ics-PA4实验报告

高辰潇
人工智能学院
SID：181220014

一、实验进度

- 完成了PA4中所有必做题。

二、必答题：分时多任务的具体过程

- 分时多任务在ics-PA中主要涉及到两个机制：分页机制和硬件中断。

分页机制

- 本质而言，分页机制实现了进程的虚拟地址空间到物理地址空间的映射。借助于这样的映射，各进程可分别使用各自的虚拟地址空间而不会相互影响
- 首先，nanos-lite在调用context_uload为仙剑奇侠传和hello生成两个不同的进程（实际上是生成进程描述符PCB）时，依次进行了以下工作：
  - 由于不同的用户进程的虚拟地址空间应当是独立的，因此要使用_protect为进程创建各自的用户进程的虚拟地址空间。

    ```
    void context_uload(PCB* pcb, void* entry) {
        _protect(&pcb->as);  // 创建用户进程的虚拟地址空间
        ...
    }
    ```

    _protect会申请新的页目录作为当前进程的页目录，并将内核地址空间的第一级页表信息拷贝到当前进程中，这样就建立起了内核区域的地址映射。
  - 调用loader，将应用程序需要加载的数据加载到内存中。在加载时，调用_map函数建立这一部分数据的虚拟-物理地址映射关系。

```
int _map(_AddressSpace *as, void *va, void *pa, int prot) {
    uint32_t base = (uint32_t) as->ptr;                          // 页目录基址
    PDE* pde =(PDE*) ((base&~0xFFF) | sizeof(PDE)*((uint32_t)va>>22));
    if ((*pde & 0b1)!=1) *pde = (uint32_t) pgalloc_usr(1)|0b1;   // 设置一级页表

    PDE PT_BASE = *pde;                                          // 页表基址

    PTE* pte =(PTE*) ((PT_BASE&~0xFFF) | sizeof(PTE)*((uint32_t)va<<10>>22));

    if ((*pte&0b1)!=1) *pte = (uint32_t)pa|0b1;            // 设置二级页表
    return 0;
}
```

- 在进程运行时,对于各进程申请的堆区，我们也需要实现地址映射。每个进程描述符的max_brk描述了该进程曾经达到过的最大的program break位置。只需要保证在这program break之下的所有虚拟地址空间我们都已经分配了物理页并建立了映射关系即可.
  mm_brk实现了堆区的空间管理和地址映射。

```
int mm_brk(uintptr_t brk, intptr_t increment) {
    extern PCB* current;
    uintptr_t new_brk = brk+increment;
    if (current->brk == 0) current->max_brk = brk;

    if(new_brk > current->max_brk) {                 // 超出max_brk
        int left = new_brk - current->max_brk;       // 超出max_brk的字节数
        int res = 4096-(current->max_brk%4096);      // 上一次申请的页的剩余空间
        left = left-res;

        void* va = (void*) (((current->max_brk-1)/4096+1)<<12);
        while(left > 0) {                            // 为超出页的数据申请新页
            void* pa = new_page(1);
            _map(&current->as, va, pa, 1);
            va += (1<<12);
            left -= 4096;
        }
        current->max_brk = new_brk;                  // 更新max_brk
    }
    return 0;
}
```

- 在虚拟-物理地址映射建立后，访问内存时只需要根据给出的虚拟地址，调用page_translate换算得到物理地址进行访问即可，不再赘述。
- 总而言之，分页机制的关键在于**建立并维护虚拟-物理地址空间映射**。PA中实现了内核空间、需加载的数据段以及堆空间的映射。并且，对于不同的用户进程，为它们各自维护一份这样的映射关系。分页机制一方面使得多个进程能同时将自己的数据加载在物理内存中，另一方面也保证了进程的数据安全与独立性，可以说是分时多任务的“空间基础”。

硬件中断

- 分时多任务指的是系统能以一定频率在所有进程之间切换，然而分页机制仅保证了多个进程能在物理内存中加载数据，却并未保证所有进程都能被“同时”运行或作出响应。因此PA中使用**时钟中断**机制指示系统在各个进程之间来回切换。
- 具体实现：以仙剑奇侠传和hello程序为例
  - 在时钟中断到来后，cpu.INTR被设置为高电平
  - cpu在执行完一条指令后调用isa_query_intr进行检测。如果当前处于开中断状态并且INTR为高电平，则发生时钟中断。将cpu设置为关中断状态，并通过CTE生成异常号为32的事件.

```
vaddr_t exec_once(void) {
    ...
    if (isa_query_intr()) {
        update_pc();
    }
    return decinfo.seq_pc;

bool isa_query_intr(void) {
    if ((cpu.eflags.IF==1)&&cpu.INTR) {          // cpu开中断且INTR为高电平
        cpu.INTR = false;                        // 关中断
        raise_intr(IRQ_TIMER, cpu.pc);           // 设置中断
        return true;
    }
    return false;
}
```

  - nanos-lite捕捉到事件_EVENT_TRQ_TIMER后，调用_yield。_yield实际上会进入异常号为0x81的异常处理程序，操作系统会通过schedule函数进行规划，切换当前进程current，并同时切换上下文。下面的代码中通过维护counter变量，使得hello程序和仙剑奇侠传运行的时间比例为1:1000. fg_pcb为当前前台程序的代号。

```
_Context* schedule(_Context *prev) {
    static int counter=0;
    if (counter==0) {
        current->cp = prev;
        current = &pcb[0];
    }
    else {
        current->cp = prev;
        current = &pcb[fg_pcb];
    }
    counter = (counter+1)%1000;
    return current->cp;
    }
```

  - 将上下文从进程A的上下文切换到B的上下文后，异常处理程序结束时系统会将B的上下文信息恢复到寄存器中。从而，系统从上次进程B中断的地方继续运行下去，实现了一次完整的进程

切换。

- 在时钟中断的控制下，进程将进行周期性地切换，从而在一定时间内每个进程都将被运行、都将作出响应。这也就是分时多任务的内涵所在。