

课程实验3：用户进程

陈海波 / 夏虞斌

负责助教：赵子铭 (dumplings_ming@126.com)

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

实验获取

- 实验代码发布在公共远端仓库（upstream）下的lab3 分支
- 使用Git操作与本地修改合并，并与自己的远端仓库（origin）同步

```
$ git commit -am 'my solution to lab2'
$ git fetch upstream
$ git checkout -b lab3 upstream/lab3
Branch lab3 set up to track remote branch refs/remotes/upstream/lab3.
Switched to a new branch "lab3"
$ git merge lab2
Merge made by recursive.
$ git commit -am "merged previous lab solutions"
$ git push -u origin lab3
Branch 'lab3' set up to track remote branch 'lab3' from 'origin'.
```

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

文件结构

- **boot**: boot代码
- **kernel**: 操作系统内核代码
 - common: kernel内部库
- **user**: 用户程序代码
 - lab3: lab3的测试程序
 - lib: 用户库
- **lib**: boot、kernel、user共用库

```
├─ boot
├─ kernel
│   ├─ CMakeLists.txt
│   ├─ common
│   ├─ exception
│   ├─ mm
│   ├─ process
│   ├─ sched
│   ├─ syscall
│   ├─ main.c
│   ├─ head.S
│   ├─ monitor.c
│   └─ tools.S
├─ lib
└─ user
    ├─ CMakeLists.txt
    ├─ binary_include.S
    ├─ lab3
    └─ lib
```

新增命令 – 编译用户程序

- **make user**
 - 编译所有用户程序
 - 耗时较长，故独立于其他所有task
 - user目录发生改动时需主动重新编译
 - 第一次运行前
 - 每次修改user代码后

新增命令 – 运行用户程序

- **make run-x**
 - 运行用户程序
 - x: 用户程序名

运行user/lab3/hello.c

```
$ make run-hello

*** Now building application hello
./scripts/docker_build.sh hello
compiling kernel ...
# .....
[BOOT] Install boot page table
# .....
[INFO] [ChCore] interrupt init finished
[INFO] [ChCore] root thread init finished
hello, world
[INFO] sys_exit with value 0
[ChCore] Lab stalling ...
```


新增命令 – 调试用户程序

- **make run-x-gdb**
 - 开启gdb服务器
- **make gdb**
 - 连接gdb服务器

```
$ make run-hello-gdb
```

```
*** Now building application hello
./scripts/docker_build.sh hello
compiling kernel ...
```

```
# .....
```

```
*** Now starting qemu-gdb
```

```
qemu-system-aarch64 -machine raspi3 -serial null -serial mon:stdio -m size=1G -kernel
./build/kernel.img -gdb tcp::1234 -S
```

```
$ make gdb
```

```
gdb-multiarch -n -x .gdbinit
```

```
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
```

```
# .....
```

```
The target architecture is assumed to be aarch64
```

```
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
```

```
0x0000000000008000 in ?? ()
```

```
add symbol table from file "./build/kernel.img" at
```

```
.text_addr = 0xffffffff000000cc000
```

```
add symbol table from file "./user/build/ramdisk/hello.bin" at
```

```
.text_addr = 0x400110
```

编译问题 – Werror

- **ChCore默认编译时开启了Werror**
 - 禁止存在任何Warning
 - 减少出现bug的概率
- **然而，实现过程中**
 - 许多代码尚未实现
 - 提供的代码框架会产生大量Warning
 - 可暂时关闭Werror（CMakeLists.txt中）
- **最终评分前请恢复Werror标签！**

实验三简介

实验三

- **发布时间: 2020-04-03**
- **截止时间: 2020-04-29 23:59 (GMT+8)**
- **负责助教: 赵子铭 (dumplings_ming@126.com)**
- **实验目的**
 - 在ChCore中成功运行第一个用户进程
 - 熟悉AArch64架构下的异常处理流程
 - 为ChCore添加异常处理和系统调用支持

三个部分

- **Part A: User Process**
- **Part B: Exception Handling**
- **Part C: System Calls and Page Faults**

Part A – User Process

- **了解Object、Capability的概念**
 - Object: 内核资源 (file)
 - Capability: 访问内核资源的引用 (fd)
- **熟悉ChCore中线程和进程的概念**
- **启动ChCore中的第一个用户态进程、线程**

Part A – 内容

- **Exercise * 2, 以读代码为主**
 - 代码填空, 完成三个函数
 - 阅读代码, 描述创建第一个进程和第一个线程的流程 (推荐使用调用图+简单描述)

start

main (kernel/main.c)

- uart_init
- mm_init
- exception_init (not com
- process_create_root
 - process_create
 - thread_create_main
- eret_to_thread
 - switch_context

Part B – Exception Handling

- 复习AArch64中的异常处理流程
- 修改ChCore，使其支持对异常的处理
- 处理未定义指令异常
 - 当fetch到的指令未出现在AArch64的定义中时，打印错误信息（`kinfo(UNKNOWN)`）

Part B – 内容

- **Exercise * 1**

- 初始化Exception Vector Table, 将异常重定向到C代码中
- 配置VBAR_EL1寄存器, 使其指向Exception Vector Table
- 修改异常处理C代码, 处理未定义指令异常

Part C – System Calls & Page Faults

- 基于Part B提供的异常处理机制
- 在ChCore中处理两类特殊 “异常”
 - 系统调用
 - Page Fault

Part C – 内容

- **Exercise * 6, 任务拆分较细**
- **系统调用 (Exercise * 3)**
 - 实现系统调用机制 (用户库、Kernel内部寻址)
 - 完成5个系统调用 (3个实现已提供, 仅连接即可)
- **用户线程退出 (Exercise *2)**
 - 分析当前用户线程结束后行为
 - 利用系统调用处理用户线程退出
- **Page Fault处理 (Exercise * 1)**
 - 处理Page Fault



Enjoy your lab