实验六

user-level threads and alarm

目的：了解并熟悉在上下文切换和系统调用的时候状态的保护和恢复。

本实验要求实现用户态线程切换，并且实现“时钟”特性，使得可以对程序产生时钟中断。

**具体操作：**

RISC-V汇编温故而知新：本实验对RISC-V的一些机理需要有比较深入的了解，[user/call.c]在命令*make fs.img*的执行下，产生用户态程序调用，以及一个可读的汇编程序 [user/call.asm]，阅读以解决之后的汇编语言编写问题。

用户态线程切换：xv6含有两个相关文件uthread\_switch.S和uthread.c，以及makefile。Uthread.c包含大多数的用户线程报，以及三个简易线程测试。但是，线程包中缺失了一部分线程创立和切换的代码。因此，本实验需要解决线程创立，寄存器保护和恢复的问题。

时钟中断：设计时钟，以周期性的告知进程它使用CPU的时间。这或许对计算型进程限制其cpu占用率，或者是那先想要进行周期性操作的线程有所帮助。总之，本实验需要实现一个简单的用户态中断和中断处理程序，这和解决页中断的方法类似。

步骤：

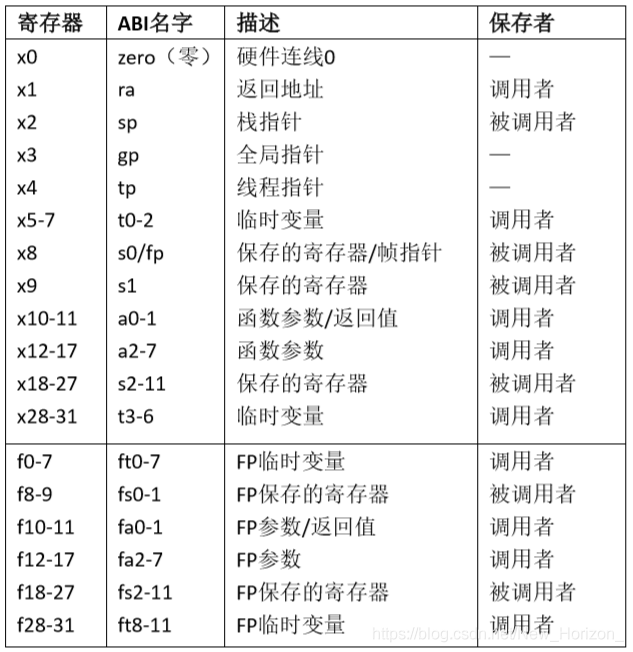
1. 修改thread\_create ：正确的创建一个初始化线程，让scheduler能够在第一次切换至次线程时能够正常执行，thread\_switch会返回线程栈中arg中func返回的值。解决方案有多种，修改线程结构体也可以，传给thread\_switch的参数也可以自定义，只要能够将线程t切换到next\_thread即可。

提醒：thread\_switch 只需要对使用的寄存器进行保护和恢复即可。可以在线程结构体中加入现场保护相关参数。可以在Uthread.asm中加入一些汇编代码帮助debug。测试的时候可以采用单步调试。

RISCV的寄存器如下，thread\_switch属于被调用者，因此需要保存栈指针sp，s0-s11，因为没有用到浮点数，则不需要保存。同时thread\_switch流程为保护上下文、切换上下文、到达下一个进程地址。注意到下一个进程地址是通过ret来跳转的，这表示ra必须为下一个跳转地址，因此我们还需要**额外**保存ra寄存器，并且初始化的时候必须为func函数地址。同时注意到thread结构体之中含有栈，这是给对应的线程分配的空间，因此还需要设置对应的栈指针。

总结：

* uthread创建：初始化线程上下文sp和ra分别为栈顶、func。
* thread\_switch调用参数：老线程和新线程参数上下文
* thread\_switch内容：保护老进程上下文，恢复新进程上下文。



<https://blog.csdn.net/new_horizon_/article/details/91814223>

1. 时钟中断：新增一个系统调用sigalarm(interval, handler)，当程序调用sigalarm(n, fn)的时候，每过n个CPU时段，系统将执行fn。Fn执行完后，系统应该要正常返回至原点。

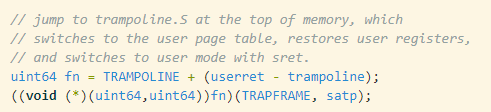
第一个问题：如何调用fn。当时钟中断发生，会进入trap，修改trap，使得其能够进入handler程序fn。这要求了解系统调用的过程（代码trampoline.S 与trap.c）。代码不多，技巧很强。

* 修改Makefile将alarmtest加入进去，并加入两个系统调用。
* Sigreturn先只要return0就好（会暂时导致出错系统宕机）。
* Sigalarm需要跟踪当前进程到底经历了多少次时段。每次时段硬件都会产生中断usertrap（which\_dev == 2），在这里给进程内的时段数加一。
* 这其中有一个隐藏的问题，periodic handler的地址为0，啥情况？

这个问题牵扯了一堆机制需要理解，详情见xv6的usertrap。阅读之后，最重要的就是这条，也就是usertrap如何返回源程序。这靠的就是tf->epc。



TRAMPOLINE中存放了userret的函数，找到，然后调用，就可以将sepc还原至pc，如果我们先一步把epc设置成handler，那么返回的就是handler。



第二个问题：如何从handler返回原程序。这要求保存源程序执行的寄存器，并且置零计时器。

这涉及系统调用的具体方法。在usertrap中将足够的状态保存在proc中，为了方便，直接把tf都保存起来，sigreturn粗暴的进行还原就可以了。

