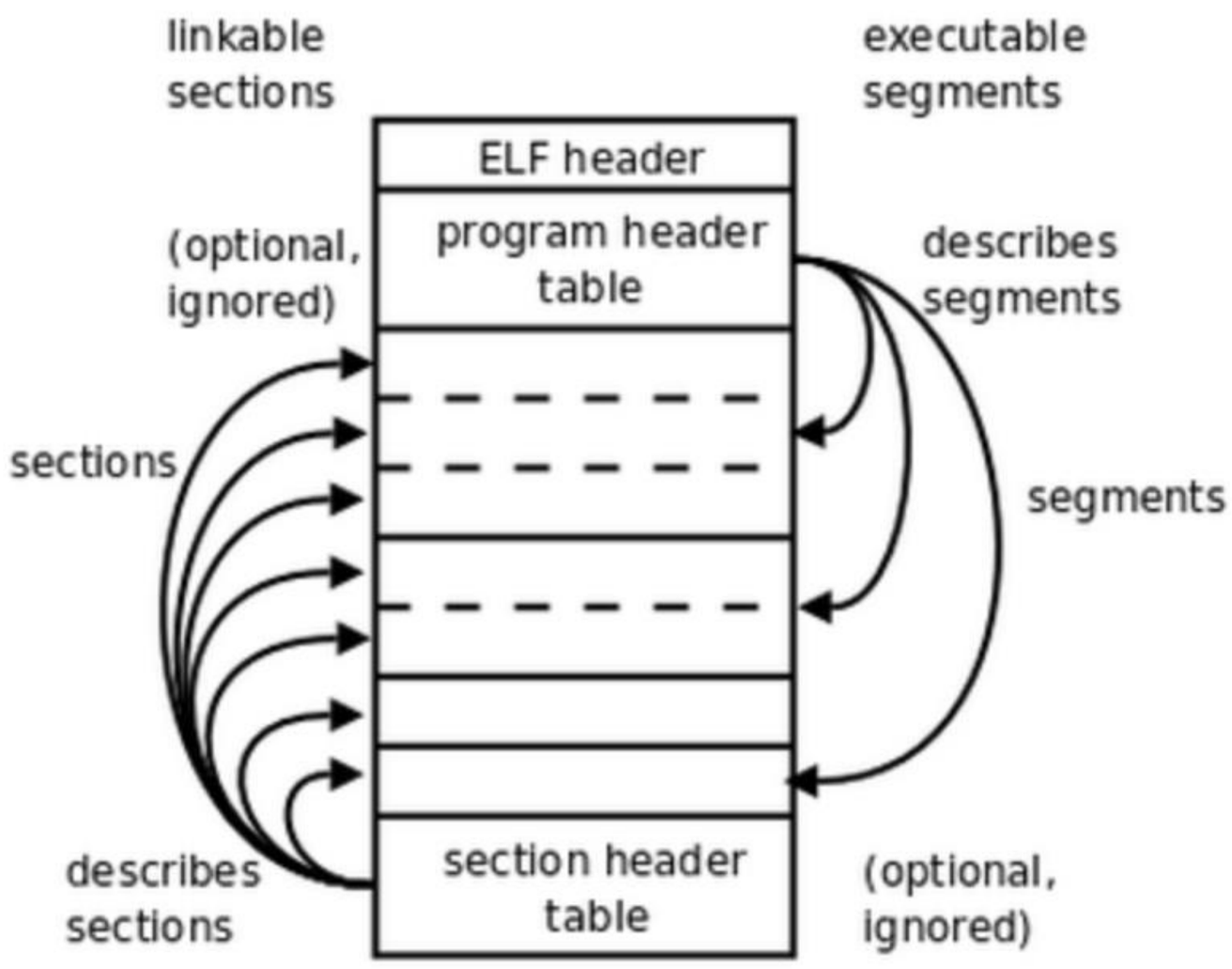
Review

- ELF
- QEMU
- A simple runtime for RISC-V

ELF

- Sections vs Segments
  - Linking view vs Execution view

Linking View	Execution View
ELF header	ELF header
Program header table *optional*	Program header table
Section 1	Segment 1
. . .	
Section *n*	Segment 2
. . .	. . .
. . .	
Section header table	Section header table *optional*



ELF

```
work@workvm:~/workspace/os2019/linux-0.11/boot$ riscv64-unknown-linux-gnu-readelf -S mai
There are 10 section headers, starting at offset 0x938:

Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS        0000000080000000  000000e8
       000000000000018a  0000000000000000  AX       0     0     4
  [ 2] .rodata           PROGBITS        0000000080000190  00000278
       0000000000000020  0000000000000001 AMS       0     0     8
  [ 3] .data             PROGBITS        00000000800001b0  00000298
       0000000000000020  0000000000000000  WA       0     0     8
  [ 4] .bss              NOBITS          00000000800001d0  000002c0
       0000000000001028  0000000000000000  WA       0     0     16
  [ 5] .riscv.attributes RISCV_ATTRIBUTE  0000000000000000  000002b8
       000000000000002b  0000000000000000           0     0     1
  [ 6] .comment          PROGBITS        0000000000000000  000002e3
       0000000000000011  0000000000000001  MS       0     0     1
  [ 7] .symtab           SYMTAB          0000000000000000  000002f8
       0000000000000450  0000000000000018           8    24     8
  [ 8] .strtab           STRTAB          0000000000000000  00000748
       000000000000019c  0000000000000000           0     0     1
  [ 9] .shstrtab         STRTAB          0000000000000000  000008e4
       000000000000004f  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

ELF

```
work@workvm:~/workspace/os2019/linux-0.11/boot$ riscv64-unknown-linux-gnu-readelf -l main

Elf file type is EXEC (Executable file)
Entry point 0x80000000
There are 3 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x00000000000000e8 0x0000000080000000 0x0000000080000000
                 0x00000000000001b0 0x00000000000001b0  R E    0x8
  LOAD           0x0000000000000298 0x00000000800001b0 0x00000000800001b0
                 0x0000000000000020 0x0000000000000020  RW     0x8
  LOAD           0x00000000000002c0 0x00000000800001d0 0x00000000800001d0
                 0x0000000000000000 0x0000000000001028  RW     0x10

 Section to Segment mapping:
  Segment Sections...
   00     .text .rodata
   01     .data
   02     .bss
```

Linker script

- The main purpose of the linker script is to describe how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file.

- Text **section** begins from 0x10000, .data **section** begins from 0x8000000

```
SECTIONS
{
  .  = 0x10000;
  .text : { *(.text) }
  .  = 0x8000000;
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

Linker script


```
OUTPUT_ARCH( "riscv" )

ENTRY( _start )

MEMORY
{
  ram   (wxa!ri) : ORIGIN = 0x80000000, LENGTH = 128M
}

PHDRS
{
  text PT_LOAD;
  data PT_LOAD;
  bss PT_LOAD;
}
```

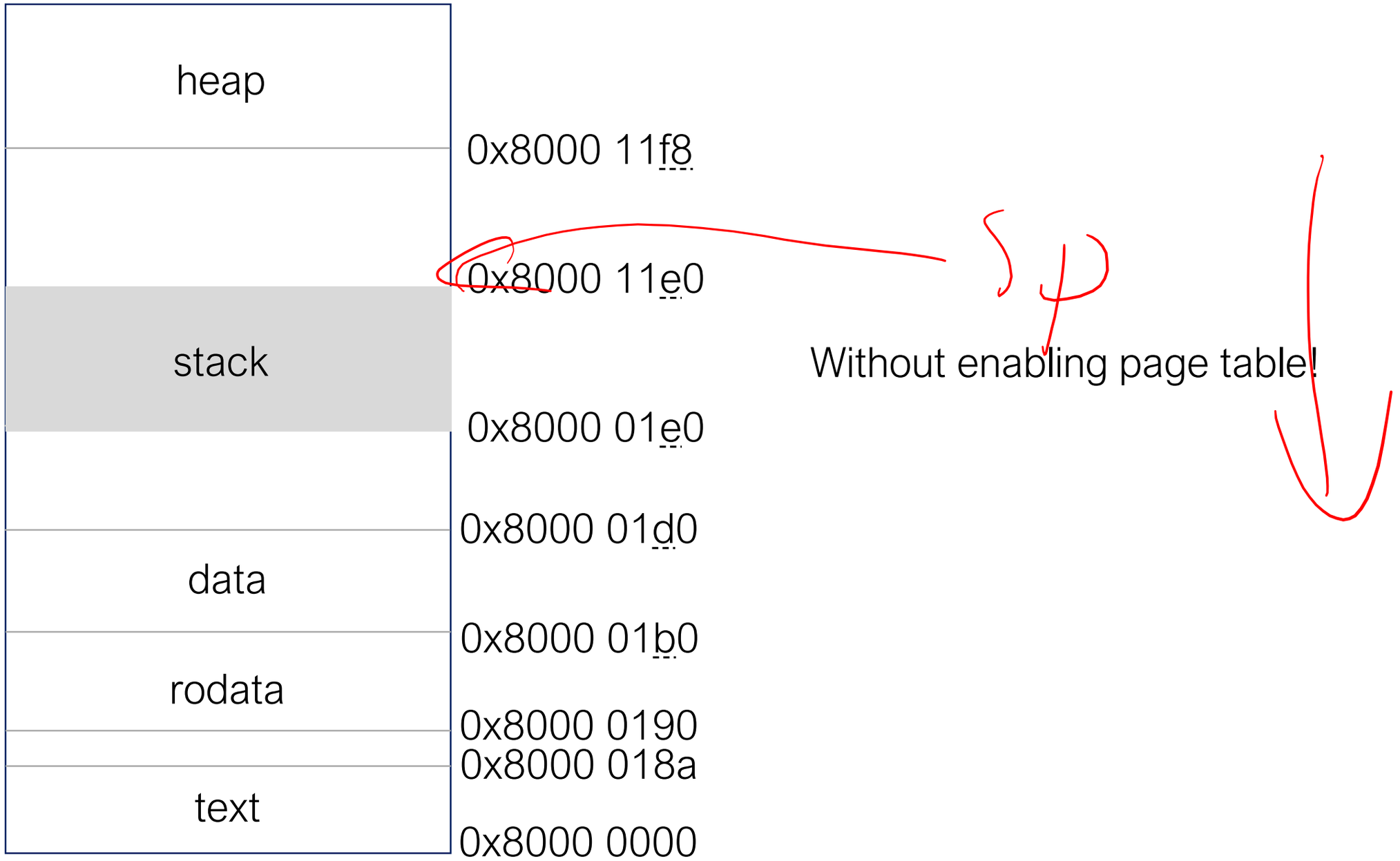
Linker script

```
SECTIONS
{
  .text : {
    PROVIDE(_text_start = .);
    *(.text.init) *(.text .text.*)
    PROVIDE(_text_end = .);
  } >ram AT>ram :text

  .rodata : {
    PROVIDE(_rodata_start = .);
    *(.rodata .rodata.*)
    PROVIDE(_rodata_end = .);
  } >ram AT>ram :text

  .data : {
    PROVIDE(_data_start = .);
    *(.sdata .sdata.*) *(.data .data.*)
    PROVIDE(_data_end = .);
  } >ram AT>ram :data
```

Final memory layout – execution view



C Runtime

- What we need?
  - Code is in memory
  - SP is ready

```
.include "macros.S"
.include "constants.S"

#
# start of trap handler
#

.section .text.init,"ax",@progbits
.globl _start

_start:
    # setup default trap vector
    la      t0, trap_vector
    csrw    mtvec, t0

    # set up stack pointer based on hartid
    csrr    t0, mhartid
    slli    t0, t0, STACK_SHIFT
    la      sp, stacks + STACK_SIZE
    add     sp, sp, t0

    # park all harts excpet hart 0
    csrr    a0, mhartid
    bnez    a0, park

    # jump to libfemto_start_main
    j       libfemto_start_main

    # sleeping harts mtvec calls trap_fn upon receiving IPI
park:
    wfi
    j       park
```

```c
__attribute__((noreturn)) void libfemto_start_main()
{
    // char *argv[] = {"femto", NULL};

    memset(&_bss_start, 0, &_bss_end - &_bss_start);

    register_console(&console_ns16550a);


    test = (uint32_t *)(void *)(SIFIVE_TEST_CTRL_ADDR);

    puts("\n\n\nHello World, Risc-v !");

    power_off();

    __builtin_unreachable();
}
```

```
0000000080000aa <libfemto_start_main>:
    800000aa:   00000517                auipc   a0,0x0
    800000ae:   12650513                addi    a0,a0,294 # 800001d0 <_data_end>
    800000b2:   00001617                auipc   a2,0x1
    800000b6:   14660613                addi    a2,a2,326 # 800011f8 <_bss_end>
    800000ba:   1141                    addi    sp,sp,-16
    800000bc:   8e09                    sub     a2,a2,a0
    800000be:   4581                    li      a1,0
    800000c0:   e406                    sd      ra,8(sp)
    800000c2:   0b6000ef                jal     ra,80000178 <memset>
    800000c6:   00000517                auipc   a0,0x0
    800000ca:   0f250513                addi    a0,a0,242 # 800001b8 <console_ns16550a>
    800000ce:   fb5ff0ef                jal     ra,80000082 <register_console>
    800000d2:   001007b7                lui     a5,0x100
    800000d6:   00000517                auipc   a0,0x0
    800000da:   0ba50513                addi    a0,a0,186 # 80000190 <_rodata_start>
    800000de:   00000717                auipc   a4,0x0
    800000e2:   0ef73923                sd      a5,242(a4) # 800001d0 <_data_end>
    800000e6:   072000ef                jal     ra,80000158 <puts>
    800000ea:   fafff0ef                jal     ra,80000098 <power_off>
```

NS16550a

- UART Device

```
static int ns16550a_getchar()
{
    if (uart[UART_LSR] & UART_LSR_DA) {
        return uart[UART_RBR];
    } else {
        return -1;
    }
}
```

```
static int ns16550a_putchar(int ch)
{
    while ((uart[UART_LSR] & UART_LSR_RI) == 0);
    return uart[UART_THR] = ch & 0xff;
}
```

```
enum {
    UART_RBR      = 0x00,  /* Receive Buffer Register */
    UART_THR      = 0x00,  /* Transmit Hold Register */
    UART_IER      = 0x01,  /* Interrupt Enable Register */
    UART_DLL      = 0x00,  /* Divisor LSB (LCR_DLAB) */
    UART_DLM      = 0x01,  /* Divisor MSB (LCR_DLAB) */
    UART_FCR      = 0x02,  /* FIFO Control Register */
    UART_LCR      = 0x03,  /* Line Control Register */
    UART_MCR      = 0x04,  /* Modem Control Register */
    UART_LSR      = 0x05,  /* Line Status Register */
    UART_MSR      = 0x06,  /* Modem Status Register */
    UART_SCR      = 0x07,  /* Scratch Register */
```

```
    UART_LSR_DA   = 0x01,  /* Data Available */
    UART_LSR_OE   = 0x02,  /* Overrun Error */
    UART_LSR_PE   = 0x04,  /* Parity Error */
    UART_LSR_FE   = 0x08,  /* Framing Error */
    UART_LSR_BI   = 0x10,  /* Break indicator */
    UART_LSR_RE   = 0x20,  /* THR is empty */
    UART_LSR_RI   = 0x40,  /* THR is empty and line is idle */
    UART_LSR_EF   = 0x80,  /* Erroneous data in FIFO */
};
```

QEMU Virt

```
work@workvm:~/risc-v/qemu$ ./riscv64-softmmu/qemu-system-riscv64  -machine help
Supported machines are:
none                 empty machine
sifive_e             RISC-V Board compatible with SiFive E SDK
sifive_u             RISC-V Board compatible with SiFive U SDK
spike                RISC-V Spike Board (default)
spike_v1.10          RISC-V Spike Board (Privileged ISA v1.10)
spike_v1.9.1         RISC-V Spike Board (Privileged ISA v1.9.1)
virt                 RISC-V VirtIO Board (Privileged ISA v1.10)
```

```
static const struct MemmapEntry {
    hwaddr base;
    hwaddr size;
} virt_memmap[] = {
    [VIRT_DEBUG] =        {        0x0,         0x100 },
    [VIRT_MROM] =         {     0x1000,        0x11000 },
    [VIRT_TEST] =         {   0x100000,         0x1000 },
    [VIRT_CLINT] =        {  0x2000000,        0x10000 },
    [VIRT_PLIC] =         {  0xc000000,     0x4000000 },
    [VIRT_UART0] =        { 0x10000000,         0x100 },
    [VIRT_VIRTIO] =       { 0x10001000,         0x1000 },
    [VIRT_DRAM] =         { 0x80000000,           0x0 },
    [VIRT_PCIE_MMIO] =    { 0x40000000,    0x40000000 },
    [VIRT_PCIE_PIO] =     { 0x03000000,    0x00010000 },
    [VIRT_PCIE_ECAM] =    { 0x30000000,    0x10000000 },
};
```