中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

Process Switching

based on Linux3.9

孟宁

Process Switching

- 为了控制进程的执行，内核必须有能力挂起正在CPU上执行的进程，并恢复以前挂起的某个进程的执行，这叫做进程切换、任务切换、上下文切换

进程上下文

- 包含了进程执行需要的所有信息
  - 用户地址空间
    包括程序代码，数据，用户堆栈等
  - 控制信息
    进程描述符，内核堆栈等
  - 硬件上下文

硬件上下文

- 尽管每个进程可以有自己的地址空间，但所有的进程只能共享CPU的寄存器。
- 因此，在恢复一个进程执行之前，内核必须确保每个寄存器装入了挂起进程时的值。这样才能正确的恢复一个进程的执行
- 硬件上下文：
  进程恢复执行前必须装入寄存器的一组数据
  - 包括通用寄存器的值以及一些系统寄存器
    - 通用寄存器如eax，ebx等
    - 系统寄存器如eip，esp，cr3等等

Linux进程上下文

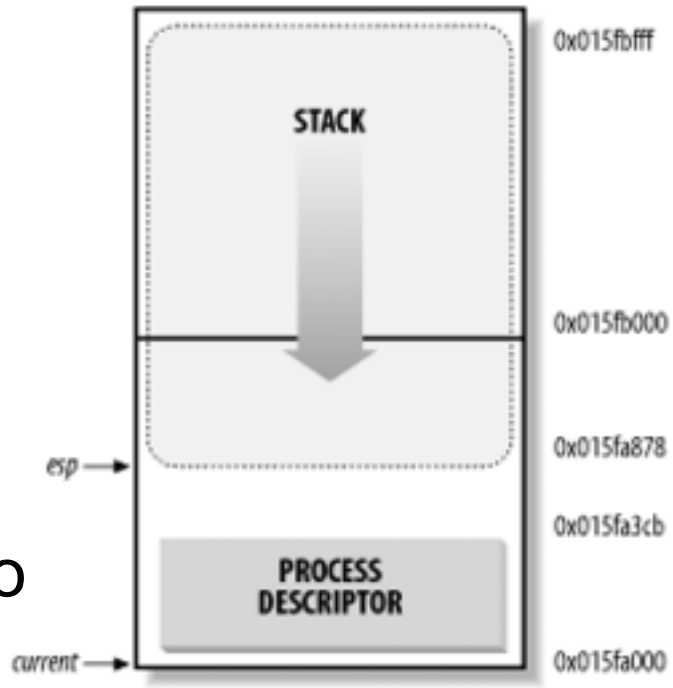
- thread_union(include/linux/sched.h)

```
union thread_union {
        struct thread_info thread_info;
        unsigned long stack[THREAD_SIZE/sizeof(long)];
};
```

- 在linux中一个进程的上下文主要保存在thread_info和task_struct的thread_struct中,其他信息放在内核态堆栈中
- thread_info由体系结构相关部分定义阅读arch/x86/include/asm/thread_info.h
- thread_struct参见arch/x86/include/asm/processor.h

进程的内核堆栈

- Linux为每个进程分配一个8KB大小的内存区域，用于存放该进程两个不同的数据结构：
  - Thread_info
  - 进程的内核堆栈
- 进程处于内核态时使用，不同于用户态堆栈
- 内核控制路径所用的堆栈很少，因此对栈和Thread_info来说，8KB足够了



上下文切换

- schedule()函数选择一个新的进程来运行，并调用context_switch进行上下文的切换，这个宏调用switch_to来进行关键上下文切换
- switch_to利用了prev和next两个参数：
  - prev：指向当前进程
  - next：指向被调度的进程

```
#define switch_to(prev, next, last)                     \
do {                                                    \
    /*                                                  \
     * Context-switching clobbers all registers, so we clobber \
     * them explicitly, via unused output variables.    \
     * (EAX and EBP is not listed because EBP is saved/restored \
     * explicitly for wchan access and EAX is the return value of \
     * __switch_to())                                   \
     */                                                 \
    unsigned long ebx, ecx, edx, esi, edi;              \
                                                        \
    asm volatile("pushfl\n\t"           /* save   flags */ \
                 "pushl %%ebp\n\t"      /* save   EBP   */ \
                 "movl %%esp,%[prev_sp]\n\t"    /* save   ESP  */ \
                 "movl %[next_sp],%%esp\n\t"    /* restore ESP  */ \
                 "movl $1f,%[prev_ip]\n\t"  /* save   EIP  */ \
                 "pushl %[next_ip]\n\t" /* restore EIP  */ \
                 __switch_canary                        \
                 "jmp __switch_to\n"    /* regparm call */ \
                 "1:\t"                                 \
                 "popl %%ebp\n\t"       /* restore EBP  */ \
                 "popfl\n"              /* restore flags */ \
                                                        \
                 /* output parameters */                \
                 : [prev_sp] "=m" (prev->thread.sp),    \
                   [prev_ip] "=m" (prev->thread.ip),    \
                   "=a" (last),                         \
                                                        \
                   /* clobbered output registers: */    \
                   "=b" (ebx), "=c" (ecx), "=d" (edx),  \
                   "=S" (esi), "=D" (edi)               \
                                                        \
                   __switch_canary_oparam               \
                                                        \
                   /* input parameters: */              \
                 : [next_sp]  "m" (next->thread.sp),    \
                   [next_ip]  "m" (next->thread.ip),    \
                                                        \
                   /* regparm parameters for __switch_to(): */ \
                   [prev]     "a" (prev),               \
                   [next]     "d" (next)                \
                                                        \
```

课后作业

- 仔细阅读switch_to宏，参见arch/x86/include/asm/system.h
- 什么时候next进程真正开始执行呢?
- 当__switch_to正常返回时，发生了什么事情?
- __switch_to用来处理其他上下文的切换，此时，使用的堆栈是next进程的堆栈，这个堆栈上没有__switch_to需要的参数prev和next，那是怎么传参呢?

中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

大家不觉得没有问题的工作很枯燥吗?

发现问题比解决问题往往更加困难！

谢谢大家!

参考资料:

《深入理解Linux内核》第三版