多平台多核与任务管理 项目申请书

1. 项目名称

   多平台多核与任务管理

   项目编号：210060236

2. 项目详细方案

   项目主要是对多任务管理的实现，包括 x86/arm/riscv 架构下的任务抽象与实际调度。

   可以分为两个部分：

   1. 对任务的抽象

      这部分内容应该设计成与平台无关，可以参考 linux 早期版本的实现。

      此外在 SimpleKernel 的 TODO 分支中，有一些可用代码：

```
// 进程状态描述
typedef enum task_status {
    // 未初始化
    TASK_UNINIT = 0,
    // 睡眠中
    TASK_SLEEPING = 1,
    // 可运行
    TASK_RUNNABLE = 2,
    // 正在运行
    TASK_RUNNING = 3,
    // 僵尸状态
    TASK_ZOMBIE = 4,
} task_status_t;

// 内核线程的上下文切换保存的信息
// Saved registers for kernel context switches.
// Don't need to save all the segment registers (%cs, etc),
// because they are constant across kernel contexts.
// Don't need to save %eax, %ecx, %edx, because the
```

```
// x86 convention is that the caller has saved them.
// Contexts are stored at the bottom of the stack they
// describe; the stack pointer is the address of the context.
// The layout of the context must match the code in swtch.S.
typedef struct task_context {
    uint32_t eip;
    uint32_t esp;
    uint32_t ebp;
    uint32_t ebx;
    uint32_t ecx;
    uint32_t edx;
    uint32_t esi;
    uint32_t edi;
} task_context_t;

// 进程内存地址结构
typedef struct task_mem {
    // 进程页表
    pgd_t *pgd_dir;
    // 栈顶
    ptr_t stack_top;
    // 栈底
    ptr_t stack_bottom;
    // 内存起点
    ptr_t task_start;
    // 代码段起止
    ptr_t code_start;
    ptr_t code_end;
    // 数据段起止
    ptr_t data_start;
    ptr_t data_end;
    // 内存结束
    ptr_t task_end;
} task_mem_t;

// 进程控制块 PCB
typedef struct task_pcb {
    // 任务状态
    volatile task_status_t status;
    // 任务的 pid
    pid_t pid;
```

```
    // 任务名称
    char *name;
    // 当前任务运行时间
    uint32_t run_time;
    // 父进程指针
    struct task_pcb *parent;
    // 任务的内存信息
    task_mem_t *mm;
    // 任务中断保存的寄存器信息
    pt_regs_t *pt_regs;
    // 任务切换上下文信息
    task_context_t *context;
    // 任务的退出代码
    int32_t exit_code;
} task_pcb_t;
```

根据已有代码，只需要进行少量改动即可实现项目目标。

2. 具体架构的实现

   各个平台切换任务的的方式大同小异，基本思路是保存现场与恢复现场。

   对于 x86 架构，可以使用以下代码进行切换：

```
// task_pcb_t * switch_to(task_pcb_t * curr, task_pcb_t * next, task_pcb_t *
// last);
#define switch_to(prev, next, last)                                          \
    do {                                                                     \
        uint32_t ebx, ecx, edx, esi, edi;                                    \
        __asm__ volatile("pushfl\n\t"                                        \
                         "pushl %%ebp\n\t"                                   \
                         "movl %%esp,%[prev_sp]\n\t"                         \
                         "movl %[next_sp],%%esp\n\t"                         \
                         "movl $1f,%[prev_ip]\n\t"                           \
                         "pushl %[next_ip]\n\t"                              \
                         "jmp __switch_to\n"                                 \
                         "1:\n\t"                                            \
                         "popl %%ebp\n\t"                                    \
                         "popfl\n"                                           \
                         : [ prev_sp ] "=m"(prev->context->esp),             \
                           [ prev_ip ] "=m"(prev->context->eip), "=a"(last), \
                           "=b"(ebx), "=c"(ecx), "=d"(edx), "=S"(esi),       \
                           "=D"(edi)                                         \
                         : [ next_sp ] "m"(next->context->esp),              \
                           [ next_ip ] "m"(next->context->eip),              \
                           [ prev ] "a"(prev), [ next ] "d"(next)            \
                         : "memory");                                        \
    } while (0);
```

riscv 架构：

```
.globl pswitch
pswitch:
        sd ra, 0(a0)
        sd sp, 8(a0)
        sd s0, 16(a0)
        sd s1, 24(a0)
        sd s2, 32(a0)
        sd s3, 40(a0)
        sd s4, 48(a0)
        sd s5, 56(a0)
        sd s6, 64(a0)
        sd s7, 72(a0)
        sd s8, 80(a0)
        sd s9, 88(a0)
        sd s10, 96(a0)
        sd s11, 104(a0)

        ld ra, 0(a1)
        ld sp, 8(a1)
        ld s0, 16(a1)
        ld s1, 24(a1)
        ld s2, 32(a1)
        ld s3, 40(a1)
        ld s4, 48(a1)
        ld s5, 56(a1)
        ld s6, 64(a1)
        ld s7, 72(a1)
        ld s8, 80(a1)
        ld s9, 88(a1)
        ld s10, 96(a1)
        ld s11, 104(a1)

        ret
```

3. 项目开发时间计划

第一周～第二周：熟悉已有代码

第三周～第六周：对任务的抽象

第七周～第八周：完成任务框架

第九周～第十一周：完成各个架构的具体实现

第十二周：撰写总结报告