Otto-Friedrich-Universität Bamberg
Fakultät WIAI

Lehrstuhl für Praktische Informatik, insbes.
Systemnahe Programmierung – Prof. Dr. Michael Engel
https://www.uni-bamberg.de/sysnap
michael.engel@uni-bamberg.de

Exercise 1

System Calls

Discussion on Monday, 30.5.2022

1.1 System Call Alternatives

Read the paper

FlexSC: Flexible System Call Scheduling with Exception-Less System Calls
https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Soares.pdf

by Livio Soares and Michael Stumm (published at OSDI 2010).

a. Take notes of things you do *not* understand in the paper to discuss next Monday
b. What is the proposed replacement for exceptions when implementing system calls?
c. What are the costs of a system call and which parts contribute to the costs?
d. Which hardware component did the authors expect to contribute to the performance of their approach, but it did not work out as expected?
e. Do you think this approach is also useful for a single-core processor?

1.2 System Call Implementation

System calls were discussed and demonstrated in this week's lecture. The skeleton code for a very simple RISC-V OS using system calls is available at https://github.com/michaelengel/OSE2022/tree/main/lecture3/

a. Fill in the missing parts and try to get system calls to work using the `syscall` function as shown in the lecture
b. Implement syscalls `printstring`, `putachar` and `getachar`
c. Add stubs for all system calls so they can be called as functions (`printstring`, `putachar` and `getachar` from user mode without having to pass the syscall number)
d. Read through the code in `ex.S` and `setup.c`. Take notes of things you do not unserstand – refer to the RISC-V specification (links in the lecture slides of lecture 3) for CSR information

1.3 System Calls in the xv6 OS

The source code for the RISC-V version of MIT's xv6 OS is available at

https://github.com/mit-pdos/xv6-riscv

Find out how xv6 implements system calls and try to answer the following questions:

a. Which functions implement exception handling in xv6?
b. In which processor mode are interrupts handled in xv6?
c. How does xv6 find out that an exception was caused by `ecall` (and not another cause such as an illegal instruction or a device interrupt)?
d. Where does the fixup to return to the correct address after an **ecall** instruction happen?
e. What does the Perl script `usys.pl` (https://github.com/mit-pdos/xv6-riscv/blob/riscv/user/usys.pl) do?