

2012543-刘沿辰-PA4报告

学号：2012543

姓名：刘沿辰

专业：计算机科学与技术

指令集：RISC-V 32

日期：2023.6.7

实验目的

- 实现多道程序系统
- 支持riscv32的分页管理
- 实现抢占式多任务系统

实验内容

- 第一阶段：实现基本多道程序系统，支持带参数的仙剑奇侠传与Hello内核线程的分时运行
- 第二阶段：实现支持分页管理的多道程序系统
- 第三阶段：实现抢占式分时多任务系统, 提交完整的实验报告

实验过程

实现上下文切换

首先需要实现CTE中的kcontext函数：

```
Context *kcontext(Area kstack, void (*entry)(void *), void
*arg) {
    Context *c = (Context *)((uint8_t *)(kstack.end) -
sizeof(Context) - sizeof(uintptr_t));
    c->mepc = (uintptr_t)entry;
    c->mstatus = 0x1800;
    c->gpr[10] = (uintptr_t)arg;
    return c;
}
```

然后在Nanos-lite中实现两个相关函数：

```
void context_kload(PCB* create_pcb,void (*entry)(void*),void
*arg){
    Area stack={create_pcb->stack,create_pcb-
>stack+STACK_SIZE};
    create_pcb->cp=kcontext(stack,entry,arg);
}

Context* schedule(Context *prev) {
    current->cp=prev;
    current=&pcb[0];
    return current->cp;
}
```

而在trap.S中，只需要在调用 `__am_asm_trap()` 之后添加 `mv a0, sp` 即可

即可看到hello_fun的运行结果：

```
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,53,schedule] Set pcb0
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg
'' for the 1711th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,53,schedule] Set pcb0
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg
'' for the 1712th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,53,schedule] Set pcb0
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg
'' for the 1713th time!
```

短时间内运行了上千次！

对于带参数的上下文，只需要创建第二个线程：

```

void init_proc() {
    size_t num1 = 1, num2 = 2;
    context_kload(&pcb[0], hello_fun, &num1);
    context_kload(&pcb[1], hello_fun, &num2);
    switch_boot_pcb();

    Log("Initializing processes...");
}

```

让schedule交换返回上下文:

```

bool ret = false;

Context* schedule(Context *prev) {
    current->cp = prev;
    if (ret) {
        Log("Set pcb1!");
        current = &pcb[1];
    }
    else {
        Log("Set pcb0!");
        current = &pcb[0];
    }
    ret = !ret;
    return current->cp;
}

```

即可交替运行程序:

```

[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906508' for the 265th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,61,schedule] Set pcb0!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906512' for the 266th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,57,schedule] Set pcb1!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906508' for the 266th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,61,schedule] Set pcb0!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906512' for the 267th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,57,schedule] Set pcb1!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906508' for the 267th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,61,schedule] Set pcb0!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906512' for the 268th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,57,schedule] Set pcb1!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906508' for the 268th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,61,schedule] Set pcb0!
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos-lite with arg '-2117906512' for the 269th time!
[/home/yyy/Documents/ics2022/nanos-lite/src/irq.c,9,do_event] new yield!

```

实现多道通信系统

首先是用户上下文切换的实现：

```

Context *ucontext(AddrSpace *as, Area kstack, void *entry) {
    Context* c = (Context*)(kstack.end) - 1;
    c->mepc=(uintptr_t)entry;
    return c;
}

```

考虑到用户进程需要用到loader，因此将函数的实现与 `naive_uload()` 函数一起放到 `loader.c` 文件中，如下：

```

void context_uload(PCB* pcb,const char *filename){
    uintptr_t entry=loader(pcb,filename);
    Area stack={pcb->stack,pcb->stack+STACK_SIZE};
    pcb->cp=ucontext(&pcb->as,stack,(void*)entry);
}

```

找到 `_start` 文件，如下（未作修改）：

```

.globl _start
_start:
    move s0, zero
    jal call_main

```

这样就可以看到两个程序共同执行了：



可以看到两个程序切换执行！当然，仙剑奇侠传的运行速度也因此断崖式下滑，变得非常卡顿。

PA4第一阶段到此结束。

理解分页机制

x86的分页机制依然是典中典：

- x86将物理内存划分为了大小相同的page frame，并将虚拟地址映射到物理地址上，由此实现内存的保护与虚拟内存。
- 每个进程都有自己的页表，页表通常根据isa的长度而划分为多级结构。
- 页表的根节点是页目录，每个页目录项都指向一个页表，而页表的最后一级为页表页，每个页表就所引到一个page frame。

页的访问过程：

- 当一个进程访问虚拟地址时，处理器根据虚拟地址的高位查找页目录项，通过中间位查找页表项，最后通过低位映射到具体的物理地址
- 得到这个地址后，若页目录表项or页表项不存在，则会让该进程阻塞，操作系统通过页面替换算法来将page frame装载到内存中

这真是，泰裤辣！

在NEMU中实现riscv32的分页机制

由于x86指令集和riscv32指令集的分页都是硬件行为，所以好久不见的nemu又回来了！

首先在nanos-lite中开启**HAS_VME**宏，使用**vme_init**函数对分页机制进行初始化。此时直接运行刚才的双线程序，则会出现如下报错：

```
[/home/yyy/Documents/ics2022/nanos-lite/src/mm.c:26,init_mm] free physical pages starting from 00053C18  
[src/isa/riscv32/inst.c:27 get_csr] Invalid csr code!
```

缺少CSR寄存器！经RTFM，补全satp寄存器，并对硬件进行一定的修改。

由于时间和技术限制，我在此处完成的分页机制暂时无法正常运行，代码进行了回溯，不影响后续运行。

PA4第二阶段到此结束。

抢占多任务

分时多任务的第二类是抢占多任务，其基于硬件中断进行强行上下文切断。

首先在cpu结构体中添加布尔INTR成员：

```
typedef struct {
    word_t gpr[32];
    vaddr_t pc;
    riscv32_System_Registers sr;
    bool INTR;
} riscv32_CPU_state;
```

实际上这是一个CPU引脚，可以用来引发中断。

然后在 `dev_raise_intr` 中切换引脚状态：

```
void dev_raise_intr() {
    cpu.INTR = true;
}
```

完善轮询函数 `isa_query_intr`：

```
word_t isa_query_intr() {
    if (cpu.INTR == true) {
        cpu.INTR = false;
        return IRQ_TIMER;
    }
    return INTR_EMPTY;
}
```

然后在每一条CPU指令执行结束后都检查是否中断：


```

static void execute(uint64_t n) {
    Decode s;
    for (; n > 0; n --) {
        exec_once(&s, cpu.pc);
        .....
        word_t intr = isa_query_intr();
        if (intr != INTR_EMPTY) {
            cpu.pc = isa_raise_intr(intr, cpu.pc);
        }
    }
}

```

同时，在软件上支持时钟中断事件：

```

static Context* do_event(Event e, Context* c) {
    switch (e.event) {
        .....
        case 5: c = schedule(c); c->mepc += 4; break;
        default: panic("Unhandled event ID = %d", e.event);
    }
    return c;
}

```

这样就可以在收到中断时强制当前程序让出CPU，使得其他程序来执行了！

```
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos  
-lite with arg '-2117906492' for the 1523th time!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos  
-lite with arg '-2117906492' for the 1524th time!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos  
-lite with arg '-2117906492' for the 1525th time!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,25,hello_fun] Hello World from Nanos  
-lite with arg '-2117906492' for the 1526th time!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!  
[/home/yyy/Documents/ics2022/nanos-lite/src/proc.c,64,schedule] Control change!
```

riscv32-NEMU



理解计算机系统

尝试在Linux中编写如下程序：

```
int main() {  
    char *p = "abc";  
    p[0] = 'A';  
    return 0;  
}
```

程序发生了段错误。实际上，字符串“abc”早在编译链接时就被放入了ELF文件格式的“只读”数据段。

此时如果我们试图去对其进行读写，硬件的MMU发现当前进程的权限不足以访问该区域，由此发生了权限不足的访问行为。这个时候为了防止磁盘等地的数据遭到不可避免的打击，直接段错误让核心崩掉是最安全的行为。