操作系统研讨课 实验报告

代瀚堃 2019K8009929051

一、实验中遇到的问题

1. 使用 gdb 调试干扰了镜像的加载

之前的实验中我一直使用 run_qemu.sh 这个脚本来 debug，但本次实验我在加载内核 ELF 时总是遇到 QEMU 资源不足的提示。实际上应该用 debug.sh，因为这个脚本刚运行起来后会像卡死一样，我就一直没用。如果直接运行，不连接 gdb 就不会有问题，这有些奇怪。经过仔细的思考，我猜测 run_qemu.sh 可能干扰了上面的过程，又想起助教之前说过 debug.sh 可以用来调试，于是对着它一通操作，进入了调试。一开始卡死的原因是 QEMU 在等待 gdb 的连接，当 gdb 用端口 1234 连接上以后，QEMU 进入 bbl 等待用户输入命令（看地址是 0x10000 处）。但这个时候还插不上断点，因为 QEMU 处于运行状态，需要 interrupt，才能插入，然后 continue，让刚刚被打断的 bbl 继续接收指令，输入 loadboot，就能进入 bootblock 了。

2. 地址变为 39/64 位，但加载地址还用之前实验的 lw 指令

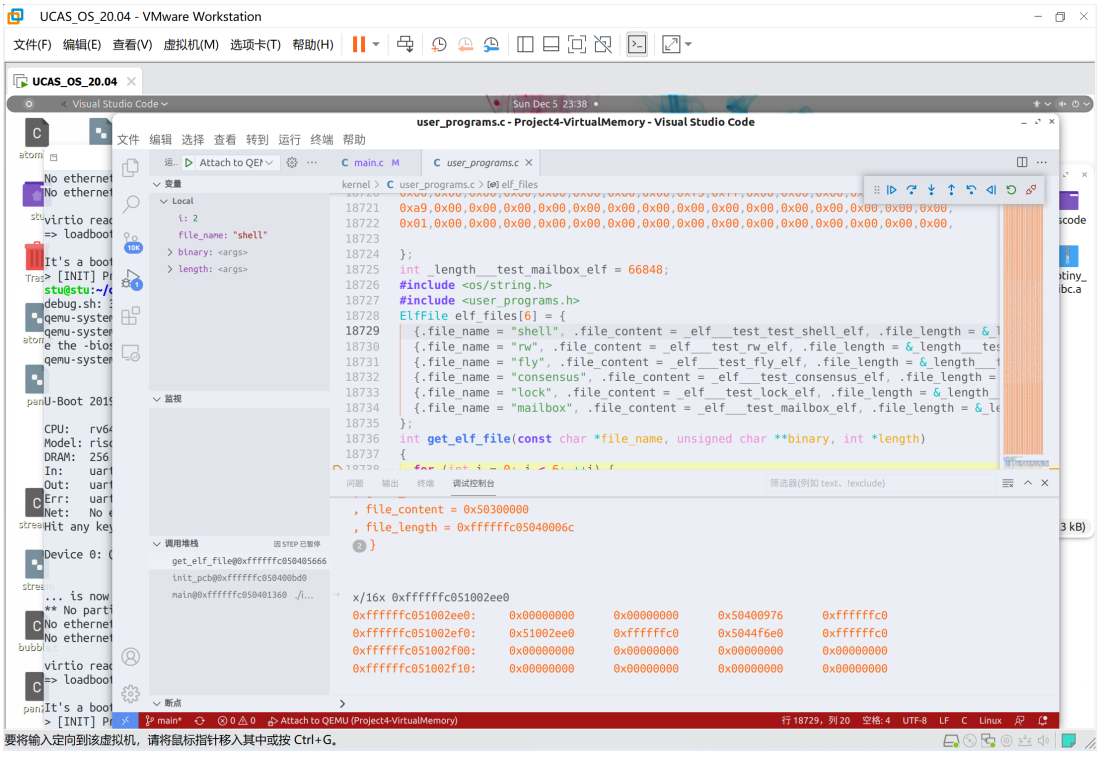
```
setup_C_core0:

    /* setup C environment */
    la      tp, pid0_pcb_core0
    ld      sp, pid0_stack_core0
    j       head_done

setup_C_core1:
```

这里的 ld 原来写的是 lw，我们的 CPU 是 64 位的，之前的实验可能还没体现出这一点，没有出问题，在 Project4 就体现出来了。

在加载 shell 的时候，运行完 get_elf_file 函数后，shell 这个字符串，连同后面的内容都没了，我们先进到这个函数的开头：



这里还没有被覆盖。。。然而查看了栈指针，发现 sp 似乎被截断了，只好从 bootblock 开始一步一步找，直到 start.S 发现了问题。

3. 页表项配置的各种错误

这里不应该加上 PAGE_SIZE：

```
//init stack
//Note that uintptr_t is uint64
uintptr_t stack = alloc_ph_page(1) + PAGE_SIZE;//kva
second_tab = (PTE *)alloc_pgtab();
third_tab = (PTE *)alloc_pgtab();
kmemset(stack, 0, NORMAL_PAGE_SIZE);
clear_pgdir(second_tab);
clear_pgdir(third_tab);

vpn2 = (USER_STACK_LOW >> (NORMAL_PAGE_SHIFT + PPN_BITS + PPN_BITS)) & VPN_MASK;
vpn1 = (USER_STACK_LOW >> (NORMAL_PAGE_SHIFT + PPN_BITS)) & VPN_MASK;
vpn0 = (USER_STACK_LOW >> (NORMAL_PAGE_SHIFT)) & VPN_MASK;

set_pfn(&first_tab[vpn2], (uintptr_t)(kva2pa(second_tab)) >> NORMAL_PAGE_SHIFT);
set_attribute(&first_tab[vpn2], _PAGE_PRESENT);

set_pfn(&second_tab[vpn1], (uintptr_t)(kva2pa(third_tab)) >> NORMAL_PAGE_SHIFT);
set_attribute(&second_tab[vpn1], _PAGE_PRESENT);

set_pfn(&third_tab[vpn0], (uintptr_t)(kva2pa(stack)) >> NORMAL_PAGE_SHIFT);
set_attribute(&third_tab[vpn0], _PAGE_PRESENT | _PAGE_READ | _PAGE_WRITE | _PAGE_EXEC | _PAGE_AC

return first_tab;
}
```

即便是栈向下增长，它也是一块空闲空间，栈顶由 init_pcb 设置，而非此处体现

运算符优先级错误。按位与的优先级比判断相等更低，原来的写法没有加括号：

```
if((pgdir[vpn2] & _PAGE_PRESENT) ==0){
    //allocate a second-level table
    set_pfn(&pgdir[vpn2], kva2pa(alloc_pgtab()) >> NORMAL_PAGE_SHIFT);
    set_attribute(&pgdir[vpn2], _PAGE_PRESENT);
    clear_pgdir(pa2kva(get_pa(pgdir[vpn2])));
}

PTE *pmd = (PTE *)pa2kva(get_pa(pgdir[vpn2]));
if((pmd[vpn1] & _PAGE_PRESENT) ==0){
    //allocate a third-level table
    set_pfn(&pmd[vpn1], kva2pa(alloc_pgtab()) >> NORMAL_PAGE_SHIFT);
    set_attribute(&pmd[vpn1], _PAGE_PRESENT);
    clear_pgdir(pa2kva(get_pa(pmd[vpn1])));
}

PTE *pte = (PTE *)pa2kva(get_pa(pmd[vpn1]));
if((pte[vpn0] & _PAGE_PRESENT) ==0){
    //allocate an available physical page
    uintptr_t new_page_kva = alloc_ph_page();
    kmemset((void *)new_page_kva,0,NORMAL_PAGE_SIZE);
    set_pfn(&pte[vpn0], kva2pa(new_page_kva) >> NORMAL_PAGE_SHIFT);
    set_attribute(&pte[vpn0], _PAGE_PRESENT | _PAGE_READ | _PAGE_WRITE | _PAGE_EXEC |
}

local_flush_tlb_all();
```

导致页表项的有效位判断错误，没能分配上物理页，一直进缺页处理程序

空闲物理页的下标出错：

```
    ph_page_array[next_replace].status=PAGE_BUSY;
    ph_page_array[next_replace].pte=pte;
    index=next_replace;
    if(next_replace==PH_PAGE-1){
        next_replace=30;
    }else{
        next_replace++;
    }

    phaddr=PHIDX_TO_PHADDR(index);

    return pa2kva(phaddr);
}

uintptr_t shm_page_get(int key)
```

导致页表项的有效位判断错误，没能分配上物理页，一直进缺页处理程序

应该是 PH_PAGE-1，错写成了 SWAP_NUM-1，导致超过了 ph_page_array 数组的长度，覆盖了其他数据结构，最后死机。

3. 拷贝 exec 的参数后发生缺页

此处调用 sys_exec 前要做一次指针的设置，而不是直接将 arg 传入：

```
    printf("\n");
}else if(strcmp(command,"exec")==0){
    char *argv[ARG_COUNT];
    for(int i=0;i<argc;i++){
        argv[i]=arg[i];
    }
    int result=sys_exec(arg[0], argc, argv, AUTO_CLEANUP_ON
    if(result>=0){
        printf("Exec %s task, with pid: %d\n", arg[0], resu
    }else{
        printf("Exec failed.\n");
    }
```

毕竟在 do_exit 内做指针的运算需要知道每次增加的值

4. 之前实验中使用了空指针，在 0 地址上取数

第一个暴雷的是 do_scheduler：

```
if(last_pcb == cpu_pcb_pointer[mycpu_id]){
    choose_index=1-mycpu_id;
}else{
    choose=(last_pcb->node).next;
    for(;;){
        if(choose==&(last_pcb->node)){
            break;
        }
        if(choose==NULL||choose==&ready_queue){
            choose=ready_queue.next;
        }else{
            choose_index=choose->index;
            if(pcb[choose_index].status==TASK_READY&&(pcb[choose_index].bo
                break;
            }else{
                choose=choose->next;
            }
        }
    }
    choose_index=choose->index;
}
current_running[mycpu_id]=&pcb[choose_index];
```

之前取 choose_index 紧跟着 choose，且在检查 choose 是否为空之前。在前面的实验中这不会有问题，因为即使遇到了 NULL，在上面取 index，也会返回 0，但 Project4 中，0 地址我们没有建立映射。

第二个是 do_mutex_lock_release：

```
void do_mutex_lock_release(mutex_lock_t *lock)
{
    //not necessary to rescheduler right now
    if((lock->block_queue).index>0){
        //NOTICE: status can't be UNLOCKED now
        //else two processes will assume they both hold the lock
        pcb[((lock->block_queue).next)->index].status=TASK_READY;
        pcb[((lock->block_queue).next)->index].hang_on=&ready_queue;
        move_node((lock->block_queue).next,&ready_queue,&(lock->block_queue));
        (lock->lock).owner=&pcb[((lock->block_queue).next)->index];
    }else{
        (lock->lock).status=UNLOCKED;
        (lock->lock).owner=NULL;
    }
}
```

把这个节点移除掉以后，block_queue 内实际上是没有节点的，不能再取 index 了。取 index 导致 pcb 被覆盖，kernel_sp 出错，最后返回时 sepc 取错为 0

二、还有待解决的问题

1. Project4 一开始为了便于调试，我没有使用双核，后来虚存机制基本建立起来后试着启用，但由于时间原因，目前还未能成功，进入 handle_other，中断原因显示为软中断，猜测可能是发送核间中断后某些位设置错误。