Traps: Syscalls, Exceptions & Interrupts

Computer Science 432 — Lecture 8 — Duane Bailey

February 28, 2022

Announcements

- Lab 3 (System Calls) due today.
  - Questions?
- Lab 4 (Page Tables) in lab, today.

Traps: System Calls, Exceptions, and Interrupts

- There are three main times when we need to suspend the current thread of execution:
  - When the user makes a *system call*. An example is the trace() call we implemented in lab 3.
  - When the machine generates an exception. An example is division by 0.
  - When an external agent is interested in registering an *interrupt*. Timers are an example.

- Every operating system (and operating system textbook) describes these events using different terms. In this course, all of these are simply called *traps*. RISC-V routes all traps the same way.

xv6 Trap Handling Approach

- All traps in xv6 are handled by the kernel (in supervisor or machine mode):
  - This greatly simplifies the process by having a single approach to interrupting processes.
  - If the user wishes to handle an exceptional condition, we might arrange for the user to register a *handler* (or, *vector*) within the kernel. Still, the kernel would be the first in line.
  - Most traps are handled in supervisor mode.
  - Timer interrupts are quickly handled in machine mode (see kernelvec.S). This gives timer handling precedence.

The Trap Handling CSR Registers

- The RISC-V machine provides a number of CSR registers which aid in the trap handling process:
  - `stvec` ("supervisor trap vector register"): the *physical address* of the (typically: assembly) code that will handle traps.
  - `sepc` ("supervisor exception program counter"): where the PC of the interrupted instruction is saved. On return from the trap, this register is copied to the PC.
  - `scause` ("supervisor cause register"): where RISC-V stores the cause of the exception (an integer).
  - `sscratch` ("supervisor scratch register"): a general purpose register used in context switching
  - `sstatus` ("supervisor status register"): a register whose bits control the handling of exceptions

- The `satp` ("supervisor address translation and protection register") is the *physical address* of the page table.

The Trap Instruction Behavior

- When a trap occurs, the following steps are taken (if `sstatus` SIE bit is set):
  1. Disable further interrupts: clear the SIE bit in the `sstatus` register.
  2. Copy the PC to the `sepc` register.
  3. Save the current mode (user or supervisor) in the SPP bit of the `sstatus` register.
  4. Set the cause of the trap in `scause`.
  5. Enter supervisor mode.
  6. Copy `stvec` to PC (called *vectoring*)

- Note that the machine *does not* switch page tables. That requires special choreography.

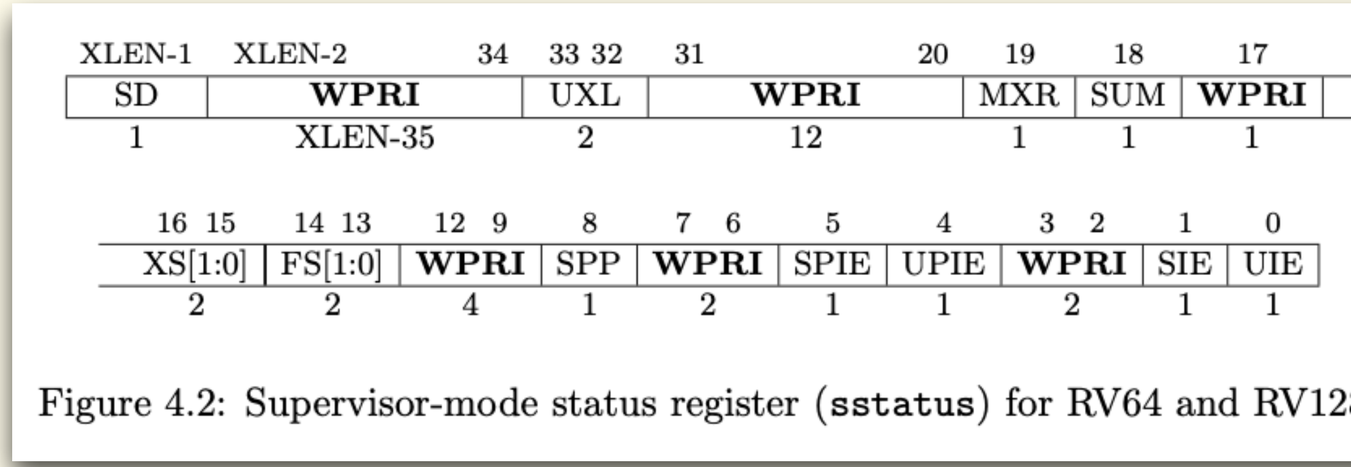


Figure 4.2: Supervisor-mode status register (**sstatus**) for RV64 and RV12

The Trap Handling Algorithm

- Trap handling entry and exit:
  - When a trap occurs, the machine essentially starts executing at routine pointed to by stvec.
  - When an exception happens in user mode, this routine is uservec (in trampoline.S)
  - `uservec`: saves the user's registers and switches to the supervisor page table. It then calls
    - `usertrap`: Set up kernel trap vector. Determine the cause of the trap and route it appropriately by calling…
      - syscall for system calls
      - devintr for device interrupts
      - kill the process for other exceptions.
        - [Trap is processed.]
    - `usertrapret`: Restore the user trap vector. Set up trapframe pointers. Then call…
  - `userret`: Change to user page table. Restore user registers. Return to resume execution.

uservec: trampoline.S usertrap, usertrapret: trap.c userret: trampoline.S

System Causes

- Because all exception handling is routed through one piece of code to dispatch/route appropriately, the *cause* is an important piece of information.
- Here, a "load page fault" due to null pointer de-reference in, OMG, the kernel:

```
$ sysinfotest
sysinfotest: start
scause 0x000000000000000d
sepc=0x0000000080000196 stval=0x0000000000000000
panic: kerneltrap
```

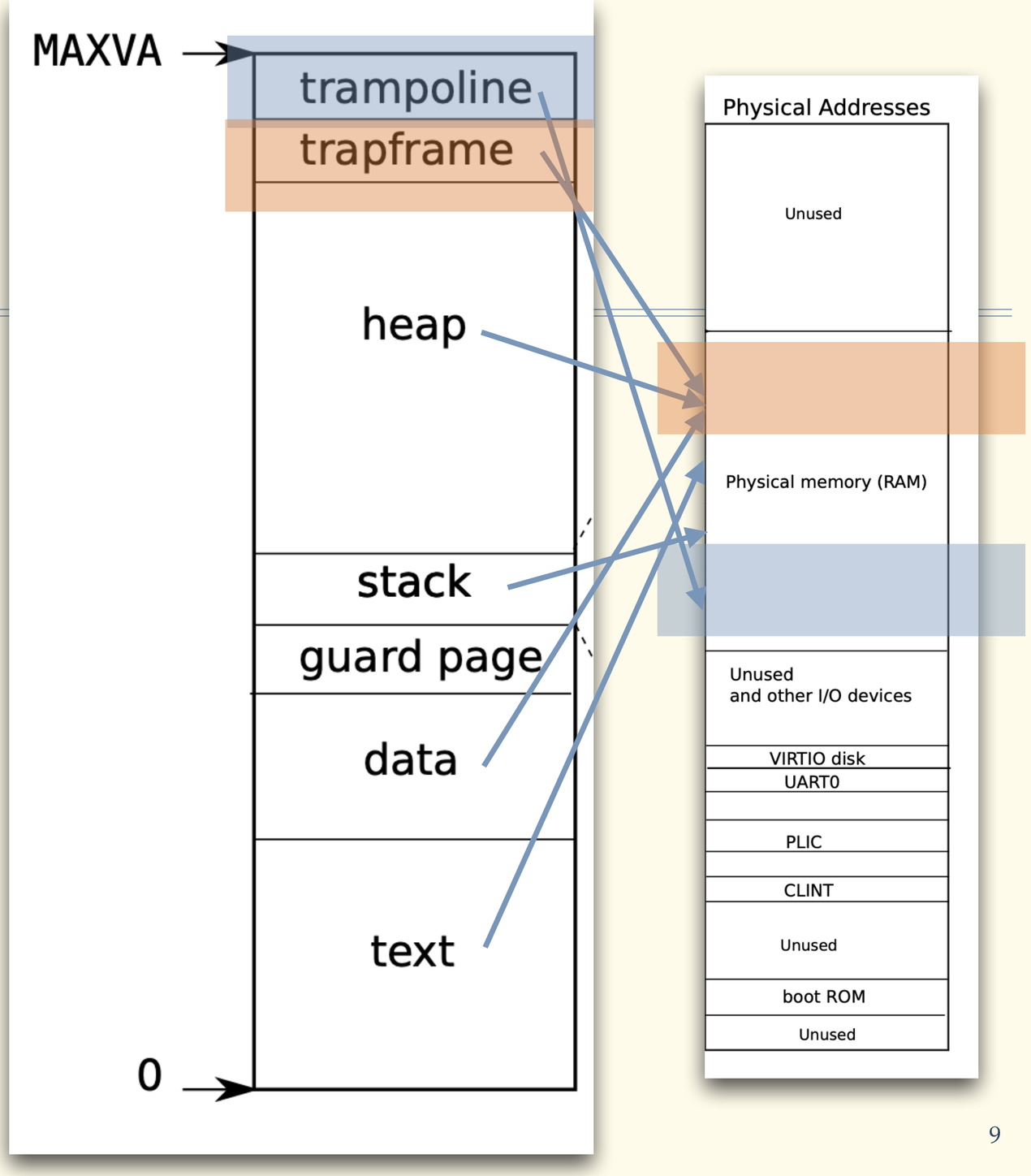
Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2–3	*Reserved*
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6–7	*Reserved*
1	8	User external interrupt
1	9	Supervisor external interrupt
1	≥10	*Reserved*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	*Reserved*
0	5	Load access fault
0	6	AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call
0	9–11	*Reserved*
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	≥16	*Reserved*

Table 4.2: Supervisor cause register (`scause`) values after trap.

Trampoline & Trapframe

- Important things to note:
  - The TRAMPOLINE page appears at the same virtual address in every process.
  - User mode processes cannot execute trampoline code; only after trap raises mode to supervisor
  - This allows trampoline code to swap page tables without problems.

  - Per-process trapframe supports swapping of contexts between user process and kernel.



Complexities in System Calls

- Because the handling of traps swaps page tables, system calls must work hard to ensure secure copying of arguments in and results out of code.
  - Supervisor has access to process page table pointer.
  - Manually *walking* this page table with a *virtual address* develops a *physical address*.
  - BUT REMEMBER, any physical address can be used as a virtual address in the kernel;
    the mapping is trivial.
  - The copy-in and copy-out code must be very careful to identify and illegal memory accesses it might make on behalf of user code.

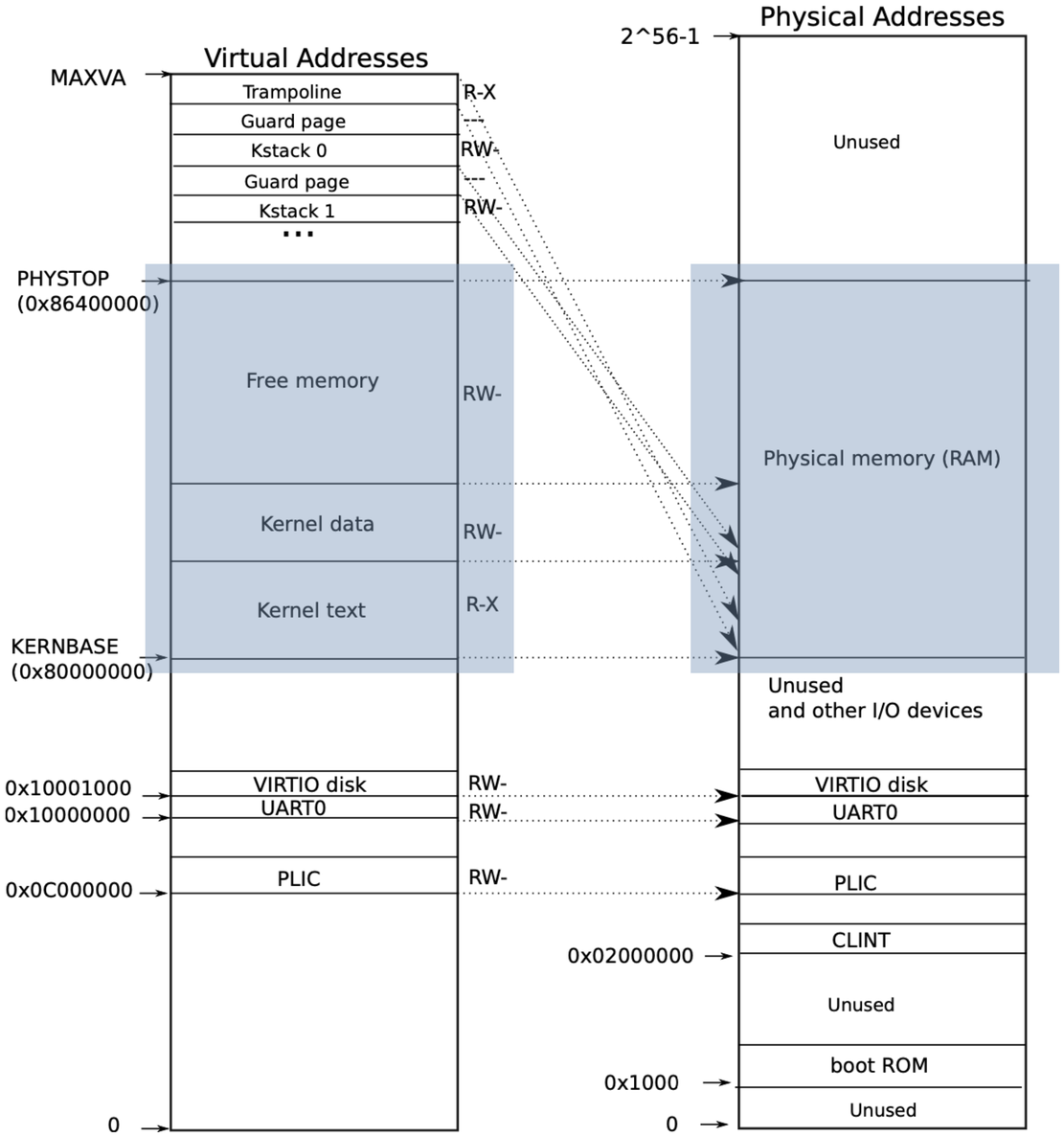


Figure 3.3: On the left, xv6's kernel address space. RWX refer to PTE read, write, and execute permissions. On the right, the RISC-V physical address space that xv6 expects to see.

The Trap Handling Algorithm Within Kernel

- The handling of traps the originate in the kernel is similar. BUT:
  - It uses a different trap vector (found in kernelvec.S).
  - The page table does not have to be changed.
  - Context is stored on the kernel stack, not in the trapframe.
  - There is a similar nesting-dolls set of routines for calling and returning from the trap handler.
  - Exceptions that happen in the kernel cause the kernel to *panic*.

kernelvec: kernelvec.S kerneltrap: trap.c

- One particular type of exception is important to think about: *page faults*.
  - Page faults are normal on real machines, so we must think carefully about how the O/S should handle them.
  - We'll discuss much of this on Wednesday.