Week 8

> Author：郑友捷

主要工作

- 实现arceos的用户态与内核态的初步分离
- 初步运行一个一等奖项目

用户态与内核态分离

- 应用的初始化
- 系统调用中Trap上下文的存储与切换

rcore中应用初始化的Trap上下文

1. 未实现不同任务空间时：

存储内容：应用用户栈栈顶，应用程序入口地址。

存储位置：应用对应的内核栈。

代码：

```
__restore(KERNEL_STACK.push_context(TrapContext::app_init_context(
    APP_BASE_ADDRESS,
    USER_STACK.get_sp(),
)) as *const _ as usize);
```

2. 已经实现任务空间切换后：

存储内容：应用程序入口，应用用户栈栈顶，内核地址空间的token，应用内核栈栈顶，trap_handler地址（即跳板页）。

存储位置：应用地址空间的次高虚拟页（最高页为跳板页）。

代码：

```
let trap_cx = task_inner.get_trap_cx();
*trap_cx = TrapContext::app_init_context(
    entry_point,
    ustack_top,
    KERNEL_SPACE.exclusive_access().token(),
    kstack_top,
    trap_handler as usize,
);
```

3. 发挥作用的方式：
    1. 在应用第一次获取CPU时，执行rcore的`trap_return`或者是`__restore`汇编函数。
    2. 将trap上下文的内容写入到对应的通用寄存器和CSR寄存器中。

3. 之后执行sret改变特权级，通过ra存储的地址进入到应用程序执行的位置。

实现系统调用的通用流程

1. 应用程序通过`ecall`或者`ebreak`陷入到trap，CPU自动修改特权级为S态。
2. 根据初始设置的`stvec`中的地址，跳转到预定义好的汇编函数，保存trap上下文到对应的位置（次高虚拟页或者内核栈），之后跳转到`trap_handler`或其他函数。
3. `trap_handler`预处理一部分内容，之后根据传入的参数进行`syscall`的分发，并执行对应的函数。
4. 在执行完`syscall`之后，`trap_handler`返回，此时会跳转到预定义好的汇编函数，从预定位置恢复trap上下文，并且执行sret回到原来的位置。根据原先的特权级恢复`sstatus`。

追溯arceos的trap处理函数建立过程

> 以riscv环境为例

1. 初始时进入`text`汇编代码段，第一个执行`text.boot`段代码
2. `axhal/src/platform/qumu_virt_riscv/boot.rs`中的`__start`函数位于`boot`顶部，执行一系列初始化操作，此时会调用`platform_init`函数。
3. `axhal/src/platform/qemu_virt_riscv/mod.rs`中执行`platform_init`函数，其中第二条指令的`set_tap_vector_base`设置了`stvec`的地址。
4. `axhal/src/arch/riscv/mod.rs`中执行`set_tap_vector_base`函数，设置stvec为汇编函数`trap_vector_base`的地址。
5. 汇编函数`trap_vector_base`定义在`axhal/src/arch/riscv/trap.S`，会判断当前trap来自于S还是U。并且进入对应的`trap_handler`，当前两个中断的处理函数相同，之后会进行区分。 执行完`trap_handler`之后会执行`restore_regs`汇编函数，回到原先的状态。

   相比于`rcore`，此时的`trap_vector_base`是之前`all_trap`和`restore`的结合体。

arceos实现系统调用的特点

1. 相比于rcore，arceos实现trap上下文的存储与读取是在一个函数`trap_vector_base`中完成，因此`trap_handler`中不用显式调用`return`。
2. 由于缺少了rcore中恢复trap上下文的`__restore`函数，故arceos进行应用初始化的trap上下文语句需要自行使用内嵌汇编书写。
3. 由于当前arceos属于是单进程多线程，所以内核栈可以直接用一段特定的物理内存代替，而不用切换任务地址空间。

为了实现系统调用而做的工作

1. 实现`syscall`分发函数，并且加入到`trap_handler`中，用于处理类型为`UserEnv`类型的异常。
2. 实现了`axhal/src/arch/riscv.rs`中的`trap_frame`的方法，该类型即是`trap context`。为其实现了类似于rcore的初始化方法：

```
impl TrapFrame {
    fn set_user_sp(&mut self, user_sp: usize) {
        self.regs.sp = user_sp;
    }
    /// 用于第一次进入应用程序时的初始化
    pub fn app_init_context(app_entry: usize, user_sp: usize) -> Self {
        let sstatus = sstatus::read();
        // 当前版本的riscv不支持使用set_spp函数，需要手动修改
        // 修改当前的sstatus为User，即是第8位置0
        let mut trap_frame = TrapFrame::default();
        trap_frame.set_user_sp(user_sp);
        trap_frame.sepc = app_entry;
        trap_frame.sstatus = unsafe { *(&sstatus as *const Sstatus as *const
usize) & !(1 << 8) };
        trap_frame
    }
}
```

3. 当每一个**进程**第一次获取CPU开始使用时，需要手写trap上下文恢复操作。**但因为arceos仅有一个进程，即是OS启动时对应的调度进程，因此直接写在了**`axruntime/src/lib.rs`**的rust_main入口中。**

```
#[cfg(feature = "user")]
pub fn init_process() -> ! {
    extern "Rust" {
        fn __user_start();      // 进程入口
    }
    use axhal::arch::TrapFrame;
    const STACK_SIZE: usize = 4096;
    static USERSTACK: [u8; STACK_SIZE] = [0; STACK_SIZE];
    static KERNELSTACK: [u8; STACK_SIZE] = [0; STACK_SIZE];
    let trap_frame = TrapFrame::app_init_context(
        __user_start as usize,
        USERSTACK.as_ptr() as usize + STACK_SIZE,
    );
    // copy from trap.S
    let frame_address = &trap_frame as *const TrapFrame;
    let kernel_sp = KERNELSTACK.as_ptr() as usize + STACK_SIZE;
    unsafe {
        core::arch::asm!(
            r"
            mv      sp, {frame_base}
            LDR     gp, sp, 2                   // load user gp and tp
            LDR     t0, sp, 3
            STR     tp, sp, 3                   // save supervisor tp
            mv      tp, t0                      // tp：线程指针
            csrw    sscratch, {kernel_sp}       // put supervisor sp to
scratch
            LDR     t0, sp, 31
            LDR     t1, sp, 32
            csrw    sepc, t0
            csrw    sstatus, t1
            POP_GENERAL_REGS
            LDR     sp, sp, 1
            sret
        ",
            frame_base = in(reg) frame_address,
```

```
            kernel_sp = in(reg) kernel_sp,
        );
    };
    core::panic!("already in user mode!")
}
```

4. 由于未将用户程序与内核的地址空间分开，因此需要关闭分页机制，即在`boot.rs`中关闭对`MMU`的初始化函数的调用。同时在`axruntime/src/lib.rs`中取消对`satp`寄存器的写入。

困难与问题

1. 一开始对特权级了解不够深入，忘记了U态下无法访问S态的内存，未意识到要关闭页表。
2. 在关闭页表时，一开始仅看到了`boot.rs`中的关闭页表操作，没有发现`axruntime/src/lib.rs`中也修改了satp寄存器，导致无法正确访问内存。
3. 对arceos的trap函数入口设置在一开始不太熟悉，花了近六个小时反复阅读源码。

一等奖项目运行：Maturin

遇到的问题与解决方法：

1. 原先默认编译target 为riscv64imac-unknown-none-elf，我的本机电脑并没有这个target。

   解决方法：`kernel/makefile`修改target为`riscv64gc-unknown-none-elf`。

2. rust工具链版本不同，导致许多特性在新的target上无法运行，如Result类型无法识别。

   出现报错形如：

```
error[E0412]: cannot find type `Ordering` in this scope
   --> /home/yoimiya/.cargo/registry/src/mirrors.ustc.edu.cn-61ef6e0cd06fb9b8/spin-0.7.1/src/rw_lock.rs:712:14
    |
712 |     success: Ordering,
    |              ^^^^^^^^ not found in this scope

error[E0412]: cannot find type `Ordering` in this scope
   --> /home/yoimiya/.cargo/registry/src/mirrors.ustc.edu.cn-61ef6e0cd06fb9b8/spin-0.7.1/src/rw_lock.rs:713:14
    |
713 |     failure: Ordering,
    |              ^^^^^^^^ not found in this scope

error[E0412]: cannot find type `Result` in this scope
   --> /home/yoimiya/.cargo/registry/src/mirrors.ustc.edu.cn-61ef6e0cd06fb9b8/spin-0.7.1/src/rw_lock.rs:715:6
    |
715 | ) -> Result<usize, usize> {
    |      ^^^^^^ not found in this scope
```

   解决方法：修改根目录下`rust-toolchain.toml`的`channel`为更新的`"nightly-2022-08-05"`，修改`rust-toolchain`值为`nightly-2022-08-05`。

3. 原有`virtio-drivers`依赖的`virtio_drivers::{VirtIOBlk, VirtIOHeader}`依赖在新的仓库代码中已经删除，导致无法引入依赖。

```
error[E0432]: unresolved imports `virtio_drivers::VirtIOBlk`, `virtio_drivers::VirtIOHeader`
 --> src/drivers/block/virtio_block.rs:4:22
  |
4 | use virtio_drivers::{VirtIOBlk, VirtIOHeader};
  |                      ^^^^^^^^^  ^^^^^^^^^^^^ no `VirtIOHeader` in the root
  |                      |
  |                      no `VirtIOBlk` in the root

For more information about this error, try `rustc --explain E0432`.
error: could not compile `maturin` due to previous error
make: *** [Makefile:60: kernel] Error 101
```

解决方法：修改引入依赖为当前仓库的某一个历史版本，修改`kernel/cargo.toml`中值为：

```
virtio-drivers = { git = "https://github.com/rcore-os/virtio-drivers", rev =
"499338115b7d462f051" }
```

参考了`https://github.com/rcore-os/rCore-Tutorial-v3/issues/86`解决方法。

4. 报错`[kernel] Panicked at src/drivers/memory/mod.rs:29 called Result::unwrap() on an Err value: CorruptedFileSystem`

   解决方法：参考readme.md，需要检查`\kernel\src\constants.rs`中的常量`pub const IS_PRELOADED_FS_IMG: bool`（在72行左右），需要修改这个值为 false。

5. 为在本地评测机运行，所以去除了编译选项--offline。

运行结果

```
testcase busybox printf "abcn" success
PID   USER     TIME  COMMAND
testcase busybox ps success
/
testcase busybox pwd success
             total        used        free      shared  buff/cache   available
Mem:             0           0           0           0           0           0
-/+ buffers/cache:           0           0
Swap:            0           0           0
testcase busybox free success
Sun Dec 31 00:00:00 1899  0.000000 seconds
testcase busybox hwclock success
sh: 10: unknown operand
testcase busybox kill 10 success
busybox                 libc.so                 run-dynamic.sh
busybox_cmd.txt         lmbench_all             run-static.sh
busybox_testcode.sh     lmbench_testcode.sh     runtest.exe
date.lua                lua                     sbin
dev                     lua_testcode.sh         sin30.lua
dlopen_dso.so           max_min.lua             sort.lua
entry-dynamic.exe       proc                    strings.lua
entry-static.exe        random.lua              test.sh
file_io.lua             remove.lua              tls_align_dso.so
lat_sig                 round_num.lua           tls_get_new-dtv_dso.so
lib                     run-all.sh              tls_init_dso.so
testcase busybox ls success
sh: 1: unknown operand
testcase busybox sleep 1 success
#### file opration test
```

```
        -1
Bandwidth measurements
Pipe bandwidth: 70.46 MB/sec
0.524288 186.27
0.524288 1865.52
0.524288 541.13
0.524288 15.82
context switch overhead

"size=32k ovr=188.36
2 551.37
4 458.42
8 491.89
16 606.12
24 410.28
32 398.02
64 560.16
96 497.26
[kernel] Panicked at src/file/device/test.rs:121
```

最后的panic代表程序测试结束，是正常现象。

下周工作

1. 完成陈老师上周安排的工作：`display.c`的运行。

   ```
   make: riscv64-linux-musl-gcc: No such file or directory
   make: *** [ulib/c_libax/build.mk:60: apps/c/helloworld/main.o] Error 127
   make: Leaving directory '/home/yoimiya/OSCOMP/arceos'
   ```

2. 查看Maturin的地址空间切换与页表机制，并做借鉴。
3. 实现多地址空间与多页表切换。为多进程做准备。