基于标签化RISC-V架构的进程共享资源管理系统

操作系统 课程设计

补充报告

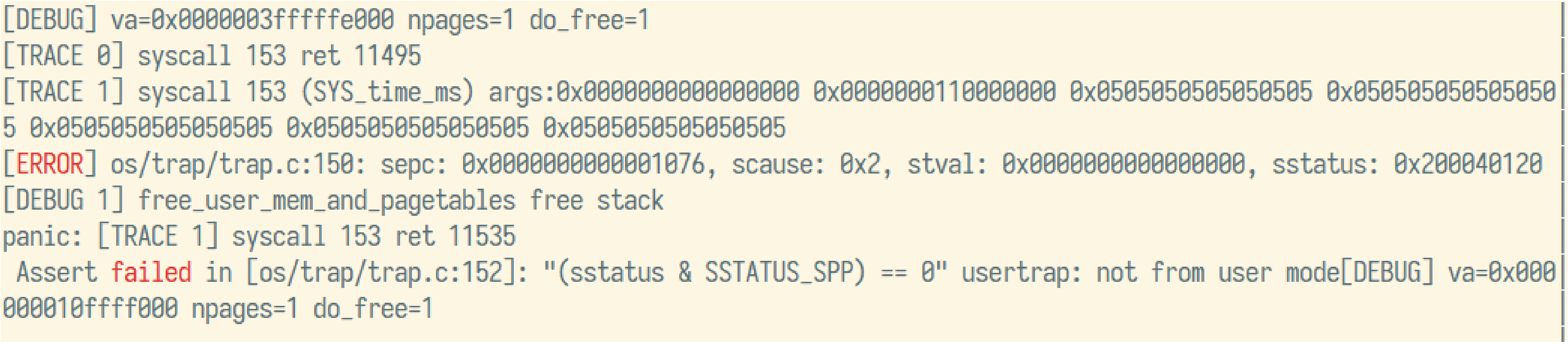
尤予阳
2021.6.7

目录

- BUG修复
- 功能更新
- 数据修正

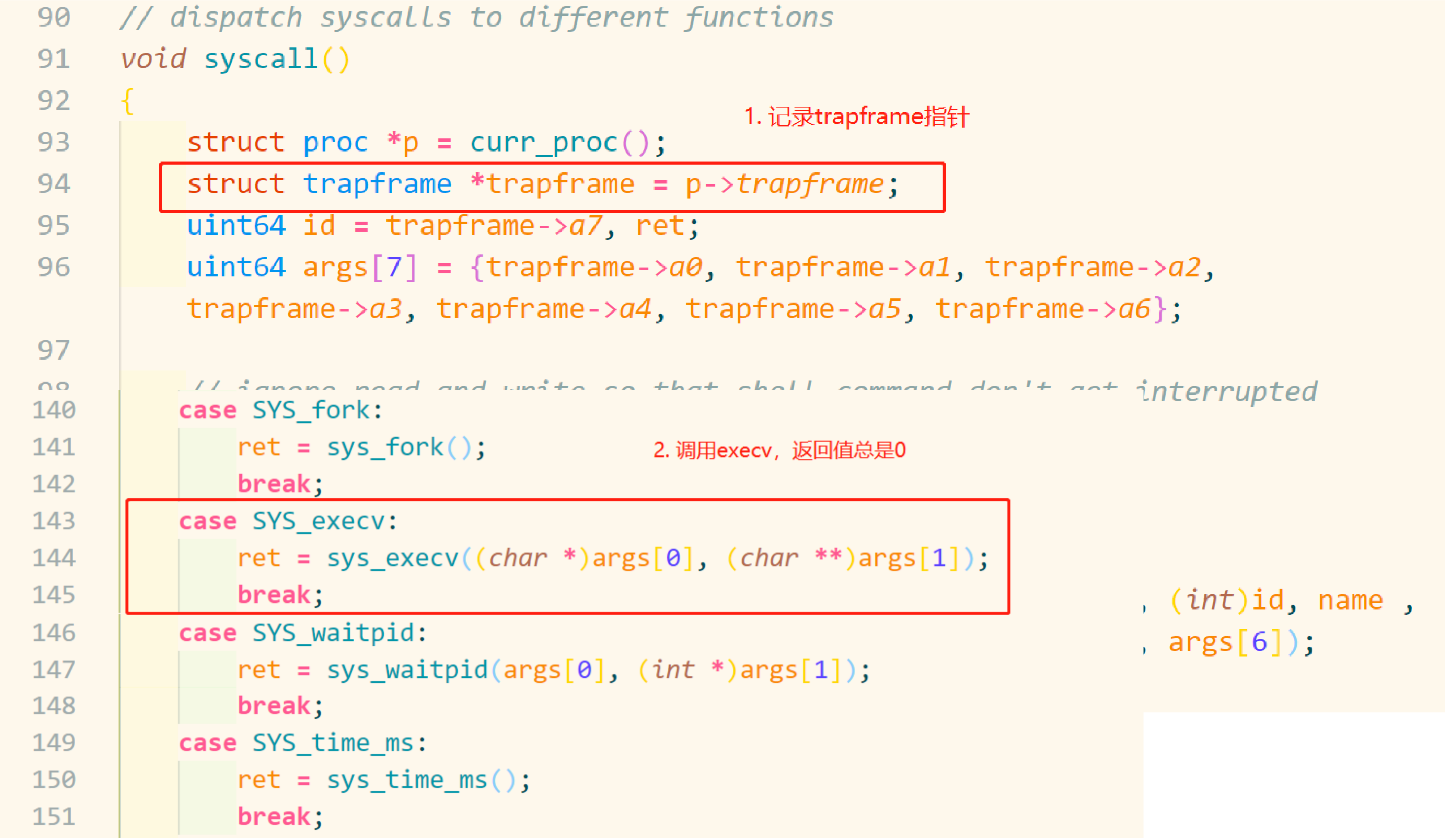
BUG修复

- 悬垂指针导致的错误



```
[DEBUG] va=0x0000003fffffe000 npages=1 do_free=1
[TRACE 0] syscall 153 ret 11495
[TRACE 1] syscall 153 (SYS_time_ms) args:0x0000000000000000 0x0000000110000000 0x0505050505050505 0x050505050505050
5 0x0505050505050505 0x0505050505050505 0x0505050505050505
[ERROR] os/trap/trap.c:150: sepc: 0x0000000000001076, scause: 0x2, stval: 0x0000000000000000, sstatus: 0x200040120
[DEBUG 1] free_user_mem_and_pagetables free stack
panic: [TRACE 1] syscall 153 ret 11535
 Assert failed in [os/trap/trap.c:152]: "(sstatus & SSTATUS_SPP) == 0" usertrap: not from user mode[DEBUG] va=0x000
000010ffff000 npages=1 do_free=1
```

BUG修复



1. 记录trapframe指针

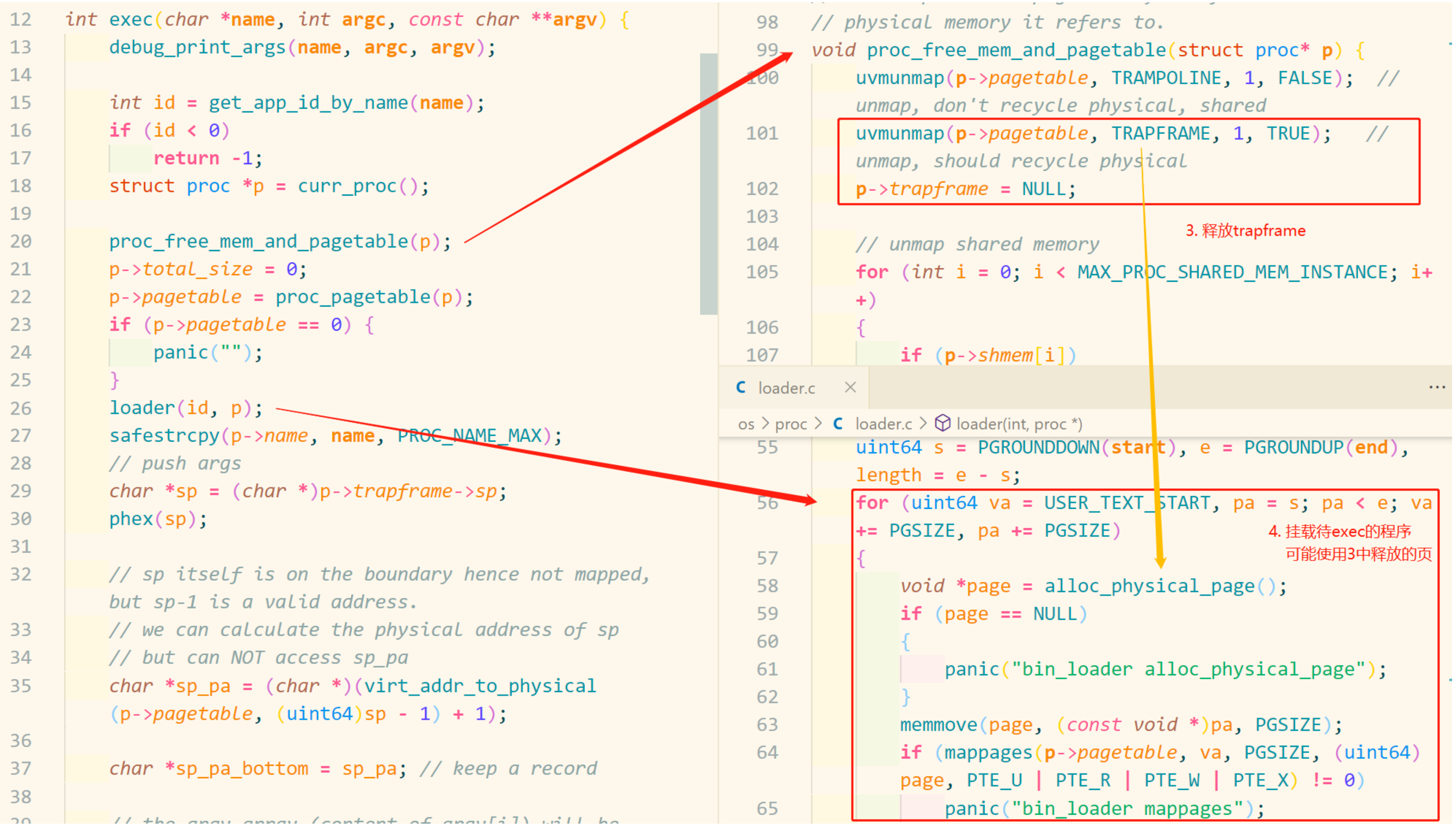
```c
// dispatch syscalls to different functions
void syscall()
{
    struct proc *p = curr_proc();
    struct trapframe *trapframe = p->trapframe;
    uint64 id = trapframe->a7, ret;
    uint64 args[7] = {trapframe->a0, trapframe->a1, trapframe->a2,
    trapframe->a3, trapframe->a4, trapframe->a5, trapframe->a6};

    // ignore read and write so that shell command don't get interrupted
```

2. 调用execv，返回值总是0

```c
    case SYS_fork:
        ret = sys_fork();
        break;
    case SYS_execv:
        ret = sys_execv((char *)args[0], (char **)args[1]);
        break;
    case SYS_waitpid:
        ret = sys_waitpid(args[0], (int *)args[1]);
        break;
    case SYS_time_ms:
        ret = sys_time_ms();
        break;
```

```c
, (int)id, name ,
, args[6]);
```

BUG修复



```c
int exec(char *name, int argc, const char **argv) {
    debug_print_args(name, argc, argv);

    int id = get_app_id_by_name(name);
    if (id < 0)
        return -1;
    struct proc *p = curr_proc();

    proc_free_mem_and_pagetable(p);
    p->total_size = 0;
    p->pagetable = proc_pagetable(p);
    if (p->pagetable == 0) {
        panic("");
    }
    loader(id, p);
    safestrcpy(p->name, name, PROC_NAME_MAX);
    // push args
    char *sp = (char *)p->trapframe->sp;
    phex(sp);

    // sp itself is on the boundary hence not mapped, but sp-1 is a valid address.
    // we can calculate the physical address of sp
    // but can NOT access sp_pa
    char *sp_pa = (char *)(virt_addr_to_physical(p->pagetable, (uint64)sp - 1) + 1);

    char *sp_pa_bottom = sp_pa; // keep a record

    // the argv array (content of argv[i]) will be
```

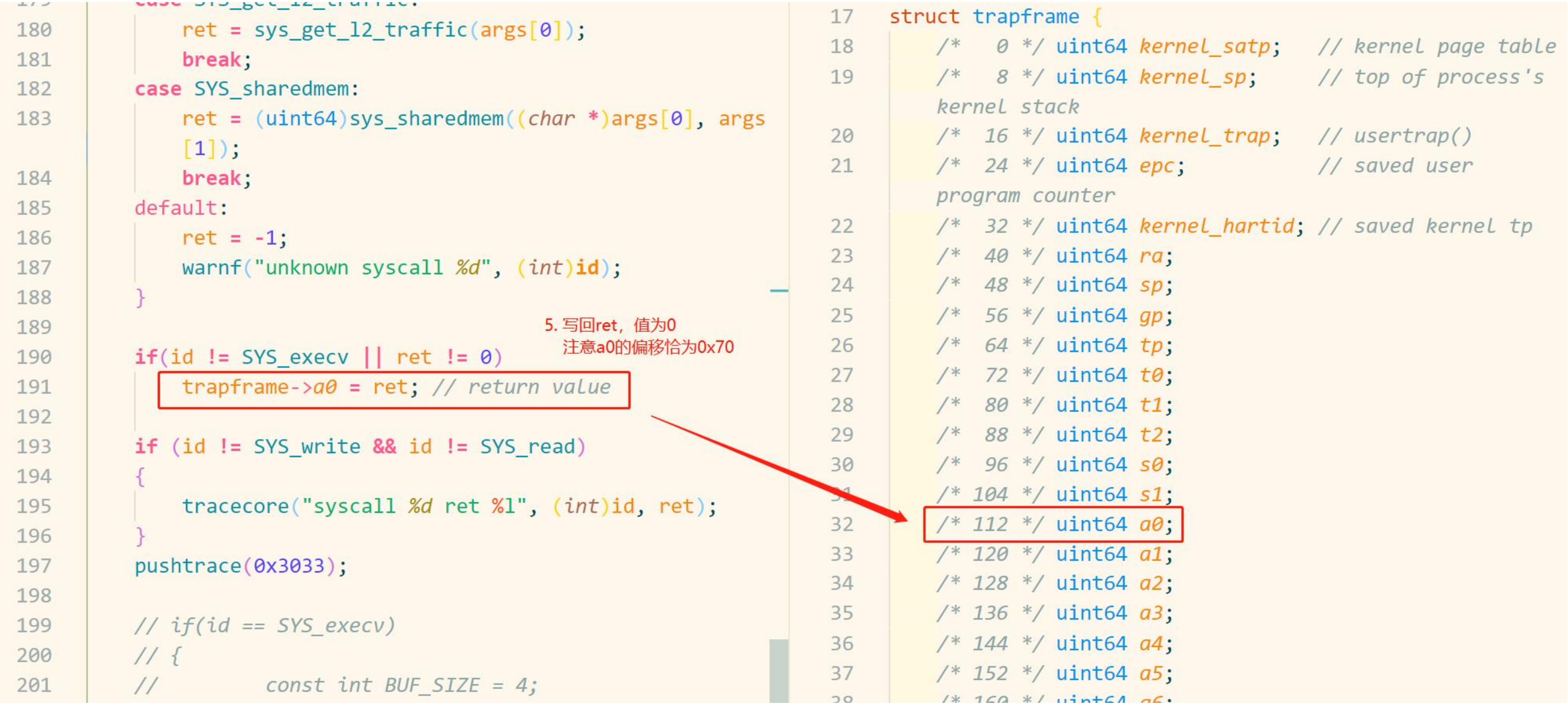
```c
// physical memory it refers to.
void proc_free_mem_and_pagetable(struct proc* p) {
    uvmunmap(p->pagetable, TRAMPOLINE, 1, FALSE);  // unmap, don't recycle physical, shared
    uvmunmap(p->pagetable, TRAPFRAME, 1, TRUE);   // unmap, should recycle physical
    p->trapframe = NULL;

    // unmap shared memory
    for (int i = 0; i < MAX_PROC_SHARED_MEM_INSTANCE; i++)
    {
        if (p->shmem[i])
```

3. 释放trapframe

```c
    uint64 s = PGROUNDDOWN(start), e = PGROUNDUP(end), length = e - s;
    for (uint64 va = USER_TEXT_START, pa = s; pa < e; va += PGSIZE, pa += PGSIZE)
    {
        void *page = alloc_physical_page();
        if (page == NULL)
        {
            panic("bin_loader alloc_physical_page");
        }
        memmove(page, (const void *)pa, PGSIZE);
        if (mappages(p->pagetable, va, PGSIZE, (uint64)page, PTE_U | PTE_R | PTE_W | PTE_X) != 0)
            panic("bin_loader mappages");
```

4. 挂载待exec的程序
可能使用3中释放的页

BUG修复



```
    case SYS_get_l2_traffic:
        ret = sys_get_l2_traffic(args[0]);
        break;
    case SYS_sharedmem:
        ret = (uint64)sys_sharedmem((char *)args[0], args[1]);
        break;
    default:
        ret = -1;
        warnf("unknown syscall %d", (int)id);
    }

    if(id != SYS_execv || ret != 0)
        trapframe->a0 = ret; // return value

    if (id != SYS_write && id != SYS_read)
    {
        tracecore("syscall %d ret %l", (int)id, ret);
    }
    pushtrace(0x3033);

    // if(id == SYS_execv)
    // {
    //          const int BUF_SIZE = 4;
```

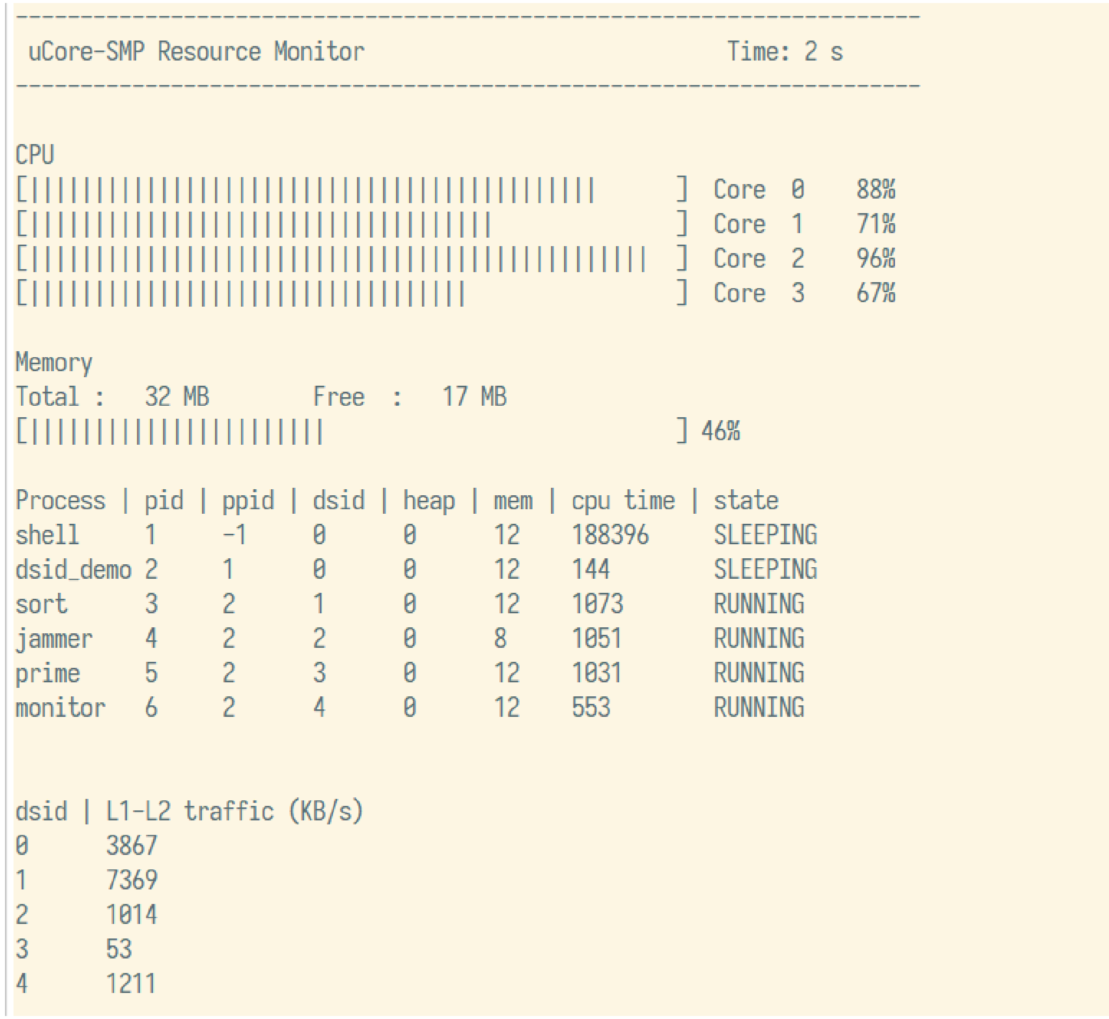
5. 写回ret，值为0
注意a0的偏移恰为0x70

```
struct trapframe {
    /*   0 */ uint64 kernel_satp;   // kernel page table
    /*   8 */ uint64 kernel_sp;     // top of process's kernel stack
    /*  16 */ uint64 kernel_trap;   // usertrap()
    /*  24 */ uint64 epc;           // saved user program counter
    /*  32 */ uint64 kernel_hartid; // saved kernel tp
    /*  40 */ uint64 ra;
    /*  48 */ uint64 sp;
    /*  56 */ uint64 gp;
    /*  64 */ uint64 tp;
    /*  72 */ uint64 t0;
    /*  80 */ uint64 t1;
    /*  88 */ uint64 t2;
    /*  96 */ uint64 s0;
    /* 104 */ uint64 s1;
    /* 112 */ uint64 a0;
    /* 120 */ uint64 a1;
    /* 128 */ uint64 a2;
    /* 136 */ uint64 a3;
    /* 144 */ uint64 a4;
    /* 152 */ uint64 a5;
    /* 160 */ uint64 a6;
```

功能更新

- 在FPGA PS部分增加u-Boot和tmux的启动脚本，简化启动流程
- 增加shell和exec能传递的参数数量上限
- 修改用户程序后，仅重新链接，无需重新编译内核
- 在monitor中增加监测L1和L2之间的流量功能

功能更新



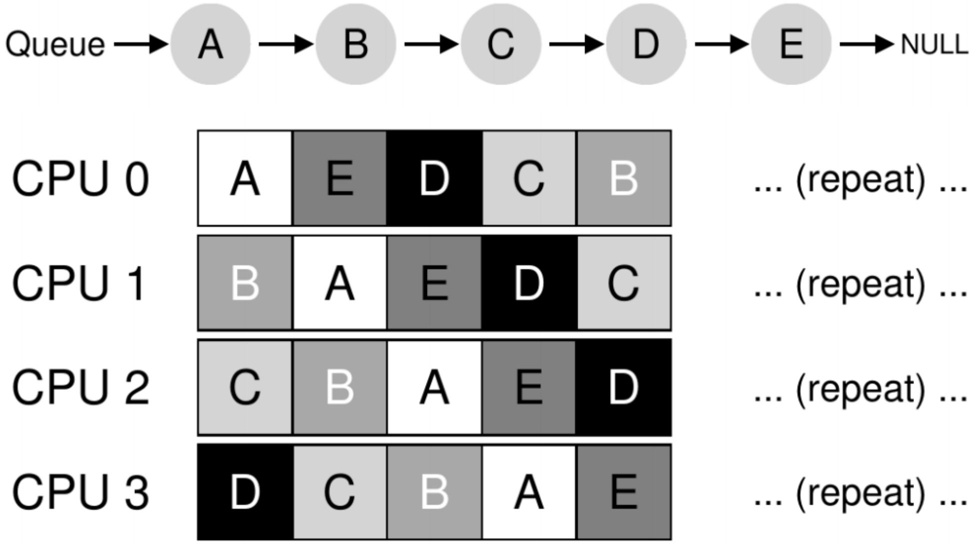
```
----------------------------------------------------------------------
 uCore-SMP Resource Monitor                            Time: 2 s
----------------------------------------------------------------------

CPU
[||||||||||||||||||||||||||||||||||||||||||||      ]  Core  0    88%
[||||||||||||||||||||||||||||||||||||              ]  Core  1    71%
[||||||||||||||||||||||||||||||||||||||||||||||||  ]  Core  2    96%
[||||||||||||||||||||||||||||||||||                ]  Core  3    67%

Memory
Total :   32 MB        Free  :   17 MB
[|||||||||||||||||||||||                           ] 46%

Process | pid | ppid | dsid | heap | mem | cpu time | state
shell     1     -1     0      0      12    188396     SLEEPING
dsid_demo 2     1      0      0      12    144        SLEEPING
sort      3     2      1      0      12    1073       RUNNING
jammer    4     2      2      0      8     1051       RUNNING
prime     5     2      3      0      12    1031       RUNNING
monitor   6     2      4      0      12    553        RUNNING


dsid | L1-L2 traffic (KB/s)
0      3867
1      7369
2      1014
3      53
4      1211
```

数据修正

- 单队列调度，进程在核间切换导致性能下降
  - prime：单独运行8.4s，四个同时运行7.8s
  - sort：单独运行7.7s，与jammer同时运行5.8s，与三个prime同时运行4.3s

数据修正

单队列调度



Queue → A → B → C → D → E → NULL

CPU 0	A	E	D	C	B	... (repeat) ...
CPU 1	B	A	E	D	C	... (repeat) ...
CPU 2	C	B	A	E	D	... (repeat) ...
CPU 3	D	C	B	A	E	... (repeat) ...

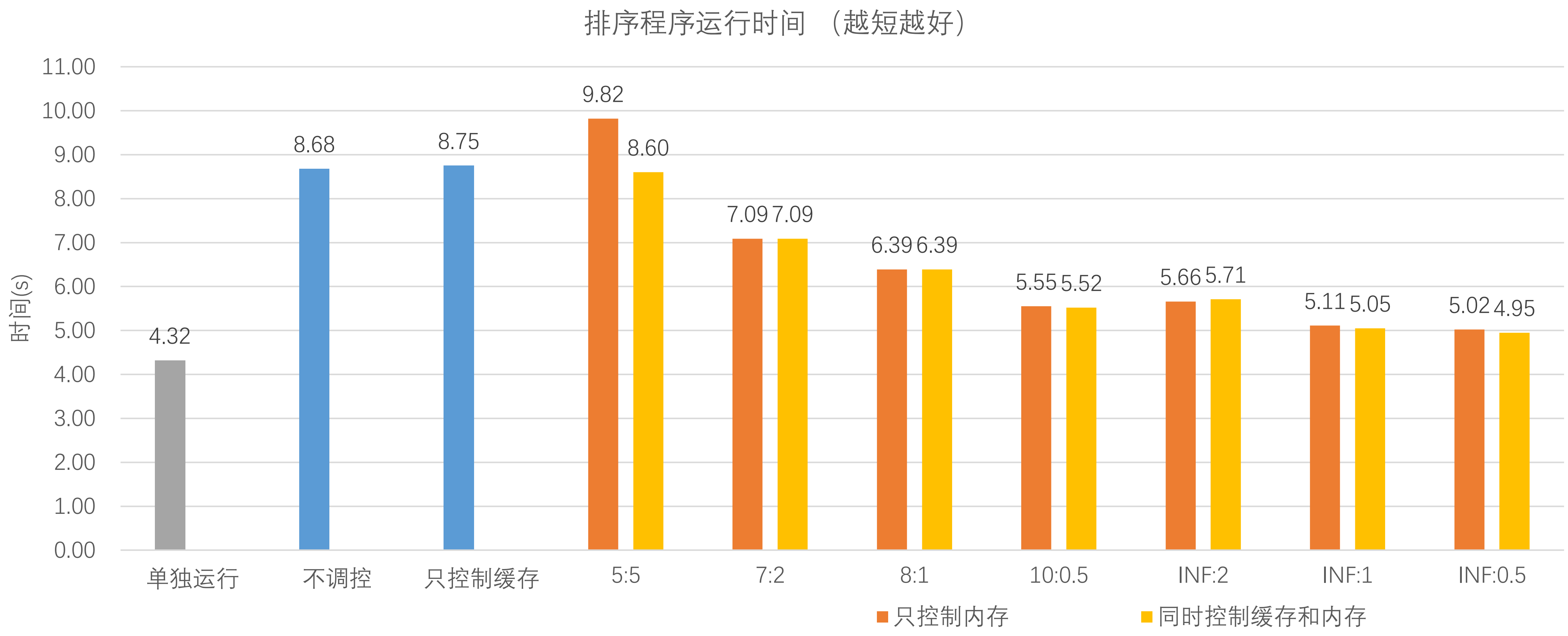
单队列多处理器调度 (SQMS)

- 缺乏可扩展性 (scalability)
- 缓存亲和性 (cache affinity) 弱

数据修正

- 单队列调度，进程在核间切换导致性能下降
  - prime：单独运行8.4s，四个同时运行7.8s
  - sort：单独运行7.7s，与jammer同时运行5.8s，与三个prime同时运行4.3s
- 缓解方法
  - 测试时加入prime，使得同时运行的进程数与核数相等
- 增加测试场景
  - 限制jammer带宽到0.5MB/s
  - 不限制sort带宽

数据修正


排序程序运行时间 （越短越好）
时间(s)
11.00
10.00
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
4.32
8.68
8.75
9.82
8.60
7.09
7.09
6.39
6.39
5.55
5.52
5.66
5.71
5.11
5.05
5.02
4.95
单独运行
不调控
只控制缓存
5:5
7:2
8:1
10:0.5
INF:2
INF:1
INF:0.5
■ 只控制内存
■ 同时控制缓存和内存

感谢聆听！