IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

课程实验1：机器启动

陈海波 / 夏虞斌

负责助教：沈斯杰 (ds_ssj@sjtu.edu.cn)

上海交通大学并行与分布式系统研究所

https://ipads.se.sjtu.edu.cn

版权声明

- 本内容版权归**上海交通大学并行与分布式系统研究所**所有
- 使用者可以将全部或部分本内容免费用于非商业用途
- 使用者在使用全部或部分本内容时请注明来源：
  - 内容来自：上海交通大学并行与分布式系统研究所+材料名字
- 对于不遵守此声明或者其他违法使用本内容者，将依法保留追究权
- 本内容的发布采用 Creative Commons Attribution 4.0 License
  - 完整文本：https://creativecommons.org/licenses/by/4.0/legalcode

实验准备

实验环境

- **Docker进行代码构建**
- **QEMU作为模拟器**
- **GDB为调试工具**
- **编辑器**
  - VIM + universal-ctags
  - VS code

- **我们提供了VirtualBox和Vmware的虚拟机**

其他推荐熟悉环境

- Makefile
- Cmake
- Tmux

实验发布方式

- **迭代式实验**
- **发布实验要求**
  - https://ipads.se.sjtu.edu.cn/courses/os/
- **GitLab**
  - Tutorial on How to Get ChCore from Gitlab
  - 请按要求建立账户以及个人项目！
  - 请熟悉Git的使用（如fork，commit，push，merge，checkout等操作）

实验的提交与评分

- **具体见各个实验要求**
- **只提交允许修改的文件和文档**
- **评分：代码80%+文档20%**
  - 文档以要求中的回答、设计思路、遇到的困难为主
  - ChCore实验的建议
  - 切勿在文档中灌水
- **正确性会提供部分测试集**

注意

- **按照要求修改指定文件或函数**
- **独立完成，切勿抄袭！**
  - 账号和个人项目请勿泄露
- **请按时提交**
  - 鼓励多次git commit & git push

实验一简介

实验一

- **发布时间: 2020-03-06**
- **截止时间: 2020-03-25 23:59 (GMT+8)**
- **负责助教: 沈斯杰 (ds_ssj@sjtu.edu.cn)**
- **实验目的**
  - 熟悉使用编写ChCore的环境
  - 学习启动的汇编与代码，并能够编写简单kernel态功能
  - 熟悉ARM架构并且用工具获取信息

三个部分

- Part A: Bootstrap
- Part B: Bootloader
- Part C: Kernel

Part A: Bootstrap

- 阅读ARM手册
  - Part A1 and A3: 熟悉Aarch64 ISA
  - Part D: 指令参考

Syntax

`BL` *label*

Where:

label

Is the program label to be unconditionally branched to. Its offset from the address of this instruction, in the range ±128MB. The branch can be forward or backward within 128MB.

Usage

Branch with Link branches to a PC-relative offset, setting the register X30 to PC+4. It provides a hint that this is a subroutine call.

Part A: Bootstrap

- **阅读ARM手册**
  - Part A1 and A2: 熟悉Aarch64 ISA
  - Part D: 指令参考
- **使用QMUE进行模拟，GDB进行调试**
  - QMUE可以作为GDB server启动，并在真正运行前暂停

QEMU+GDB

```
make qemu-nox-gdb
***
*** Now run 'make gdb'.
***
qemu-system-aarch64 -nographic -machine raspi3 -serial null -serial m
on:stdio -m size=1G -kernel ./build/kernel.img -gdb tcp::1234 -S
```

```
(gdb) where
#0  0x0000000000080000 in _start ()
Backtrace stopped: not enough registers or memory available to unwind
 further
(gdb) s
Single stepping until exit from function _start,
which has no line number information.

Thread 4 received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1.4]
0x0000000000080004 in _start ()
(gdb) where
#0  0x0000000000080004 in _start ()
Backtrace stopped: not enough registers or memory available to unwind
```

Part B: Bootloader

- **单核启动**
  - 如何限制单核启动？
- **熟练使用指针**
- **使用objdump查看kernel.img**
  - ELF段信息
  - readelf

Part C: Kernel

- **实现bootloader和kernel的printf**
  - 不是用户态的printf
  - 实现逻辑类似
  - 实现数字和地址打印功能（支持不同进制的输出）

Aarch64的函数调用

- **和x86-64不全相同**
  - $X29 (FP): 保存栈底
  - $X30 (LP): 保存返回地址(PC+4)
  - BL指令：类似于call，将返回地址保存到X30跳转
  - 参数保存：X1-X9
  - 返回值：X0

Aarch64的函数调用

- **调用惯例**
  - 使用gdb查看函数的头部和尾部操作
  - 寄存器传参保存？返回地址的保存？
  - x/30i test_backtrace

实现函数回溯

```
// Test the stack backtrace function (lab 1 only)
void
test_backtrace(long x)
{
  kinfo("entering test_backtrace %d\n", x);
  if (x > 0)
    test_backtrace(x-1);
  else
    mon_backtrace();
  kinfo("leaving test_backtrace %d\n", x);
}
```

```
Stack backtrace:
  LR ffffff00000d009c  FP ffffff000020f330  Args 0 0 ffffff000020f350 ffffff00000d009c 1
  LR ffffff00000d009c  FP ffffff000020f350  Args 1 3e ffffff000020f370 ffffff00000d009c 2
  LR ffffff00000d009c  FP ffffff000020f370  Args 2 3e ffffff000020f390 ffffff00000d009c 3
  ...
```