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 2 线程状态及转换（4 分）

实验目的：以一个教学型操作系统 EOS 为例，熟悉线程状态及其转换，理解线程状态转换与线程调度的关系；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 线程在各种状态间的转换过程，分析 EOS 中线程状态及其转换的相关源代码，描述 EOS 定义的线程状态以及状态转换的实现方法；修改 EOS 的源代码，为线程增加挂起状态。

实验步骤：

1）EOS 线程状态转换过程的跟踪与源代码分析

（分析 EOS 中线程状态及其转换的核心源代码，总结 EOS 定义的线程状态以及状态转换的实现方法，包括数据结构和算法等；简要说明在本部分实验过程中完成的主要工作，包括对 EOS 线程状态转换过程的跟踪等）

一、源代码分析：

和进程相似，EOS 利用线程控制块 TCB 来管理线程，进程会通过链表来组织隶属于它的线程。TCB 中存储了描述线程的多种信息，包括所属的进程指针、线程链表项、优先级、当前状态等。需要说明的是 EOS 系统中的线程没有用户态和内核态之分，所有的线程一视同仁，共同竞争 CPU 的使用权。

EOS 的 TCB 在 ps/psp.h 的 68 行定义如下：

```
//
// 线程对象结构体 (TCB)。
//
typedef    struct _THREAD {
    PPROCESS Process;                    // 线程所属进程指针
    LIST_ENTRY ThreadListEntry;          // 进程的线程链表项
    UCHAR Priority;                      // 线程优先级
    UCHAR State;                         // 线程当前状态
    ULONG RemainderTicks;                // 剩余时间片，用于时间片轮转调度
```

```
    STATUS WaitStatus;                  // 阻塞等待的结果状态
    KTIMER WaitTimer;                   // 用于有限等待唤醒的计时器
    LIST_ENTRY StateListEntry;          // 所在状态队列的链表项
    LIST_ENTRY WaitListHead;            // 等待队列，所有等待线程结束的线程都在此队列等待。
    PVOID KernelStack;                  // 线程位于内核空间的栈
    CONTEXT KernelContext;              // 线程执行在内核状态的上下文环境状态

    //
    // 线程必须在所属进程的地址空间中执行用户代码，但可在任何进程的地址空间中执行
    // 内核代码，因为内核代码位于所有进程地址空间共享的系统地址空间中。
    //
    PMMPAS AttachedPas;                 // 线程在执行内核代码时绑定进程地址空间。

    PTHREAD_START_ROUTINE StartAddr;  // 线程的入口函数地址
    PVOID Parameter;                    // 传递给入口函数的参数

    ULONG LastError;                    // 线程最近一次的错误码
    ULONG ExitCode;                     // 线程的退出码
} THREAD;
```

EOS 定义了 5 种线程状态：0-Zero、1-Ready、2-Running、3-Waiting、4-Terminated，具体的代码在 ps/psp.h 的 103 行：

```
typedef enum _THREAD_STATE {
    Zero,           // 0
    Ready,          // 1
    Running, // 2
    Waiting,  // 3
    Terminated     // 4
} THREAD_STATE;
```

值得注意的是，这里定义的 Zero 状态原意是描述线程切换的中间态，在后面实现挂起状态（Suspend）时将其当做了挂起状态。

线程的操作：

1. 创建线程

EOS 创建线程最终都是通过 PspCreateThread 来完成的，其流程为：创建线程对象->为线程分配执行在内核态时所需的内核栈->初始化线程控制块->将线程插入所在进程的线程链表->使线程进入就绪状态->进行线程调度。

PspCreateThread 的具体定义如下（在 ps/create.c 第 520 行）：

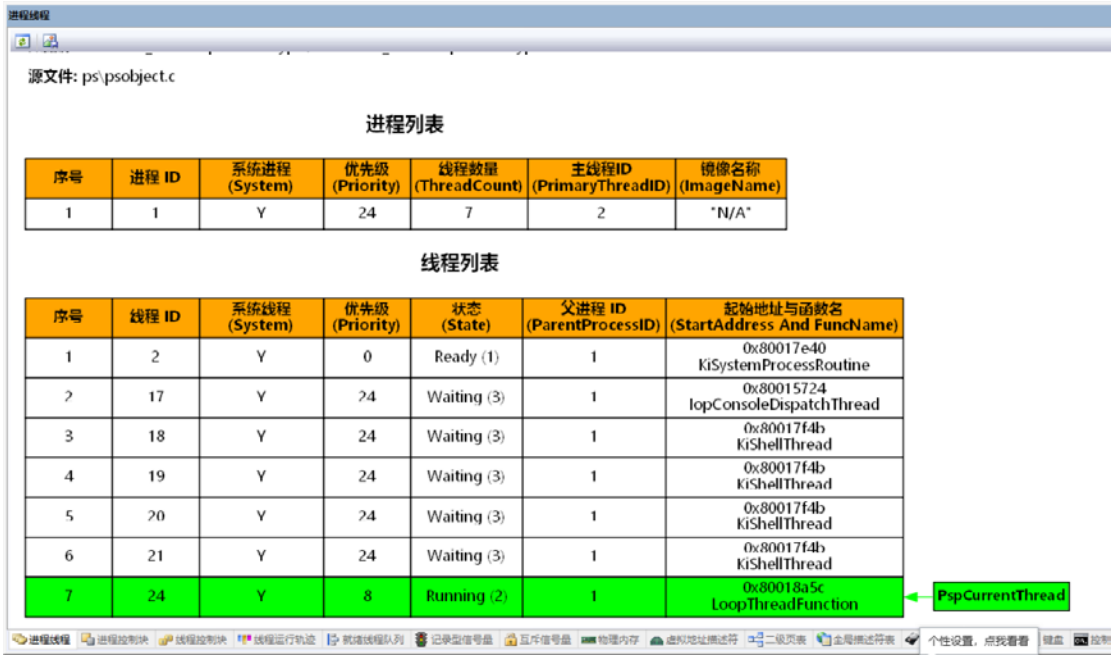
```
STATUS
PspCreateThread(
    IN PPROCESS Process, // 进程对象指针，新创建的线程将属于这个进程。
    IN SIZE_T StackSize,   // 用户模式线程栈大小
    IN PTHREAD_START_ROUTINE StartAddr,// 线程开始执行的函数的指针
    IN PVOID ThreadParam, // 传递给线程函数的参数
    IN ULONG CreateFlags, // 创建参数，目前尚无参数可选
    OUT PTHREAD *Thread     // 用于输出新线程对象的指针
    )
```

2. 线程调度

线程调度即根据线程优先级和调度算法来变换线程的状态，从而使各个线程都能得到有效执行，使效用最大。线程的每个状态都会有一个队列（或称为链表，因为 EOS 经常需要修改队列中间的元素），存储着处于该状态的所有线程的指针，当对线程进行调度时，主要的思想就是把线程指针从当前状态队列中移除，再移动至目标状态的队尾，接着通过线程调度算法选择一个线程执行。EOS 中的线程调度主要通过调用 ps/sched.c 文件中的 PspReadyThread 、 PspUnreadyThread 、 PspWait 、 PspUnwaitThread 、 PspWakeThread 、 PspSelectNextThread 来实现，这些函数的功能可以顾名思义。

二、线程状态切换的跟踪

在 loop（ke/sysproc.c 的 786 行）添加断点，启动调试，控制观察进程线程窗口如下：



观察到此时只有 LoopThreadFunction 在运行。

接下来分别在 ps/sched.c 中的第 129,161,226,289,402 行添加断点，继续按 F5 运行，按下空格，不断按 F5 可以观察到程序在上述断点处中断，总结系统响应键盘事件的过程如下：

1. 按下空格
2. 控制台线程从等待队列中取出
3. 控制台线程进入就绪队列队尾（修改位图，修改线程状态为 ready）
4. 控制台线程优先级高于当前正在运行的 loop，控制台线程进入就绪队列队首
5. 处理完键盘事件，EOS 将控制台线程从所在的就绪队列中取出
6. 控制台线程进入等待队列队尾
7. 取出就绪队列队首（loop）运行。

2）为线程增加挂起状态

（给出实现方法的简要描述、源代码、测试及结果等）

1. 实现方式：

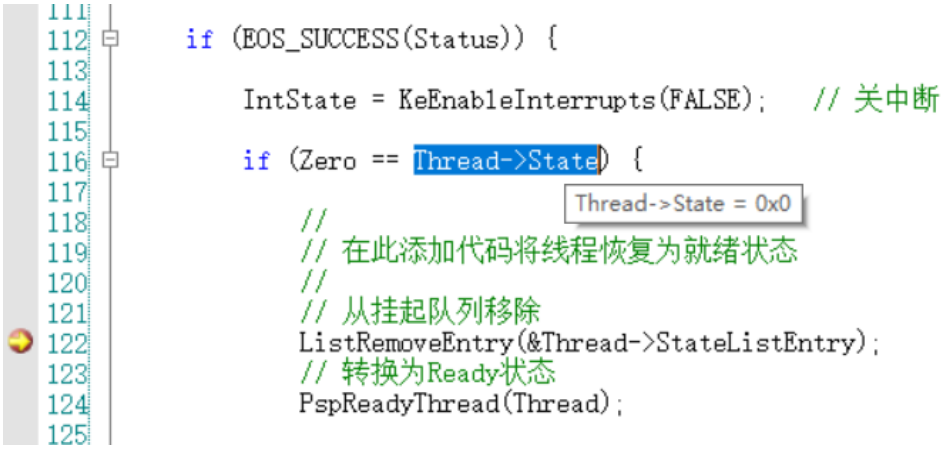
EOS 已经实现了 Suspend 操作，将线程的 Zero 状态当做为挂起。但是 EOS 没有实现 Resume 操作，即挂起->就绪的转换。从挂起状态到就绪状态一共需要两步操作：首先将目标线程从挂起队列中移除，之后将线程切换为就绪状态。

在 ps/psspnd.c 的 120 行添加下面两行代码：

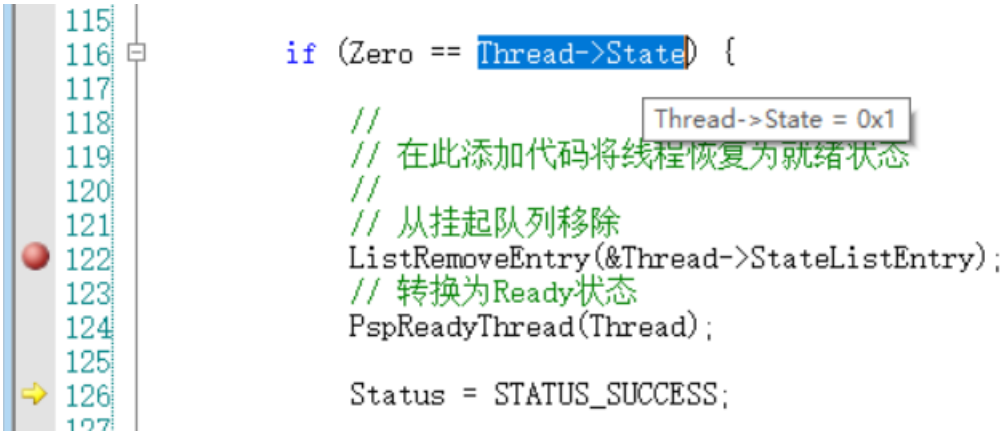
```
// 线程从挂起队列移除
ListRemoveEntry(&Thread->StateListEntry);
// 将线程转换为 Ready 状态，并加入就绪队列队尾
PspReadyThread(Thread);
```

2. 验证

在上面添加的 ListRemoveEntry(&Thread->StateListEntry);一行加断点，启动调试，在控制台中输入 loop，启动 loop 线程，之后键入 suspend 24，将线程挂起；然后键入 resume 24 进行我们实现的 Resume 操作，这时程序会在我们所加的断点处中断，将鼠标移动到 Thread->Status 查看线程状态，如下图所示：



可以看到，Thread->State 的值为 0，处在挂起状态。接着按两下 F10 单步调试，运行完 PspReadyThread(Thread)之后，再用相同的方法查看 Thread->State 的值：



可以看到，Thread->State 的值变为了 1，线程已经进入了就绪状态，说明我实现的 Resume 操作功能正确。在这之后，也可以切换回控制台，观察到 loop 线程继续运行，屏幕上的数字不断增加。

结果分析：

（对本实验所做工作及结果进行分析，包括 EOS 线程状态及其转换方法的特点、不足及改进意见；结合 EOS 对线程状态及其转换的相关问题提出自己的思考，分析线程状态转换与线程调度的关系；分析为线程增加挂起状态实现方法的有效性、不足和改进意见，如果同时采用了多种实现方法，则进行对比分析；其他需要说明的问题）

EOS 的为线程定义了 4 种状态：0-中间态，1-就绪，2-运行，3-等待，4-结束。每个状态都会对应一个线程队列，同时还有一个位图用来方便查询各个线程的状态。下面我总结了 EOS 进行线程状态转换的过程：

1. 将线程从当前队列中移除

2. 将线程移动到目标状态的队尾

3. 修改线程状态

4. 进行线程调度

线程状态是线程调度的依据，线程调度算法会选择当前就绪队列的队首线程来运行，而处在其它队列的线程，可能会在等待某项资源的就绪，比如 IO，这时如果让线程继续占据 CPU，它无法进行任何操作，就造成了 CPU 时间的浪费。线程状态和线程调度的最终目标都是提高 CPU 利用率。

在实现 Resume 操作时，一共需要两步操作：首先将目标线程从挂起队列中移除，之后将线程切换入就绪状态，接下来进行线程调度，就可以保证当前优先级最高的线程占用 CPU。