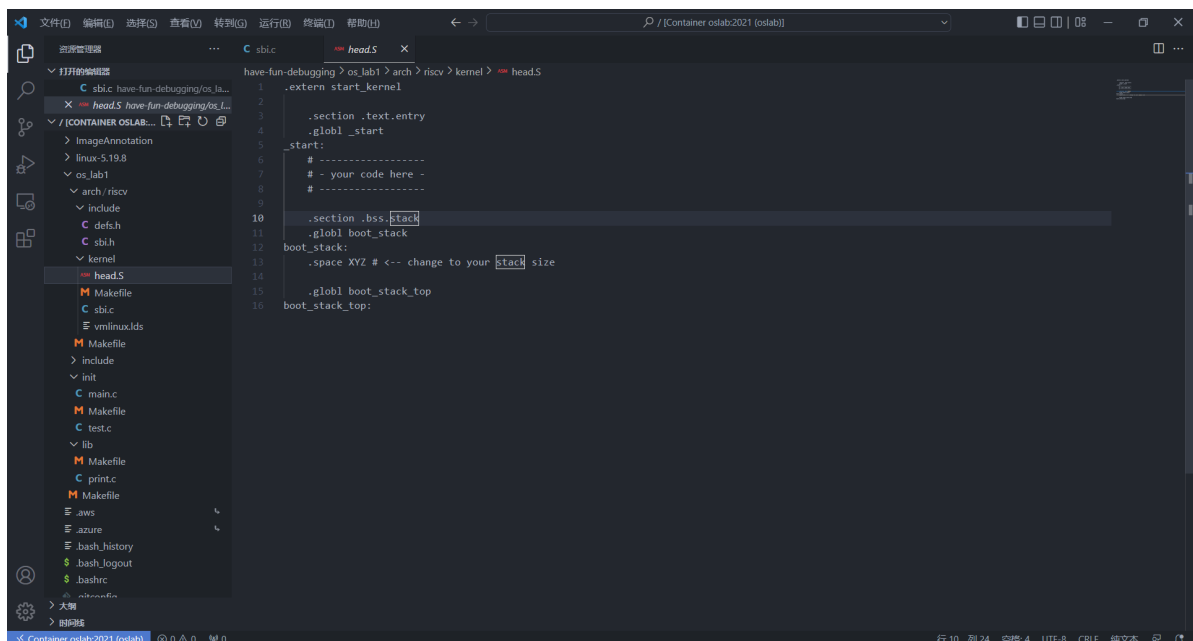
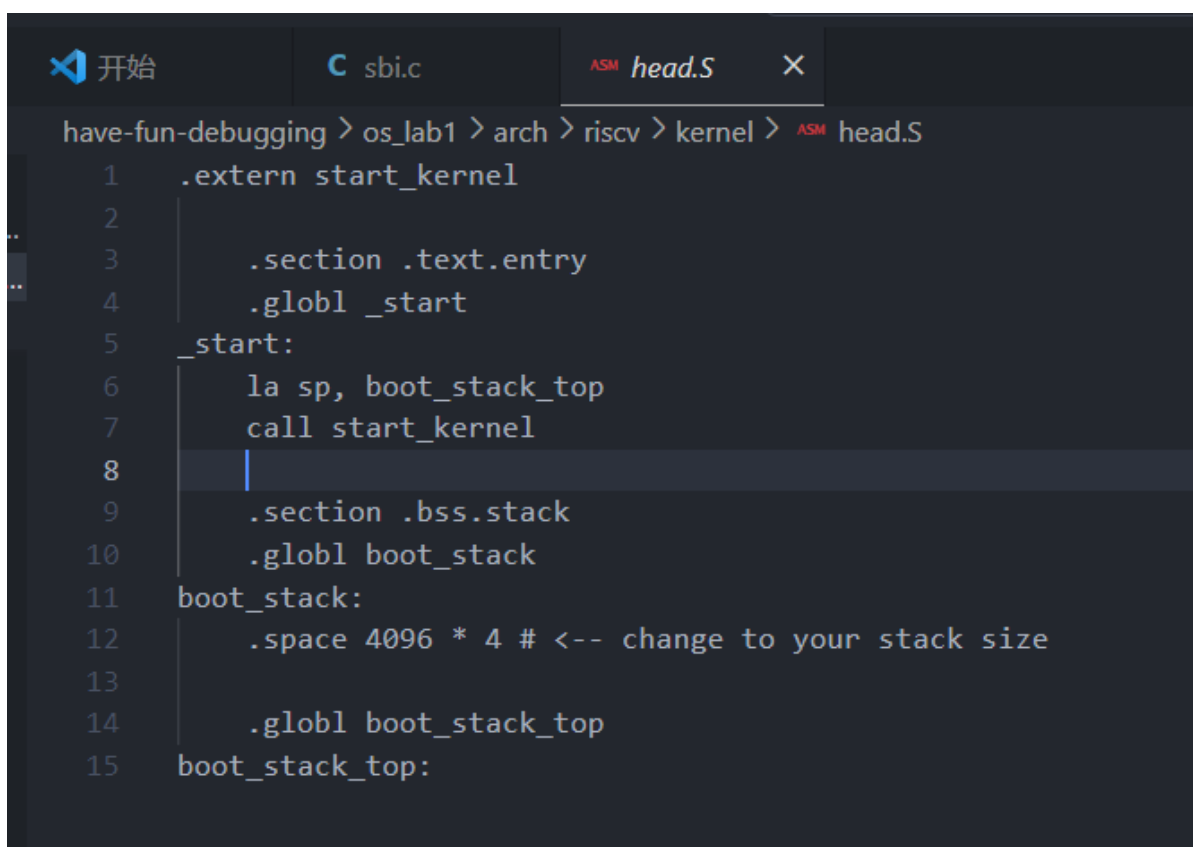


# OS-lab1

## 连接并开启docker



## 编写head.S



## 编写lib/Makefile

```
C sbi.c 3 M Makefile ../lib X M Makefile ../init ASM head.S
have-fun-debugging > os_lab1 > lib > M Makefile
1 C_SRC      = $(sort $(wildcard *.c))
2 OBJ        = $(patsubst %.c,%.o,$(C_SRC))
3
4 all:$(OBJ)
5
6 %.o:%.c
7 |   ${GCC}  ${CFLAG} -c $<
8
9 clean:
10 |   $(shell rm *.o 2>/dev/null)
```

## 补充sbi.c

```
开始 C sbi.c X M Makefile
have-fun-debugging > os_lab1 > arch > riscv > kernel > C sbi.c > sbi_ecall(int, int, uint64, uint64, uint64, uint64, uint64, uint64)
4 struct sbiret sbi_ecall(int ext, int fid, uint64 arg0,
5 |   uint64 arg1, uint64 arg2,
6 |   uint64 arg3, uint64 arg4,
7 |   uint64 arg5)
8 {
9     struct sbiret ret;
10    __asm__ volatile(
11        "mv a7, %[ext]\n"
12        "mv a6, %[fid]\n"
13        "mv a0, %[arg0]\n"
14        "mv a1, %[arg1]\n"
15        "mv a2, %[arg2]\n"
16        "mv a3, %[arg3]\n"
17        "mv a4, %[arg4]\n"
18        "mv a5, %[arg5]\n"
19        "ecall\n"
20        "mv %[error], a0\n"
21        "mv %[value], a1\n"
22        : [error] "=r"(ret.error), [value] "=r"(ret.value)
23        : [ext] "r"(ext), [fid] "r"(fid), [arg0] "r"(arg0), [arg1] "r"(arg1), [arg2] "r"(arg2),
24        [arg3] "r"(arg3), [arg4] "r"(arg4), [arg5] "r"(arg5)
25        : "memory", "a0", "a1", "a2", "a3", "a4", "a5", "a6", "a7");
26
27    ret.error = arg0;
28    ret.value = arg1;
29    return ret;
30
31
```

## puts()和puti()

have-fun-debugging > os\_lab1 > lib > C print.c > ...

```
1  #include "print.h"
2  #include "sbi.h"
3
4  void puts(char *s)
5  {
6      char tmp;
7      for (int i = 0;; i++)
8      {
9          if (s[i] != '\0')
10             sbi_ecall(0x1, 0x0, s[i], 0, 0, 0, 0, 0);
11          else
12             break;
13      }
14  }
15
16  void puti(int x)
17  {
18      char a[10];
19      int count = 0;
20      while(x != 0){
21          a[count++] = x % 10 + '0';
22          x /= 10;
23      }
24      for(int i = count-1; i >=0; i--){
25          struct sbiret ret = sbi_ecall(0x1, 0x0, a[i], 0, 0, 0, 0, 0);
26      }
27  }
28  }
29
```

## 修改defs

---

```

have-fun-debugging > os_lab1 > arch > riscv > include > C defs.h > ...
4  #include "types.h"
5
6  #define csr_read(csr) \
7      ({ \
8          register uint64 __v; \
9          asm volatile("csrr %[__v] , " #csr "\n" \
10             : [__v] "=r"(__v) \
11             : \
12             : "memory"); \
13          __v; \
14      })
15
16 #define csr_write(csr, val) \
17     ({ \
18         uint64 __v = (uint64)(val); \
19         asm volatile("csw " #csr ", %0" \
20            : \
21            : "r"(__v) \
22            : "memory"); \
23     })
24
25 #endif
26

```

## 思考题

### 1

- 变量应该尽量存放在寄存器中，且避免频繁恢复和保存寄存器
- 储存返回值的寄存器、传递参数的寄存器都可以不保留原值。其他寄存器需要保留原值。
- 如果参数和局部变量太多而无法在寄存器中存下，函数的开头会在栈中为函数帧分配空间来存放。当一个函数的功能完成后将会释放栈帧并返回调用点。

Caller Saved Register字面意思是由调用方负责保存的寄存器，被调用方可以随意使用。

Callee Saved Register字面意思是由被调用方负责保存的寄存器，在调用结束时，被调用方需要恢复被调用前的寄存器的值。

### 2

```

Build finished OK
● root@f40b9b9cbdfb:/have-fun-debugging/os_lab1# ls
Makefile  System.map  arch  include  init  lib  vmlinux
● root@f40b9b9cbdfb:/have-fun-debugging/os_lab1# nm vmlinux
0000000080200000 A BASE_ADDR
0000000080202000 d _GLOBAL_OFFSET_TABLE_
0000000080207000 B _ebss
0000000080202000 D _edata
0000000080207000 B _kernel
000000008020100f R _erodata
00000000802002cc T _etext
0000000080203000 B _sbss
0000000080202000 D _sdata
0000000080200000 T _skernel
0000000080201000 R _srodata
0000000080200000 T _start
0000000080200000 T _stext
0000000080203000 B boot_stack
0000000080207000 B boot_stack_top
00000000802001d8 T puti
0000000080200150 T puts
000000008020000c T sbi_ecall
0000000080200100 T start_kernel
0000000080200140 T test
○ root@f40b9b9cbdfb:/have-fun-debugging/os_lab1#

```

### 3

修改了main.c使得我们可以读取sstatus

```

have-fun-debugging > os_lab1 > init > C main.c > start_kernel()
...
1  #include "print.h"
...
2  #include "sbi.h"
...
3  #include "defs.h"
...
4
...
5  extern void test();
6
7  int start_kernel() {
8      puti(2022);
9      puts(" Hello RISC-V\n");
10     csr_read(0x100);
11
12     test(); // DO NOT DELETE !!!
13
14     return 0;
15 }
16

```

重新编译并开启debug, 使用gdb查看, 最终的读取结果是:

```
15 #define csr_write(csr, val)
输出 调试控制台 问题 终端 端口 1

0x80200100 <start_kernel>      addi    sp,sp,-16
0x80200104 <start_kernel+4>     sd      ra,8(sp)
0x80200108 <start_kernel+8>     sd      s0,0(sp)
0x8020010c <start_kernel+12>    addi    s0,sp,16
B+ 0x80200110 <start_kernel+16> li      a0,2022
0x80200114 <start_kernel+20>    jal      ra,0x802001dc <puti>
0x80200118 <start_kernel+24>    auipc   a0,0x1
0x8020011c <start_kernel+28>    addi    a0,a0,-280
0x80200120 <start_kernel+32>    jal      ra,0x80200154 <puts>
B+ 0x80200124 <start_kernel+36> csrr     a5,sstatus
>0x80200128 <start_kernel+40>  jal      ra,0x80200144 <test>
0x8020012c <start_kernel+44>    li      a5,0
0x80200130 <start_kernel+48>    mv      a0,a5
0x80200134 <start_kernel+52>    ld      ra,8(sp)

remote Thread 1.1 In: start_kernel
Breakpoint 3, start_kernel () at main.c:10
(gdb) i r a5
a5                0x0          0
(gdb) i r sstatus
sstatus           0x8000000000006000  -9223372036854751232
(gdb) si
(gdb) i r a5
a5                0x8000000000006000  -9223372036854751232
(gdb) |
```

说明读取成功。

查询资料后可以得知:

SD 位是一个只读位, 被置1, 说明了 FS、VS 或 XS 字段存在一些需要将扩展用户上下文保存到内存的脏状态。

而FS域描述浮点数单元状态, 被置11, 说明浮点数单元设置为dirty状态。全局关闭中断模式。

## 4

修改main.c为:

```
have-fun-debugging > os_lab1 > init > C main.c > start_kernel()
1  #include "print.h"
2  #include "sbi.h"
3  #include "defs.h"
4
5  extern void test();
6
7  int start_kernel() {
8      puti(2022);
9      puts(" Hello RISC-V\n");
10     csr_write(0x140,1);
11
12     test(); // DO NOT DELETE !!!
13
14     return 0;
15 }
16
```

开启gdb进行调试，发现csr\_w成功修改sscratch的值

```
输出 调试控制台 问题 终端 端口 1

0x80200100 <start_kernel>      addi    sp,sp,-32
0x80200104 <start_kernel+4>    sd      ra,24(sp)
0x80200108 <start_kernel+8>    sd      s0,16(sp)
0x8020010c <start_kernel+12>   addi    s0,sp,32
B+ 0x80200110 <start_kernel+16> li      a0,2022
0x80200114 <start_kernel+20>  jal     ra,0x802001e8 <puti>
0x80200118 <start_kernel+24>  auipc   a0,0x1
0x8020011c <start_kernel+28>  addi    a0,a0,-280
0x80200120 <start_kernel+32>  jal     ra,0x80200160 <puts>
0x80200124 <start_kernel+36>  li      a5,1
0x80200128 <start_kernel+40>  sd      a5,-24(s0)
0x8020012c <start_kernel+44>  ld      a5,-24(s0)
B+ 0x80200130 <start_kernel+48> csrw    sscratch,a5
> 0x80200134 <start_kernel+52> jal     ra,0x80200150 <test>
0x80200138 <start_kernel+56>  li      a5,0
0x8020013c <start_kernel+60>  mv      a0,a5
0x80200140 <start_kernel+64>  ld      ra,24(sp)
0x80200144 <start_kernel+68>  ld      s0,16(sp)
0x80200148 <start_kernel+72>  addi    sp,sp,32
0x8020014c <start_kernel+76>  ret
0x80200150 <test>            addi    sp,sp,-16
0x80200154 <test+4>          sd      s0,8(sp)

remote Thread 1.1 In: start_kernel

Breakpoint 2, 0x0000000080200130 in start_kernel () at main.c:10
(gdb) i r a5
a5                0x1      1
(gdb) i r sscratch
sscratch          0x0      0
(gdb) si
(gdb) i r sscratch
sscratch          0x1      1
(gdb) █
```

## 5

下载并解压linux-6.0源码

```
root@f40b9b9cbdfb:/have-fun-debugging# ls
C                linux-5.19.8      linux-6.0-rc5.tar.gz      os_lab1      server_xlab  test.c
ImageAnnotation  linux-5.19.8.tar.xz  linux-6.0-rc5.tar.gz:Zone.Identifier  rootfs.img  test        test.txt
root@f40b9b9cbdfb:/have-fun-debugging# tar -xf linux-6.0-rc5.tar.gz
root@f40b9b9cbdfb:/have-fun-debugging# ls
C                linux-5.19.8.tar.xz  linux-6.0-rc5.tar.gz:Zone.Identifier  server_xlab  test.txt
ImageAnnotation  linux-6.0-rc5       os_lab1
linux-5.19.8     linux-6.0-rc5.tar.gz  rootfs.img        test.c
root@f40b9b9cbdfb:/have-fun-debugging# cd linux-6.0-rc5
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# ls
COPYING      kbuild      MAINTAINERS  arch  crypto  include  ipc  mm      scripts  tools
CREDITS      Kconfig     Makefile     block  drivers  init     kernel  net      security  usr
Documentation LICENSES     README      certs  fs       io_uring  lib    samples  sound    virt
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5#
```

选择arm64架构



```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# make ARCH=arm64 defconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/kconfig/conf.o
HOSTCC  scripts/kconfig/confdata.o
HOSTCC  scripts/kconfig/expr.o
LEX      scripts/kconfig/lexer.lex.c
YACC     scripts/kconfig/parser.tab.[ch]
HOSTCC  scripts/kconfig/lexer.lex.o
HOSTCC  scripts/kconfig/menu.o
HOSTCC  scripts/kconfig/parser.tab.o
HOSTCC  scripts/kconfig/preprocess.o
HOSTCC  scripts/kconfig/symbol.o
HOSTCC  scripts/kconfig/util.o
HOSTLD  scripts/kconfig/conf
*** Default configuration is based on 'defconfig'
#
# configuration written to .config
#
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# |

```

开始用arm64架构交叉编译

```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j$(nproc) arch/arm64/kernel/sys.i
arch/arm64/Makefile:36: Detected assembler with broken .inst; disassembly will be unreliable
SYNC    include/config/auto.conf.cmd
*
* Restart config...
*
*
* ARMv8.3 architectural features
*
Enable support for pointer authentication (ARM64_PTR_AUTH) [Y/n/?] y
Use pointer authentication for kernel (ARM64_PTR_AUTH_KERNEL) [Y/n/?] (NEW) |

```

编译完成后我们可以看到我们成功获得了sys.i

```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# ls
COPYING  Documentation  Kconfig  MAINTAINERS  README  block  crypto  fs  init  ipc  lib  net  scripts
sound  usr
CREDITS  Kbuild  LICENSES  Makefile  arch  certs  drivers  include  io_uring  kernel  mm  samples  security
tools  virt
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# cd arch/
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5/arch# cd arm64
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5/arch/arm64# ls
Kbuild  Kconfig  Kconfig.debug  Kconfig.platforms  Makefile  boot  configs  crypto  hyperv  include  kernel  kvm  lib  mm
net  tools  xen
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5/arch/arm64# cd kernel/
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5/arch/arm64/kernel# ls
Makefile  cacheinfo.c  crash_dump.c  entry-fpsimd.S  hw_breakpoint.c  jump_label.c  module.
c  pi  reloc_test_syms.S  sleep.S  sys32.c  vdso-wrap.S
acpi.c  cpu-reset.S  debug-monitors.c  entry-ftrace.S  hyp-stub.S  kaslr.c  mte.c
pointer_auth.c  relocate_kernel.S  smccc-call.S  sys_compat.c  vdso.c
acpi_numa.c  cpu_errata.c  efi-entry.S  entry.S  idle.c  kexec_image.c  paravirt
t.c  probes  return_address.c  smpc.c  syscall.c  vdso32
acpi_parking_protocol.c  cpu_ops.c  efi-header.S  fpsimd.c  idreg-override.c  kgdb.c  patchin
g.c  process.c  sdei.c  smp_spin_table.c  time.c  vdso32-wrap.S
alternative.c  cpufeature.c  efi-rt-wrapper.S  ftrace.c  image-vars.h  kuser32.S  pci.c
proton-pack.c  setup.c  stacktrace.c  topology.c  vmlinux.lds.S
armv8_deprecated.c  cpuidle.c  efi.c  head.S  image.h  machine_kexec.c  perf_ca
llchain.c  psci.c  signal.c  suspend.c  trace-events-emulation.h
asm-offsets.c  cpuinfo.c  elfcore.c  hibernate-asm.S  io.c  machine_kexec_file.c  perf_ev
ent.c  ptrace.c  signal32.c  sys.c  traps.c
asm-offsets.s  crash_core.c  entry-common.c  hibernate.c  irq.c  module-plts.c  perf_re
gs.c  reloc_test_core.c  sigreturn32.S  sys.i  vdso
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5/arch/arm64/kernel# |

```

## 6

### ARM32, x86(32 bit), x86\_64

我们使用find命令在文件夹里面寻找，可以发现ARM32, x86(32 bit), x86\_64的syscall\_table已经存在了。

```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# find . -name '*.tbl'
./arch/m68k/kernel/syscalls/syscall.tbl
./arch/x86/entry/syscalls/syscall_32.tbl
./arch/x86/entry/syscalls/syscall_64.tbl
./arch/s390/kernel/syscalls/syscall.tbl
./arch/powerpc/kernel/syscalls/syscall.tbl
./arch/parisc/kernel/syscalls/syscall.tbl
./arch/sparc/kernel/syscalls/syscall.tbl
./arch/microblaze/kernel/syscalls/syscall.tbl
./arch/sh/kernel/syscalls/syscall.tbl
./arch/xtensa/kernel/syscalls/syscall.tbl
./arch/ia64/kernel/syscalls/syscall.tbl
./arch/arm/tools/syscall.tbl
./arch/mips/kernel/syscalls/syscall_n32.tbl
./arch/mips/kernel/syscalls/syscall_o32.tbl
./arch/mips/kernel/syscalls/syscall_n64.tbl
./arch/alpha/kernel/syscalls/syscall.tbl
./scripts/atomic/atomics.tbl
./tools/perf/arch/x86/entry/syscalls/syscall_64.tbl
./tools/perf/arch/s390/entry/syscalls/syscall.tbl
./tools/perf/arch/powerpc/entry/syscalls/syscall.tbl
./tools/perf/arch/mips/entry/syscalls/syscall_n64.tbl
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5#

```

接下来我们需要的就是RISC-V的syscall\_table

## RISC-V(64 bit)

我们使用64位编译配置，使用交叉编译编译出arch/riscv/kernel/syscall\_table.i

```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/kconfig/conf.o
HOSTCC  scripts/kconfig/confdata.o
HOSTCC  scripts/kconfig/expr.o
LEX      scripts/kconfig/lexer.lex.c
YACC     scripts/kconfig/parser.tab.[ch]
HOSTCC  scripts/kconfig/lexer.lex.o
HOSTCC  scripts/kconfig/menu.o
HOSTCC  scripts/kconfig/parser.tab.o
HOSTCC  scripts/kconfig/preprocess.o
HOSTCC  scripts/kconfig/symbol.o
HOSTCC  scripts/kconfig/util.o
HOSTLD  scripts/kconfig/conf
*** Default configuration is based on 'defconfig'
#
# configuration written to .config
#
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5#

```

```

root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5# make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc) ./arch/riscv/kernel/syscall_table.i
CALL    scripts/atomic/check-atomics.sh
CALL    scripts/checksyscalls.sh
CPP     arch/riscv/kernel/syscall_table.i
root@f40b9b9cbdfb:/have-fun-debugging/linux-6.0-rc5#

```

可以看到syscall\_table就在该文件中。

```
C syscall_table.i 9+ X
have-fun-debugging > linux-6.0-rc5 > arch > riscv > kernel > C syscall_table.i > ...
55107
55108
55109
55110
55111 void * const sys_call_table[451] = {
55112     [0 ... 451 - 1] = sys_ni_syscall,
55113     # 1 "../arch/riscv/include/asm/unistd.h" 1
55114     # 24 "../arch/riscv/include/asm/unistd.h"
55115     # 1 "../arch/riscv/include/uapi/asm/unistd.h" 1
55116     # 26 "../arch/riscv/include/uapi/asm/unistd.h"
55117     # 1 "../include/uapi/asm-generic/unistd.h" 1
55118     # 34 "../include/uapi/asm-generic/unistd.h"
55119     [0] = (sys_io_setup),
55120
55121     [1] = (sys_io_destroy),
55122
55123     [2] = (sys_io_submit),
55124
55125     [3] = (sys_io_cancel),
55126
55127
55128     [4] = (sys_io_getevents),
55129
55130
55131
55132
55133     [5] = (sys_setxattr),
55134
55135     [6] = (sys_lsetxattr),
55136
55137     [7] = (sys_fsetxattr),
55138
55139     [8] = (sys_getxattr),
55140
55141     [9] = (sys_lgetxattr),
55142
55143     [10] = (sys_fgetxattr),
55144
```

## RISC-V(32 bit)

我们使用32位编译配置。

```
root@f40b9b9cbd9b:/have-fun-debugging/linux-6.0-rc5# make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- rv32_defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdata.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/menu.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
*** Default configuration is based on 'defconfig'
#
# configuration written to .config
#
Using .config as base
Merging ./arch/riscv/configs/32-bit.config
Value of CONFIG_PORTABLE is redefined by fragment ./arch/riscv/configs/32-bit.config:
Previous value: CONFIG_PORTABLE=y
New value: # CONFIG_PORTABLE is not set
```

使用交叉编译工具进行编译。

```
root@f40b9b9cbd9b:/have-fun-debugging/linux-6.0-rc5# make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc) ./arch/riscv/kernel/syscall_table.i
SYMC include/config/auto.conf.cmd
WRAP arch/riscv/include/generated/uapi/asm/errno.h
WRAP arch/riscv/include/generated/uapi/asm/fcntl.h
WRAP arch/riscv/include/generated/uapi/asm/ioctl.h
WRAP arch/riscv/include/generated/uapi/asm/ioctls.h
WRAP arch/riscv/include/generated/uapi/asm/ipcbuf.h
WRAP arch/riscv/include/generated/uapi/asm/mman.h
WRAP arch/riscv/include/generated/uapi/asm/msgbuf.h
WRAP arch/riscv/include/generated/uapi/asm/param.h
WRAP arch/riscv/include/generated/uapi/asm/poll.h
WRAP arch/riscv/include/generated/uapi/asm/posix_types.h
WRAP arch/riscv/include/generated/uapi/asm/resource.h
WRAP arch/riscv/include/generated/uapi/asm/sembuf.h
WRAP arch/riscv/include/generated/uapi/asm/setup.h
```

## 成功得到32位的syscall\_table

```
C syscall_table.i X
have-fun-debugging > linux-6.0-rc5 > arch > riscv > kernel > C syscall_table.i > ...
54492
54493
54494
54495
54496 void * const sys_call_table[451] = {
54497     [0 ... 451 - 1] = sys_ni_syscall,
54498     # 1 "/arch/riscv/include/asm/unistd.h" 1
54499     # 24 "/arch/riscv/include/asm/unistd.h"
54500     # 1 "/arch/riscv/include/uapi/asm/unistd.h" 1
54501     # 26 "/arch/riscv/include/uapi/asm/unistd.h"
54502     # 1 "/include/uapi/asm-generic/unistd.h" 1
54503     # 34 "/include/uapi/asm-generic/unistd.h"
54504     [0] = (sys_io_setup),
54505
54506     [1] = (sys_io_destroy),
54507
54508     [2] = (sys_io_submit),
54509
54510     [3] = (sys_io_cancel),
54511
54512
54513
54514
54515
54516
54517
54518     [5] = (sys_setxattr),
54519
54520     [6] = (sys_lsetxattr),
54521
54522     [7] = (sys_fsetxattr),
54523
54524     [8] = (sys_getxattr),
54525
54526     [9] = (sys_lgetxattr),
54527
54528     [10] = (sys_fgetxattr),
```

## 7

ELF文件是一种用于二进制文件、可执行文件、目标代码、共享库和核心转储格式文件的文件格式。

我们以本次lab的head.S编译出的head.o文件为例

```
root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel# ls
Makefile head.S head.o sbi.c sbi.o vmlinux.lds
root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel# file head.o
head.o: ELF 64-bit LSB relocatable, UCB RISC-V, version 1 (SYSV), with debug_info, not stripped
root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel#
```

使用readelf命令：

```
root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel# readelf -a head.o
```

ELF Header:

```

Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00
Class:                               ELF64
Data:                                   2's complement, little endian
Version:                             1 (current)
OS/ABI:                               UNIX - System V
ABI Version:                          0
Type:                                 REL (Relocatable file)
Machine:                               RISC-V
Version:                               0x1
Entry point address:                  0x0
Start of program headers:              0 (bytes into file)
Start of section headers:              1632 (bytes into file)
Flags:                                 0x0
Size of this header:                   64 (bytes)
Size of program headers:               0 (bytes)
Number of program headers:             0
Size of section headers:               64 (bytes)
Number of section headers:             18
Section header string table index: 17
```

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	0000000000000000	00000040
	0000000000000000	0000000000000000	AX 0 0	4
[ 2]	.data	PROGBITS	0000000000000000	00000040
	0000000000000000	0000000000000000	WA 0 0	1
[ 3]	.bss	NOBITS	0000000000000000	00000040
	0000000000000000	0000000000000000	WA 0 0	1
[ 4]	.text.entry	PROGBITS	0000000000000000	00000040
	0000000000000010	0000000000000000	AX 0 0	1
[ 5]	.rela.text.entry	RELA	0000000000000000	000003d0
	0000000000000078	0000000000000018	I 15 4	8
[ 6]	.bss.stack	NOBITS	0000000000000000	00000050
	0000000000000400	0000000000000000	WA 0 0	1
[ 7]	.debug_line	PROGBITS	0000000000000000	00000050
	000000000000003f	0000000000000000	0 0	1
[ 8]	.rela.debug_line	RELA	0000000000000000	00000448
	0000000000000078	0000000000000018	I 15 7	8
[ 9]	.debug_info	PROGBITS	0000000000000000	0000008f
	000000000000002e	0000000000000000	0 0	1
[10]	.rela.debug_info	RELA	0000000000000000	000004c0
	00000000000000a8	0000000000000018	I 15 9	8
[11]	.debug_abbrev	PROGBITS	0000000000000000	000000bd
	0000000000000014	0000000000000000	0 0	1
[12]	.debug_aranges	PROGBITS	0000000000000000	000000e0

```

[13] .rela.debug_arang RELA      0000000000000000 00000568
      0000000000000060 0000000000000018 I      15      12      8
[14] .debug_str PROGBITS 0000000000000000 00000110
      0000000000000041 0000000000000001 MS      0      0      1
[15] .symtab SYMTAB      0000000000000000 00000158
      0000000000000240 0000000000000018      16      20      8
[16] .strtab STRTAB      0000000000000000 00000398
      0000000000000034 0000000000000000      0      0      1
[17] .shstrtab STRTAB      0000000000000000 000005c8
      0000000000000097 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)

There are no section groups in this file.

There are no program headers in this file.

There is no dynamic section in this file.

Relocation section '.rela.text.entry' at offset 0x3d0 contains 5 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000000000	001500000014	R_RISCV_GOT_HI20	0000000000004000	boot_stack_top + 0
000000000004	001100000018	R_RISCV_PCREL_LO1	0000000000000000	.L0 + 0
000000000004	000000000033	R_RISCV_RELAX		0
000000000008	001600000012	R_RISCV_CALL	0000000000000000	start_kernel + 0
000000000008	000000000033	R_RISCV_RELAX		0

Relocation section '.rela.debug\_line' at offset 0x448 contains 5 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000002a	000500000002	R_RISCV_64	0000000000000000	.L0 + 0
000000000036	000600000022	R_RISCV_ADD16	0000000000000008	.L0 + 0
000000000036	000500000026	R_RISCV_SUB16	0000000000000000	.L0 + 0
00000000003a	000800000022	R_RISCV_ADD16	0000000000000010	.L0 + 0
00000000003a	000600000026	R_RISCV_SUB16	0000000000000008	.L0 + 0

Relocation section '.rela.debug\_info' at offset 0x4c0 contains 7 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000000006	000f00000001	R_RISCV_32	0000000000000000	.debug_abbrev + 0
00000000000c	001000000001	R_RISCV_32	0000000000000000	.debug_line + 0
000000000010	000a00000002	R_RISCV_64	0000000000000000	.L0 + 0
000000000018	000b00000002	R_RISCV_64	0000000000000010	.L0 + 0
000000000020	000c00000001	R_RISCV_32	0000000000000000	.L0 + 0
000000000024	000d00000001	R_RISCV_32	0000000000000007	.L0 + 0
000000000028	000e00000001	R_RISCV_32	0000000000000035	.L0 + 0

Relocation section '.rela.debug\_aranges' at offset 0x568 contains 4 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000000006	000900000001	R_RISCV_32	0000000000000000	.debug_info + 0
000000000010	000a00000002	R_RISCV_64	0000000000000000	.L0 + 0
000000000018	000b00000024	R_RISCV_ADD64	0000000000000010	.L0 + 0

```
000000000018 000a00000028 R_RISCV_SUB64 00000000000000 .L0 + 0
```

The decoding of unwind sections for machine type RISC-V is not currently supported.

Symbol table '.symtab' contains 24 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000	0	SECTION	LOCAL	DEFAULT	1	
2:	0000000000000000	0	SECTION	LOCAL	DEFAULT	2	
3:	0000000000000000	0	SECTION	LOCAL	DEFAULT	3	
4:	0000000000000000	0	SECTION	LOCAL	DEFAULT	4	
5:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	4	.L0
6:	0000000000000008	0	NOTYPE	LOCAL	DEFAULT	4	.L0
7:	0000000000000000	0	SECTION	LOCAL	DEFAULT	6	
8:	0000000000000010	0	NOTYPE	LOCAL	DEFAULT	4	.L0
9:	0000000000000000	0	SECTION	LOCAL	DEFAULT	9	
10:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	4	.L0
11:	0000000000000010	0	NOTYPE	LOCAL	DEFAULT	4	.L0
12:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	14	.L0
13:	0000000000000007	0	NOTYPE	LOCAL	DEFAULT	14	.L0
14:	0000000000000035	0	NOTYPE	LOCAL	DEFAULT	14	.L0
15:	0000000000000000	0	SECTION	LOCAL	DEFAULT	11	
16:	0000000000000000	0	SECTION	LOCAL	DEFAULT	7	
17:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	4	.L0
18:	0000000000000000	0	SECTION	LOCAL	DEFAULT	12	
19:	0000000000000000	0	SECTION	LOCAL	DEFAULT	14	
20:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	4	_start
21:	0000000000004000	0	NOTYPE	GLOBAL	DEFAULT	6	boot_stack_top
22:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	UND	start_kernel
23:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	6	boot_stack

No version information found in this file.

root@f40b9b9cbd7b:/have-fun-debugging/os\_lab1/arch/riscv/kernel#

使用objdump命令:

```

root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel# objdump
Usage: objdump <option(s)> <file(s)>
Display information from object <file(s)>.
At least one of the following switches must be given:
-a, --archive-headers    Display archive header information
-f, --file-headers       Display the contents of the overall file header
-p, --private-headers    Display object format specific file header contents
-P, --private=OPT,OPT... Display object format specific contents
-h, --[section-]headers  Display the contents of the section headers
-x, --all-headers        Display the contents of all headers
-d, --disassemble        Display assembler contents of executable sections
-D, --disassemble-all   Display assembler contents of all sections
--disassemble=<sym>     Display assembler contents from <sym>
-S, --source             Intermix source code with disassembly
--source-comment[=<txt>] Prefix lines of source code with <txt>
-s, --full-contents      Display the full contents of all sections requested
-g, --debugging          Display debug information in object file
-e, --debugging-tags     Display debug information using ctags style
-G, --stabs              Display (in raw form) any STABS info in the file
-W[!LiaprmfFsoRtUuTgAckK] or
--dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
=frames-interp,=str,=loc,=Ranges,=pubtypes,
=gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
=addr,=cu_index,=links,=follow-links]
Display DWARF info in the file
--ctf=SECTION           Display CTF info from SECTION
-t, --syms              Display the contents of the symbol table(s)
-T, --dynamic-syms      Display the contents of the dynamic symbol table
-r, --reloc             Display the relocation entries in the file
-R, --dynamic-reloc     Display the dynamic relocation entries in the file
@<file>                Read options from <file>
-v, --version           Display this program's version number
-i, --info              List object formats and architectures supported
-H, --help              Display this information

```

由于可打印选项太多，我们以objdump -f为例

```

root@f40b9b9cbdfb:/have-fun-debugging/os_lab1/arch/riscv/kernel# objdump -f head.o

head.o:      file format elf64-little
architecture: UNKNOWN!, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000

```

为了演示一个可以直接run的elf文件，我们以我之前写的一个奇怪的c程序为例：

```

root@f40b9b9cbdfb:/have-fun-debugging/C/poem_translate# ls
poem_translate.c  resource  voice.vbs
poem_translate.exe  test     '$\345\244\247\350\213\261\344\275\234\344\270\232\347\232\204\350\257\227\346\255\214\347\277\273\350\257\221''.md'
root@f40b9b9cbdfb:/have-fun-debugging/C/poem_translate# file test
test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=0ef61fd2c52901350ce250691e5260fab87bf2, for GNU/Linux 3.2.0, not stripped
root@f40b9b9cbdfb:/have-fun-debugging/C/poem_translate# |

```

查询该程序的PID

```

root@f40b9b9cbdfb:/# ps -a
  PID TTY          TIME CMD
  6713 pts/1        00:00:00 test
  6917 pts/3        00:00:00 ps
root@f40b9b9cbdfb:/# |

```

使用cat命令成功获取该程序的memory layout



```

root@f40b9b9cbdfb: /# cat /proc/6713/maps
564f15e55000-564f15e56000 r--p 00000000 08:10 324133 /have-fun-debugging/C/poem_translate/test
564f15e56000-564f15e57000 r-xp 00001000 08:10 324133 /have-fun-debugging/C/poem_translate/test
564f15e57000-564f15e58000 r--p 00002000 08:10 324133 /have-fun-debugging/C/poem_translate/test
564f15e58000-564f15e59000 r--p 00002000 08:10 324133 /have-fun-debugging/C/poem_translate/test
564f15e59000-564f15e5a000 rw-p 00003000 08:10 324133 /have-fun-debugging/C/poem_translate/test
564f1627f000-564f162a0000 rw-p 00000000 00:00 0 [heap]
7fd553722000-7fd553747000 r--p 00000000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd553747000-7fd5538bf000 r-xp 00025000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd5538bf000-7fd553909000 r--p 0019d000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd553909000-7fd55390a000 ---p 001e7000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd55390a000-7fd55390d000 r--p 001e7000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd55390d000-7fd553910000 rw-p 001ea000 08:10 62105 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd553910000-7fd553916000 rw-p 00000000 00:00 0
7fd55391b000-7fd55391c000 r--p 00000000 08:10 61879 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd55391c000-7fd55393f000 r-xp 00001000 08:10 61879 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd55393f000-7fd553947000 r--p 00024000 08:10 61879 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd553948000-7fd553949000 r--p 0002c000 08:10 61879 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd553949000-7fd55394a000 rw-p 0002d000 08:10 61879 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd55394a000-7fd55394b000 rw-p 00000000 00:00 0
7ffffd856e000-7ffffd858f000 rw-p 00000000 00:00 0 [stack]
7ffffd85c3000-7ffffd85c7000 r--p 00000000 00:00 0 [vvar]
7ffffd85c7000-7ffffd85c8000 r-xp 00000000 00:00 0 [vdso]
root@f40b9b9cbdfb: /# |

```