

Lab3 进程切换

黄涛

dg1533011@mail.nju.edu.cn

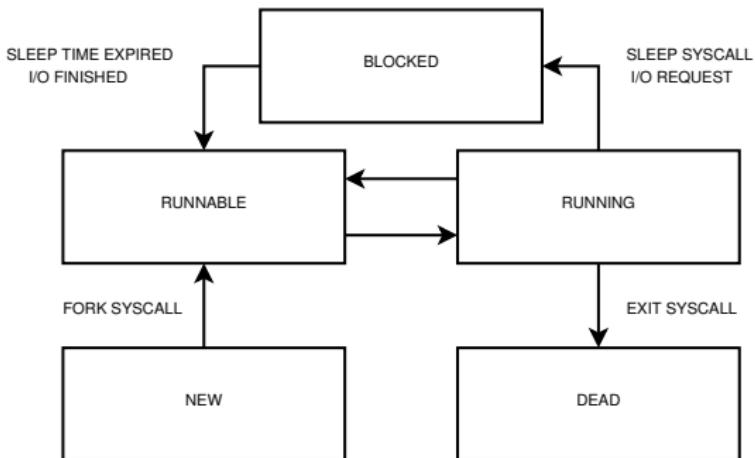
2018-4-16

- 内核：实现进程切换机制，并提供系统调用 fork、sleep、exit
- 库：对上述系统调用进行封装
- 用户：对上述系统调用进行测试

- Bootloader 从实模式进入保护模式, 加载内核至内存, 并跳转执行
- 内核初始化 IDT, 初始化 GDT, 初始化 TSS, 初始化串口, 初始化 8259A, ...
- 启动时钟源
- 加载用户程序至内存
- 初始化内核 IDLE 线程的进程控制块 (Process Control Block), 初始化用户程序的进程控制块
- 切换至用户程序的内核堆栈, 弹出用户程序的现场信息, 返回用户态执行用户程序

进程与线程

- 进程为操作系统资源分配的单位，每个进程都有独立的地址空间（代码段、数据段），独立的堆栈，独立的进程控制块
- 线程作为任务调度的基本单位，与进程的唯一区别在于其地址空间并非独立，而是与其他线程共享



- FORK 系统调用用于创建子进程

- 内核需要为子进程分配一块独立的内存，将父进程的地址空间、用户态堆栈完全拷贝至子进程的内存中
- 为子进程分配独立的进程控制块，完成对子进程的进程控制块的设置
- 若子进程创建成功，则对于父进程，该系统调用的返回值为子进程的 pid，对于子进程，其返回值为 0
- 若子进程创建失败，该系统调用的返回值为 -1

```
...
int ret = fork();
if (ret == 0) {
    ... // Child Process
} else if (ret != -1) {
    ... // father Process
} else {
    ... // fork failed
}
...
```

- SLEEP 系统调用用于进程主动阻塞自身
 - 内核需要将该进程由 RUNNING 状态转换为 BLOCKED 状态
 - 设置该进程的 SLEEP 时间片
 - 切换运行其他 RUNNABLE 状态的进程
- EXIT 系统调用用于进程主动销毁自身
 - 内核需要将该进程由 RUNNING 状态转换为 DEAD 状态
 - 回收分配给该进程的内存、进程控制块等资源
 - 切换运行其他 RUNNABLE 状态的进程

- 若没有处于 RUNNABLE 状态的进程可供切换，则需要切换至内核 IDLE 线程
 - 内核 IDLE 线程调用 waitForInterrupt() 执行 hlt 指令
 - hlt 指令会使得 CPU 进入暂停状态，直到外部硬件中断产生

```
static inline void waitForInterrupt() {
    asm volatile("hlt");
}
...
while(1)
    waitForInterrupt();
...
```

进程控制块

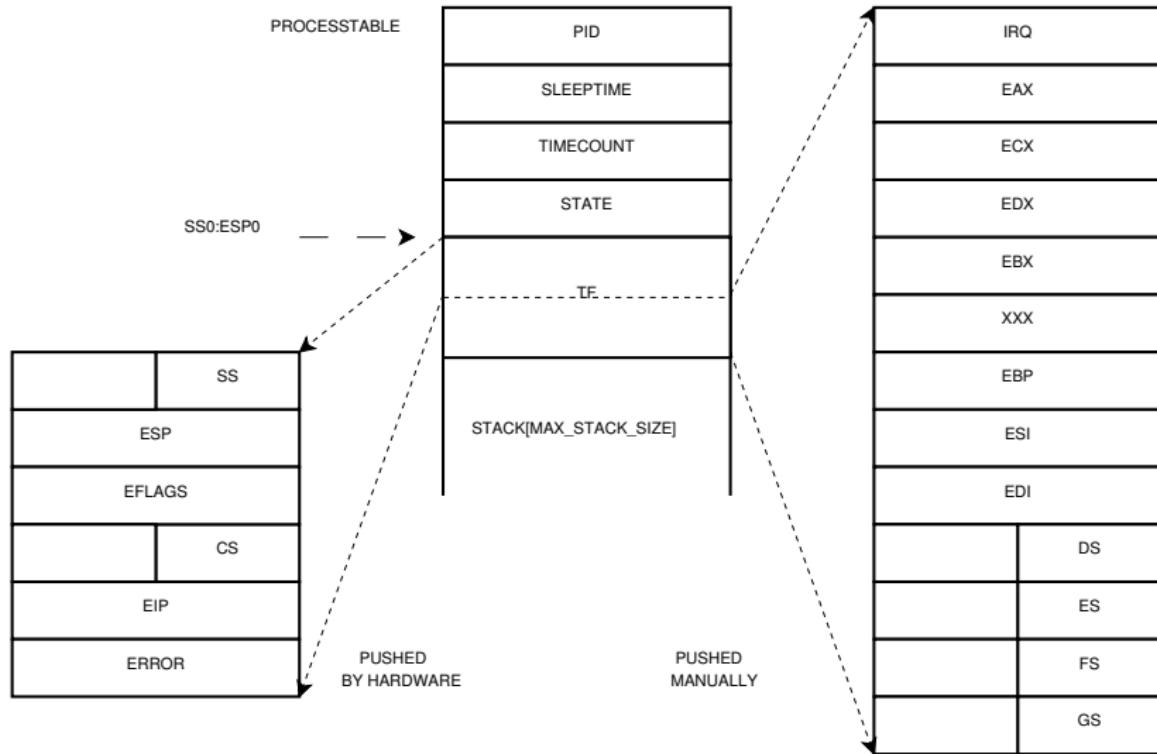
- 使用 ProcessTable 这一数据结构作为进程控制块记录每个进程的信息

```
struct TrapFrame {
    uint32_t gs, fs, es, ds;
    uint32_t edi, esi, ebp, xxx, ebx, edx, ecx, eax;
    uint32_t irq; // 中断号
    uint32_t error; // Error Code
    uint32_t eip, cs, eflags, esp, ss;
};

struct ProcessTable {
    uint32_t stack[MAX_STACK_SIZE]; // 内核堆栈
    struct TrapFrame tf;
    int state;
    int timeCount;
    int sleepTime;
    uint32_t pid;
    ...
};

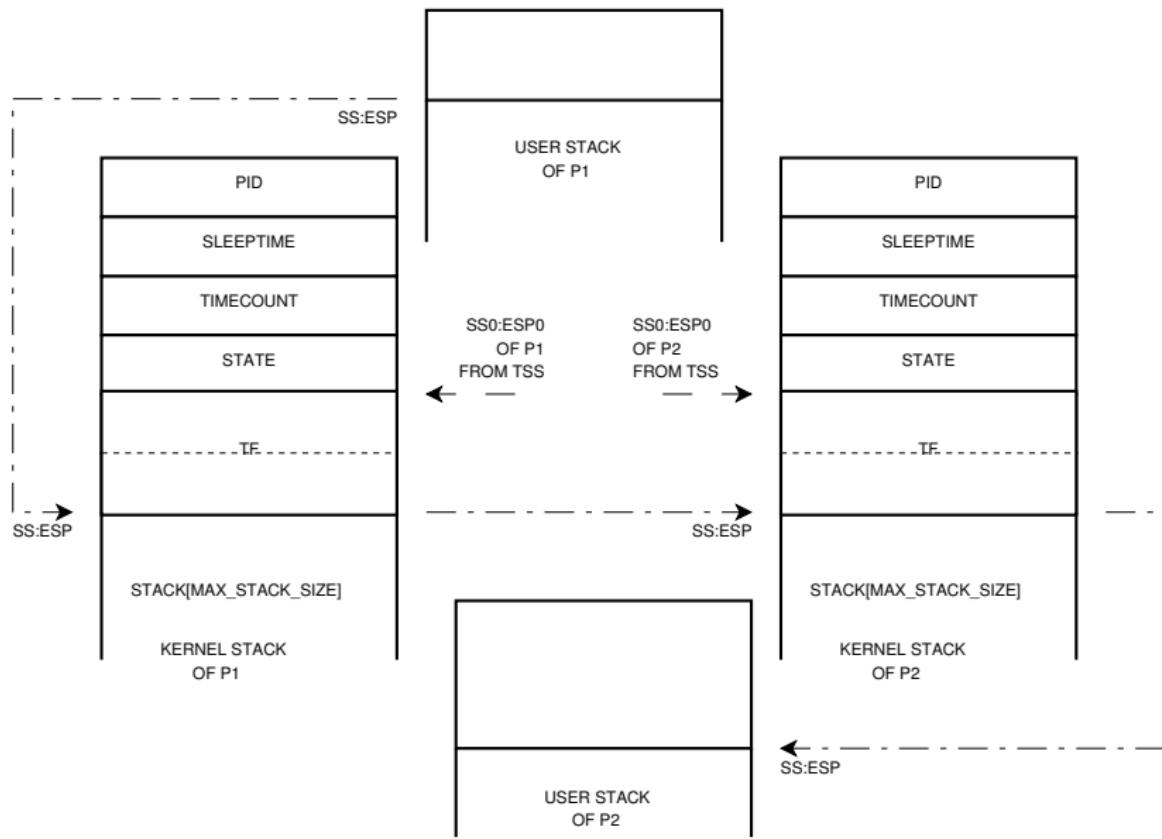
struct ProcessTable pcb[MAX_PCB_NUM];
```

进程控制块



- 时间中断到来后，两个用户态进程 P1、P2 进行进程切换的流程如下
 - 进程 P1 在用户态执行，8253 可编程计时器产生时间中断
 - 依据 TSS 中记录的进程 P1 的 SS0:EPS0，从 P1 的用户态堆栈切换至 P1 的内核堆栈，并将 P1 的现场信息压入内核堆栈中，跳转执行时间中断处理程序
 - 进程 P1 的处理时间片耗尽，切换至就绪状态的进程 P2，并从当前 P1 的内核堆栈切换至 P2 的内核堆栈
 - 从进程 P2 的内核堆栈中弹出 P2 的现场信息，切换至 P2 的用户态堆栈，从时间中断处理程序返回执行 P2

进程切换与堆栈切换



- 由于系统调用的处理时间往往很长，为保证进程调度的公平性，需要在系统调用中开启外部硬件中断，以便当前进程的处理时间片耗尽时，进行进程切换
- 由于可以在系统调用中进行进程切换，因此可能会出现多个进程并发地处理系统调用，对共享资源（例如内核的数据结构，视频显存等等）进行竞争

- 考虑以下场景

- 进程 P1 在内核态处理系统调用，处理视频显存，此时外部硬件中断开启
- 8253 可编程计时器产生一个时间中断
- 在内核态处理系统调用的进程 P1 将现场信息压入 P1 的内核堆栈中，跳转执行时间中断处理程序
- 进程 P1 的处理时间片耗尽，切换至就绪状态的进程 P2，并从当前 P1 的内核堆栈切换至 P2 的内核堆栈
- 从进程 P2 的内核堆栈中弹出 P2 的现场信息，从时间中断处理程序返回执行 P2
- 进程 P2 在内核态处理系统调用，处理视频显存，与进程 P1 形成竞争

中断嵌套与临界区

```
void syscallPrint(struct StackFrame *sf) {
    ...
    for (i = 0; i < size; i++) {
        asm volatile("movb %%es:(%1), %0":=r"(character):"r"(str+i));
        if(character == '\n') {
            displayRow++;
            displayCol = 0;
            if(displayRow == 25) {
                displayRow = 24;
                displayCol = 0;
                scrollScreen();
            }
        }
    else {
        data = character | (0x0c << 8);
        pos = (80*displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)::r"(data), "r"(pos+0xb8000));
        displayCol++;
        if(displayCol == 80) {
            displayRow++;
            displayCol = 0;
            if(displayRow == 25){
                displayRow = 24;
                displayCol = 0;
                scrollScreen();
            }
        }
    }
    asm volatile("int $0x20"); // 测试系统调用嵌套时间中断
}
...
}
```



中断嵌套与临界区

```
001005dc <syscallPrint>:  
    ...  
100606:    or    $0xc,%ah  
100609:    lea   (%ecx,%ecx,4),%edx  
10060c:    shl   $0x4,%edx  
10060f:    add   0x102404,%edx  
100615:    lea   0xb8000(%edx,%edx,1),%edx  
10061c:    mov   %ax,(%edx)  
10061f:    mov   0x102404,%eax  
100624:    inc   %eax  
100625:    mov   %eax,0x102404  
10062a:    cmp   $0x50,%eax  
10062d:    je    10063d  
10062f:    int   $0x20  
100631:    inc   %ebx  
100632:    cmp   %esi,%ebx  
100634:    je    10066c  
100636:    mov   %es:(%ebx),%al  
100639:    cmp   $0xa,%al  
10063b:    jne   100606  
10063d:    inc   %ecx  
10063e:    mov   %ecx,0x102408  
100644:    mov   $0x0,0x102404  
10064e:    cmp   $0x19,%ecx  
100651:    jne   10062f  
    ...  
00102404 <displayCol>:  
102404:    00 00  
    ...  
00102408 <displayRow>:  
102408:    00 00  
    ...
```

- P1 从时钟中断返回，顺序执行 0x100631、0x100632、0x100634、0x100636、0x100639、0x10063b、0x10063d、0x10063e、0x100644、0x10064e、0x100651、0x10062f，再次陷入时间中断，切换至 P2
- P2 从时间中断返回，顺序执行 0x100631、0x100632、0x100634、0x100636、0x100639、0x10063b、0x100606、0x100609、0x10060c、0x10060f、0x100615、0x10061c
- 全局变量 displayRow 的更新产生一致性问题

- 多个进程并发地进行系统调用，对共享资源进行竞争可能会产生一致性问题，带来未知的 BUG
- 在系统调用过程中，对于临界区的代码不宜开启外部硬件中断
- 在系统调用过程中，对于非临界区的代码则可以开启外部硬件中断，允许中断嵌套

作业提交

- 截止时间：2018-4-30 00:00:00