OS Project Part 3: 线程、进程

提交须知：本Lab包括至少2个问题，只有回答完问题并且输出预期的结果才可以获得全部分数。

如果你时间不足，无法完成请联系TA@peterzheng，寻找折中方案。

期望结果

你完成本次作业之后，在项目根目录，可以通过`./scripts/docker_build.sh`构建系统镜像。在完成内核的构建之后，可以输入`make qemu-nox`启动qemu并且模拟执行我们构建好的内核。由于会随着user目录下的程序的变化而变化，因此可能略有区别。但是你的输出应该至少包括：

```
[Pass: pt_kern_vmmap, kernel/memory/pagetable.c:47]: "unmap"
// 此处省略其余输出
Hello world!
```

你的输出不应该包括

```
putc: Should never be here!
```

此外，为了保证用户态程序的正确性， 最后的putc函数可能会修改用户程序使得输出略有区别。

关于作业

本次作业理论上你可以修改所有`kernel/process`下的所有文件。由于程序结构发生变化，因此助教调整了整个项目的结构。将所有的头文件全部移动到了`include`目录下。因此`include/process.h`和`include/syscall.h`都可以修改。

本次作业预期时长为 3 周，发布时间 2021年5月29日 ，截止时间暂定 2021年6月24日 13:00，且不会延期。

预期的代码量为1200行左右（事实上你可以用很短的代码段就完成所有的任务，因此这个代码量仅供参考）

本次作业的检查方式主要为Code Review + 检查输出。

部分1：中断

在这里中断根据触发的时候系统处于什么状态做区分。在给出的示例代码中，内核态的中断处理已经给出。你需要完成用户态的中断处理。需要填写的文件至少包括了：`kernel/process/tramp.S`和`kernel/process/trap.c`。

考虑到你可能需要保存上下文，请小心设计如何保存上下文，这一部分可能和`thread_t`中的设计相关。

对于中断，为了简化操作，用户态的时钟中断需要进行一次调度，而内核态的时钟中断可以不用做任何操作。

部分2：线程和进程

在这个project之中，我们的最小运行和调度单位都是线程。为了给大家最大的自由，因此线程和进程中包含的对象在这里全部给大家留空。你可以选择完全自己设计（但是请注意最小的调度单元应该是线程）。

限制的进程和线程的关系：

1. 进程是若干个线程的组合。
2. 线程仅有自己的线程栈空间，和其他进程共享其他所有区域。

部分3：调度器

这一部分你应该完成一个简单的调度器，调度的规则你可以自己定。给出了一个enqueue的例子和dequeue的例子方便你理解如何使用内置的list对象。

一个调度的流程可以这么描述：

调度请求（SYSCALL发起或者时钟中断）== 保存上下文 == 将线程保存到队列尾（或者插入到你的某种数据结构）== 还原目标的上下文 == 返回用户态

调度器的初始化操作是sched_init，完成初始化之后开始调度。

注意：我们目前仍然在单核场景下，考虑如果要变换成多核需要修改什么。

部分4：Syscall与用户进程

用户进程已经写在User目录下，并且以某种方式保证可以访问到。用户进程包含了自己的头文件stdlib（简单的LibC），你需要完成Syscall的内核处理函数和用户态的libc。同时，你还需要完成如何装载一个ELF。

目前你需要支持的syscall只有sys_exit、sys_putc、sys_yield。

问题

1. 在给出的代码中，用户的程序在qemu中是如何被访问到的?
2. 用户态的上下文保存和内核态的上下文保存有什么区别。