北京科技大学实验报告

学院：计算机与通信工程学院　专业：计算机科学与技术　班级：计 1703

姓名：　张宝丰　　学号：41724081　　实验日期：2019 年　12 月　1 日

实验名称：操作系统实验 4 线程调度（4 分）

实验目的：以一个教学型操作系统 EOS 为例，深入理解线程（进程）调度的执行时机、过程以及调度程序实现的基本方法；能对核心源代码进行分析和修改；训练分析问题、解决问题以及自主学习能力，逐步达到能独立对小型操作系统的功能进行分析、设计和实现。

实验环境：EOS 操作系统及其实验环境。

实验内容：

跟踪 EOS 的线程调度程序，分析 EOS 基于优先级的抢占式调度的核心源代码，阐述其实现方法；修改 EOS 的调度程序，添加时间片轮转调度功能。

实验步骤：

1）EOS 基于优先级的抢占式调度工作过程的跟踪与源代码分析

（分析 EOS 基于优先级的抢占式调度的核心源代码，阐述其实现方法，包括相关数据结构和算法等；简要说明在本部分实验过程中完成的主要工作，包括对 EOS 调度程序的跟踪等）

1. EOS 抢占式调度代码分析

EOS 为线程划分了 32 个优先级，每个优先级对应一个就绪线程队列。此外还有一个线程指针指向当前正在运行的线程。为了方便进行线程调度，EOS 还定义了一个 32 位的就绪位图 PspReadyBitmap，若位图的第 n 位为 1，则表明优先级为 n 的就绪队列非空，这样就可以快速地判断某个优先级的就绪队列是否有线程，从而进行线程调度，具体的代码在 ps/sched.c 中定义如下：

```
// 32 个优先级的线程队列链表头
LIST_ENTRY PspReadyListHeads[32];
// 32 位就绪位图。
//  如果位图的第 n 位为 1，则表明优先级为 n 的就绪队列非空
volatile ULONG PspReadyBitmap = 0;
//  线程调用 Sleep 后，在这个队列中进行等待。
LIST_ENTRY PspSleepingListHead;
//  已结束线程队列。
LIST_ENTRY PspTerminatedListHead;
//  当前运行线程。
```

```
volatile PTHREAD PspCurrentThread = NULL;
```

EOS 的线程调度函数是 PspThreadSchedule，在 ps/sched.c 中定义。该函数确定了调度时机，其函数逻辑如下：

1. 若当前在处理中断，则结束，因为中断返回时系统会自动执行线程调度；

2. 若当前线程已经处在非运行状态，则执行线程调度；

3. 若 1 和 2 都不满足，则说明当前有线程正在运行，之后扫描位图，若存在比当前线程优先级高的就绪线程，则进行线程调度。

```
VOID
PspThreadSchedule(
    VOID
    )
{
    ULONG HighestPriority;
    // 注意，如果当前正在处理中断（中断嵌套深度不为 0）则什么也不做，
    if (KeGetIntNesting() == 0) {
        if (Running != PspCurrentThread->State) {
            // 当前线程已经处于非运行状态，执行线程调度。
            KeThreadSchedule();
        } else if (0 != PspReadyBitmap) {
            // 扫描就绪位图，如果存在比当前线程优先级高的就绪线程则执行线程调度。
            BitScanReverse(&HighestPriority, PspReadyBitmap);
            if (HighestPriority > PspCurrentThread->Priority)
                KeThreadSchedule();
        }
    }
}
```

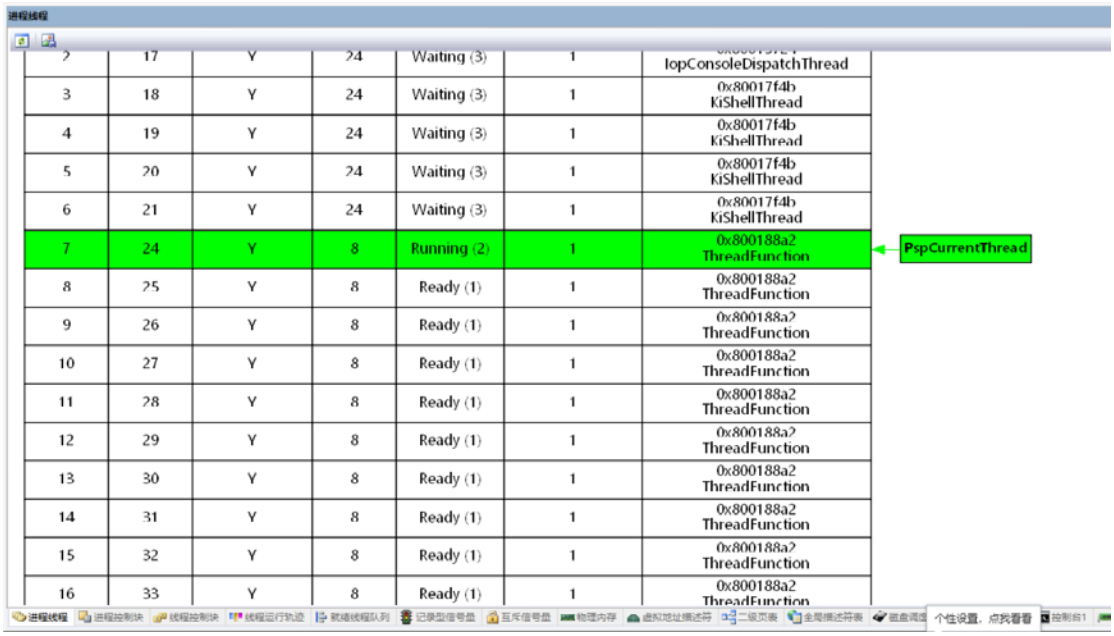
PspSelectNextThread 实现了基于优先级的抢占式调度，函数首先扫描就绪位图获取当前最高优先级，之后选择优先级最高的非空就绪队列的队首线程作为当前运行线程（若当前线程优先级最高，则不进行调度）。

```
PCONTEXT
PspSelectNextThread(
    VOID
    )
{
    ULONG HighestPriority;
    SIZE_T StackSize;
```

```
    //
    // 扫描就绪位图，获得当前最高优先级。注意：就绪位图可能为空。
    BitScanReverse(&HighestPriority, PspReadyBitmap);
    if (NULL != PspCurrentThread && Running == PspCurrentThread->State) {
        if (0 != PspReadyBitmap && HighestPriority > PspCurrentThread->Priority) {
            // 如果存在比当前运行线程优先级更高的就绪线程，当前线程应被抢先。
            // 因为当前线程仍处于运行状态，所以被高优先级线程抢先后应插入其
            // 优先级对应的就绪队列的队首。注意，不能调用 PspReadyThread。
            ListInsertHead( &PspReadyListHeads[PspCurrentThread->Priority],
                                    &PspCurrentThread->StateListEntry );
            BIT_SET(PspReadyBitmap, PspCurrentThread->Priority);
            PspCurrentThread->State = Ready;
        }
    ……
    // 选择优先级最高的非空就绪队列的队首线程作为当前运行线程。
    PspCurrentThread = CONTAINING_RECORD(PspReadyListHeads[HighestPriority].Next,
THREAD, StateListEntry);
    ObRefObject(PspCurrentThread);
    PspUnreadyThread(PspCurrentThread);
    PspCurrentThread->State = Running;
    // 换入线程绑定运行的地址空间。
    MmSwapProcessAddressSpace(PspCurrentThread->AttachedPas);
    // 返回线程的上下文环境块，恢复线程运行。
    return &PspCurrentThread->KernelContext;
}
```

2. EOS 线程调度程序 PspSelectNextThread 调试

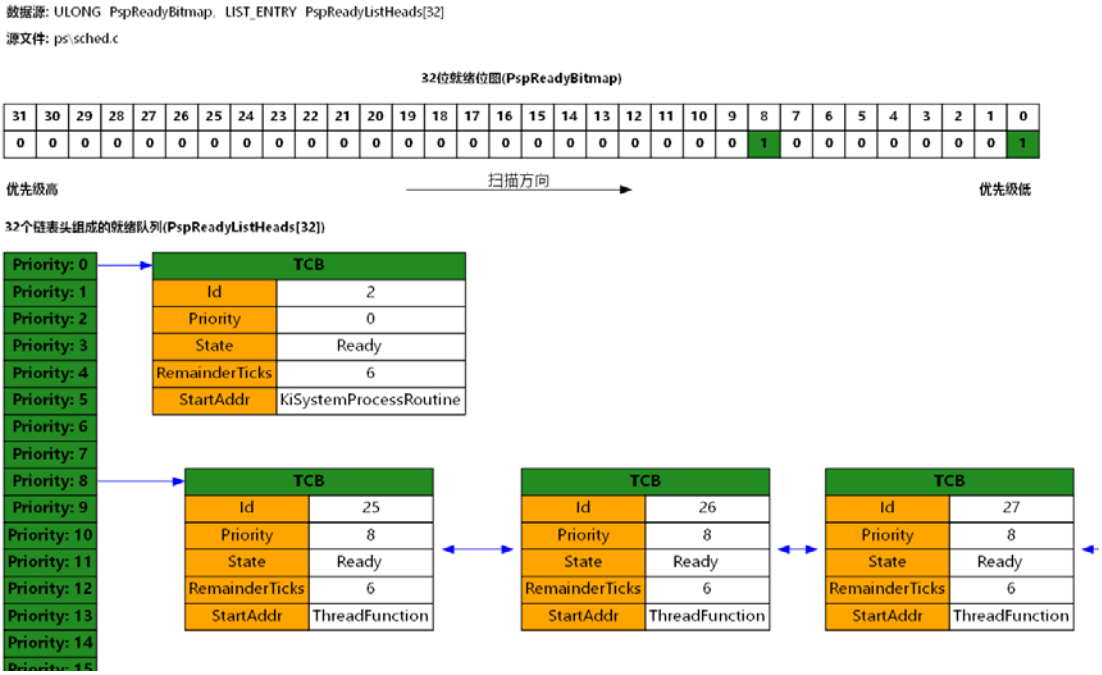
首先调试当前线程不被抢先的情况，在 ke/sysproc.c 的 679 行（ThreadFunction）添加断点，启动调试，在控制台中输入 rr 命令，这时程序会在断点处中断，可以在“进程线程窗口”中看到只有 rr 创建的 ID=24 的线程处在运行状态。



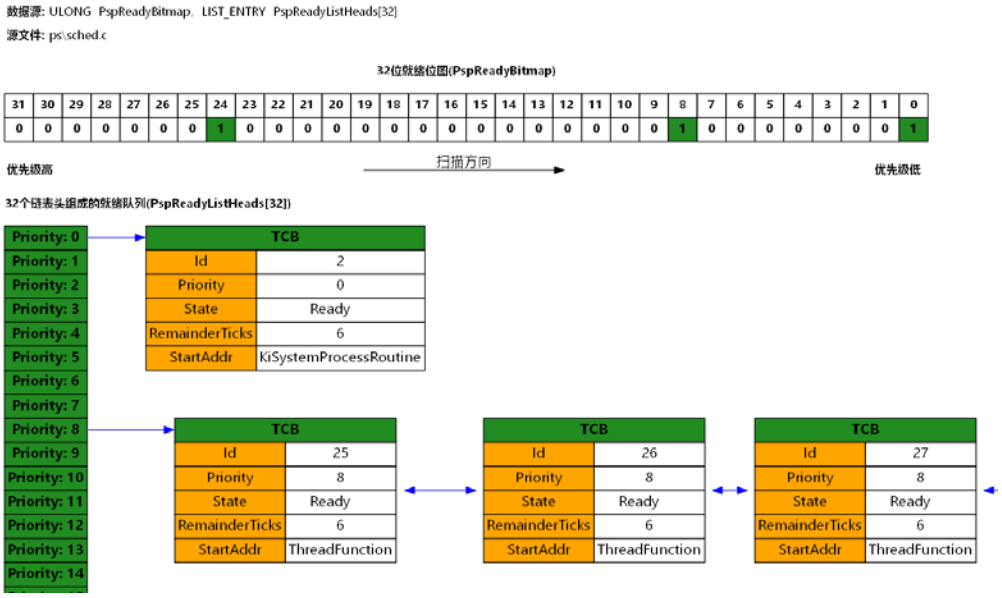
2	17	Y	24	Waiting (3)	1	IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
7	24	Y	8	Running (2)	1	0x800188a2 ThreadFunction
8	25	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
9	26	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
10	27	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
11	28	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
12	29	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
13	30	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
14	31	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
15	32	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
16	33	Y	8	Ready (1)	1	0x800188a2 ThreadFunction

PspCurrentThread

同样，还可以从“就绪线程队列”中看到 32 位就绪位图和就绪线程队列。我们可以看到，优先级为 8 的就绪线程队列队首的 ID 是 25 而非 24，这是由于 rr 创建的第一个 ID=24 的线程正在运行，不在就绪队列中。



接下来调试当前线程被抢先的情况。命令 rr 创建的线程优先级为 8，低于控制台派遣线程的优先级 24。在控制台按下空格键，就会让控制台派遣线程进入就绪状态，而后线程调度函数就会让控制台派遣线程抢占处理器。下图显示了控制台派遣线程（ID=24）就绪后的位图：



2）为 EOS 添加时间片轮转调度

（给出实现方法的简要描述、源代码、测试及结果等）

ke/sched.c 文件中定义了 PspRoundRobin 函数，该函数会被定时计数器中断服务程序 KiIsrTimer 每间隔一段时间调用，时间长度即为用户定义的时间片 TICKS_OF_TIME_SLICE。

每当运行到 PspRoundRobin，操作系统都应判断当前正在运行线程的剩余时间片数量，若剩余时间片大于零，则线程继续运行；否则进行线程调度，利用 BIT_TEST(PspReadyBitmap,PspCurrentThread->Priority)扫描位图中有无优先级更高的就绪线程，若有优先级更高的线程，则调用 PspReadyThread 进行线程调度。具体代码如下：

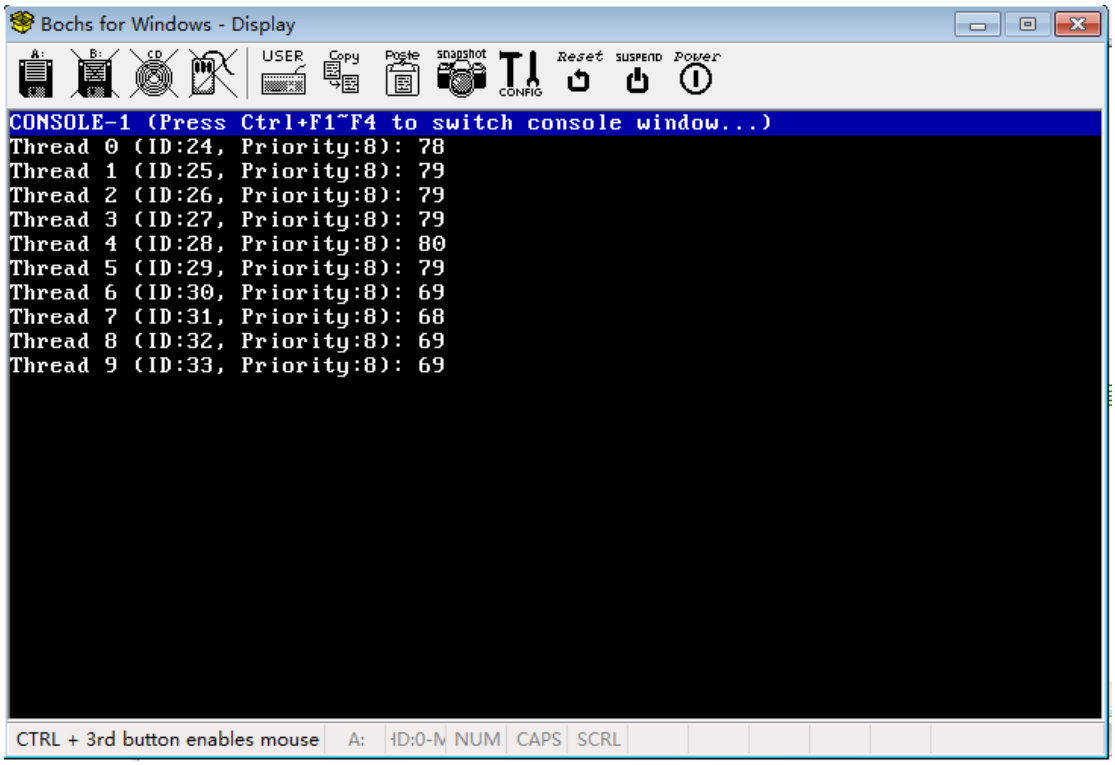
```
VOID
PspRoundRobin(
    VOID
    )
    // 时间片轮转调度函数，被定时计数器中断服务程序 KiIsrTimer 调用。
{
        if(NULL!=PspCurrentThread && Running == PspCurrentThread->State){
        // 剩余时间片大于 0
        if(PspCurrentThread->RemainderTicks>0){
            PspCurrentThread->RemainderTicks--; // 减少一个剩余时间片
        }

        // 剩余时间片=0，需要进程调度
        if(PspCurrentThread->RemainderTicks==0){
            PspCurrentThread->RemainderTicks = TICKS_OF_TIME_SLICE;
```

```
            // 判断位图之中，与被中断线程优先级相同的队列，有无就绪线程
            if(BIT_TEST(PspReadyBitmap,PspCurrentThread->Priority)){
                PspReadyThread(PspCurrentThread);
            }
        }
    }

    return;
}
```

下图是验证结果，可以从图中看到，Thread 0-9 不断交替运行，说明 RoundRobin 功能正确，Thread 0 在时间片用尽后被其它线程抢占。



结果分析：

EOS 本身只实现了基于优先级的抢占式调度，同一优先级只能有一个线程运行，后面的线程只能等待。在实现 RoundRobin 之后，同一优先级的线程可以交替执行。但这仍有不足，若高优先级的线程很多，不断交替执行，低优先级的线程就会陷入长时间的等待，甚至永远不会被执行。为了解决这个问题，还需要加入优先级提升的机制，其中一种实现方法就是为每个线程设定一个“等待时间”，单位是时间片，初始设定一个“最长等待时间”。每当 EOS 调用 KiIsrTimer 时，就扫描所有就绪队列，把所有就绪线程的等待时间“减一”，当一个线程的等待时间被减为 0 时，说明它已经等待了很长时间，这时就把这个线程的优先级加一，同时调度到更高一级优先级就绪队列的队尾，从而避免一个线程长时间等待。