南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

操作系统实验报告

Lab4 内核线程管理

2014074 费泽锟

年级：2020 级

专业：信息安全

指导教师：宫晓利

2022 年 12 月 10 日

摘要

本次实验主要探究 ucore kernel 中的内核线程管理策略，其实也就是如何记录每个线程（进程）的属于自己的信息，并且如何进行调度切换的过程。

在探究本次实验的过程之中，我们还需要对 lab1 之中的中断处理过程有一定的了解，因为这里涉及到了 context 何 trapframe 的转换的过程，这个过程不是很好理解，尤其是在针对 lab4 仅仅是内核线程的管理下进行理解就更困难了，会认为这两个地方是有些重复的，但是如果针对用户进程理解再返回来就容易一点。

最后就是在探究过程之中，我们还需要对 fork 和 exec 有一定的理解，才能充分理解 do_fork 函数进行的操作。

关键字：ucore、FIFO、proc struct、context、trapframe、内核线程管理

目录

一、 实验部分“吐槽”

对于 lab4 之中的内核线程管理，还是有不少的槽点需要吐槽的，首先的一点就是本实验之中需要我们一定要对中断处理流程和为什么需要中断有深刻的理解，其次就是我们还需要对虚拟内存管理的策略需要有所了解以及物理内存管理（不过这一部分涉及的不是很多，只需要知道其中如 mm 结构，cr3 寄存器以及 kmalloc 的 slab 策略即可），相当于这次的实验需要对前几个实验的知识点有不错的了解和一定的总结才行。

lab4 之中的还有一大特点就是，需要我们对汇编代码的了解有深入的了解，它其实隐去了其中不少的细节内容，给出如下的示例：

```
.globl switch_to
switch_to: # switch_to(from, to)
# save from's registers
movl 4(%esp), %eax # eax points to from
popl 0(%eax) # esp--> return address, so save return addr in FROM’s
context
movl %esp, 4(%eax)
……
movl %ebp, 28(%eax)
# restore to's registers
movl 4(%esp), %eax # not 8(%esp): popped return address already
# eax now points to to
movl 28(%eax), %ebp
……
movl 4(%eax), %esp
pushl 0(%eax) # push TO’s context’s eip, so return addr = TO’s eip
ret # after ret, eip= TO’s eip
```

图 1: context 切换

对于这一部分的代码，可能第一下很懵，为什么这里需要使用 esp+4 的索引，大家肯定都知道 C 语言函数参数入栈是从右向左入栈的，所以这里肯定 +4 是 from 的 context 结构而 +8 是 to 的 context 结构，但是这里其实隐去了一个步骤也就是在调用这个函数，是需要将返回地址入栈的，这也就是这里为什么需要 pop 0(%eax) 的原因。

然后在整个实验之中最令我疑惑的一点就是 context 和 trapframe 这两个地方了，显然这里 gitbook 实验文档之中写的是有点问题的，实验文档之中同时说了 context 的 eip 和 trapframe 的 eip 都是实际上 initproc 线程的开始地址，但实际上 trapframe 的 eip 才是正解。

这里这两个结构，看起来有点重复是因为这里是内核线程的原因，其实我们想想用户进程需要中断进入内核态后才能进行状态转换就知道这里为什么需要这两个结构了，中断帧是来进行中断处理所必需的结构。这里和助教进行讨论后，发现对于内核线程完全可以直接设置好 context 的 eip 来直接转换，但是为了体现这个流程，所以实验代码之中还添加了临时中断帧的操作。

这显然是很难理解的，感觉如果先用用户进程的实验来进行理解的话会更容易理解一点点。而且该实验的 challenge 已经实现了 slab，还要重构框架实现基于不同策略的 slab，还是内存管理的内容（有点儿忘了都），这个 challenge 就先鸽了。

最后就是那个 get pid 函数真的好难理解，那个函数其实是考虑了如果 pid 大的进程进入了 zombie 状态号后，pid 会形成一个区间中的空洞，所以维持一个区间就从空洞之中先去寻找，这个函数真的很难理解。

二、 实验总结

本次实验主要对 ucore 之中的内核线程管理策略进行了探究，虽然有了前几次 lab 的学习经验，本次实验的学习过程还是有点难度的，最关键的是学习资源竟然都没找到有关其余策略的 slab 算法实现。

在本次实验之中，重新对 fork 和 exec 进行了进一步的理解，同时还补充上了一块儿空白，就是对于刚开始用户进程之中的线程自己进行管理和操作系统进行管理的区别，当时没咋理解，现在快明白了。

参考文献

http://oslab.mobisys.cc/
https://kiprey.gitee.io/2020/08/uCore-4/
https://blog.csdn.net/dingdingdodo/article/details/100623864
https://blog.csdn.net/weixin_44765402/article/details/112414001