实例分析-1：进程

第零部分：初识 xv6

在 xv6 的项目结构中，操作系统内核的代码在 `kernel` 文件夹下，运行在操作系统之上用户态程序在 `user` 文件夹下。本节的目的是带大家初步认识 xv6 操作系统，涉及的源码文件都在 `user` 文件夹下。

基础题

- 安装相应工具链，在 qemu 中运行 xv6 。并在操作系统中运行 `ls` 命令。
- 阅读 `ls.c` 。请回答：代码中的 `read()` 函数和 `printf()` 函数哪个是系统调用？它们的函数声明在哪里，函数定义在哪里？
- 阅读 `usys.S` ，查阅 RISC-V 相关知识，请问 `ecall` 指令的功能是什么？（如果找不到 `usys.S` ，请自我反思一下，不要阅读静态代码）

进阶题

- 阅读 xv6 项目相关的 `Makefile` 文件、`mkfs.c` 文件和链接器相关文件，分析 xv6 把内核和用户态程序编译链接的整个过程。并思考一个问题，xv6 运行的时候，`ls` 可以看到里面有一个 README 文件，我在 xv6 操作系统把它删除，为什么本地项目的 README 文件仍存在（反之如此）？

第一部分：系统启动与进程数据结构

虽然真实的计算机中只有少量的物理 CPU 可用，但是操作系统通过虚拟化 CPU 的功能，为用户提供了几乎无数个 CPU 可用的假象。操作系统为正在运行的程序所提供的抽象，就是进程。本节主要分析 xv6 内核的启动流程，理解 xv6 中的进程抽象，所涉及的源码均在 `kernel` 文件夹下。

基础题

- 使用 gdb 给 xv6 的内核打断点，请回答 xv6 内核运行的第一条指令是什么，它的物理地址是什么?
- `entry.S` 文件中的 `_entry` 做了什么，之后程序跳转到哪里执行?
- 阅读 `riscv.h` 并查阅相关资料，回答 C 代码在调用汇编代码时，汇编代码是如何获取到相应参数的？ `riscv.h` 中涉及的 C 代码调用的汇编指令的含义是什么?
- 在 `start.c` 的 `start()` 函数中， `mret` 指令的作用是什么？在调用 `mret` 指令的之前和之后，CPU 分别运行在 RISC-V 架构的什么工作模式下？ `start()` 函数调用完后，程序走到了哪里?
- 阅读 `proc.c` 和 `proc.h` ，理解进程相关的数据结构，理解其中重要字段的含义和作用，如： `pid` ， `state` ， `trapframe` ， `context` 。
- 在 `proc.c` 的 `procinit()` 函数做了什么？ `procinit()` 之后，每个进程结构体的内核栈布局是什么样?
- 第一个用户进程是在哪里初始化的?

进阶题

- Linux 中进程相关的数据结构在哪里？相比 xv6 的 PCB 增加了哪些结构？请选取其中 1 ~ 2 项结合代码详细分析。

第二部分：进程调度

在 xv6 操作系统进行完一系列初始化后，它将进入到进程调度的逻辑中，实现时分共享 CPU 的能力。

基础题

- 进程调度逻辑的主体 `scheduler()` 函数的主要逻辑是什么？访问进程时为什么需要加锁？
- `scheduler()` 函数中的 `swtch()` 函数做了什么？它将控制转移到了什么地方？这是如何实现的？
- `swtch()` 函数保存了哪些寄存器，为什么只需要保存这些寄存器？
- `proc.c` 中的 `sched()` 和 `yield()` 函数做了什么？

进阶题

- xv6 和 Linux 中调度器如何选择下一个要执行的进程？可选取一个 Linux 调度算法针对代码详细分析。

第三部分：trap

由于不同教材的翻译问题，*trap*、*exception*、*interrupt*、*device interrupt*、*system call* 等概念总是让学生混乱。请阅读 xv6 配套教材，理解在 xv6 的语境下，上述概念的含义。这有助于代码的理解。

基础题

- 在用户态发生 trap 时，CPU 会跳转到哪里执行？为什么跳到这里（从 RISC-V 体系结构的角度解释）？
- 在内核态发生 trap 时，CPU 会跳转到哪里执行？为什么跳到这里（从 RISC-V 体系结构的角度解释）？
- `trapframe` 是 trap 处理时的重要数据结构，它的作用是什么？
- 在用户态发生 trap 和内核态发生 trap 时，保存和恢复现场的方式有什么区别？
- 如何区分 trap 的类型？
- 作为进程数据结构中的重要字段，`trapframe` 和 `context` 中都定义了很多寄存器，但是两者有些区别，请从它们的功能角度分析原因。
- 内核态发生的 trap ，在处理完 trap 后如何返回的？返回后操作系统处于内核态还是用户态？
- 用户态发生的 trap ，在处理完 trap 后如何返回的？返回后操作系统处于内核态还是用户态？
- xv6 中实现了哪些系统调用，这些系统调用的定义在哪里？
- 在发起系统调用时，xv6 使用了哪条指令实现用户态到内核态的切换？xv6 中是如何在用户态和内核态传递系统调用的参数的？

进阶题

- 如果 xv6 在内核态处理 trap 的过程中，需要切换到用户态来执行一个处理函数，那么它需要保存现场（我们假设这个处理函数是安全的，并且它会调用另一个系统调用来恢复现场）。请问它需要保存那些寄存器？
- Linux 中内核态和用户态之间数据传递的有哪些方式？请结合具体代码分析。

第四部分：进程的运行

基础题

- `proc.c` 中 `uchar initcode[]` 的诡异的二进制数据的含义是什么？为什么 `userinit()` 函数要把它拷贝到 usr page 中？它和 `user/initCode.S` 有什么联系？
- 第一个进程启动后在用户态执行的程序是什么？这个程序执行了哪个系统调用？
- 在 `exec.c` 的 `exec()` 中，用到的 `struct elfhdr` 数据结构，其中 `magic` ，`phnum` ，`phoff` 等字段的作用是什么？以及 `struct proghdr` 的数据结构，其中 `vaddr` ，`memsz` ，`filesz` 等字段的作用是什么？
- 在 `exec()` 中，如何确定并设置待运行的程序的 PC 值和栈指针？
- `exec()` 执行完以后，返回的地址是什么？为什么？
- 在`user/init.c` 中调用了 `fork()` 函数创建子进程。请问在调用 `fork()` 系统调用后，是父进程先返回还是子进程先返回？
- 在调用 `fork()` 系统调用后，子进程是如何从 RUNNABLE 转换到 RUNNING 状态的？
- 对于父进程和子进程，`fork()` 返回的 pid 相同么？为什么？
- `wait` 系统调用的功能？

进阶题

- 请结合代码详细分析 Linux 中 elf 文件格式（利用 `readelf` 命令），以及链接和加载的机制。