第 1 章　进程线程

一、源码阅读

阅读要求：

1. 基本头文件：types.h param.h memlayout.h defs.h x86.h asm.h mmu.h elf.h
2. 进程线程部分：vm.c proc.h proc.c swtch.S kalloc.c 以及相关其他文件代码

基本头文件阅读：

这里主要介绍一下基本头文件的功能：

type.h 文件，通过 typedef 定义了数据类型的别名；

param.h 文件，通过#define 定义了特殊参数的具体值，如系统最大进程数为 64、系统最大打开文件数为 100、每个进程的打开文件上限为 16 个等；

memlayout.h 文件，通过#define 定义了内存的布局；

defs.h 文件，不仅声明了其他文件所需的数据结构，如 file（文件）、inode（i 节点）、context（上下文场景），还声明了一些基础的系统方法，如 bread 和 bweite 字节读写操作、系统调用、console 调试等。

x86.h 文件，其主要功能为让 C 代码使用特殊的 x86 指令。

asm.h 文件，通过汇编宏创建 x86 扇区，并定义扇区容量为 4Kb。同时，也定义了可执行扇区、可读扇区以及可写入扇区。

mmu.h 文件，这个文件包含 x86 内存管理单元(MMU)的定义。

elf.h 文件，定义了 elf header，即 elf 可执行文件头。同时，也定义进程空间的头部结构。

以上，就是这些基础头文件的主要功能，以及涉及的方面。它们是 XV6 操作系统实现的底层基础。

基本头文件阅读：

进程是操作系统资源分配的最小单位，线程是操作系统中调度的最小单位。进程是指一个正在运行的程序的实例，而线程是一个 CPU 指令执行流的最小单位。

XV6 系统中只实现了进程，并没有提供对线程的额外支持，一个用户进程永远只会有一个用户可见的执行流。对于每一个进程都存在对应的唯一的进程控制块（PCB），在 XV6 操作系统中，其定义在 proc.h 文件中。如下图所示，PCB 中包含了进程的相关信息，如进程 ID、

占用内存大小、内核栈底指针、进程状态等

```
struct proc {
  uint sz;                     // 占用内存大小
  pde_t* pgdir;                // 所属页表
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // 进程状态
  int pid;                     // 进程 ID
  struct proc *parent;         // 父进程
  struct trapframe *tf;        // 当前系统调用陷入的存储
  struct context *context;     // 上下文信息
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // 若非0，则说明进程已被kill
  struct file *ofile[NOFILE];  // 打开文件
  struct inode *cwd;           // 当前目录-inode
  char name[16];               // 进程名 (debugging)
};
```

此外，还包括进程所调用的文件对应打开文件和所属页表。同时，当进程进行切换时，上下文信息被存储至 context 中。

XV6 操作系统还定义了 ptable 数据结构，用于存储和管理所有进程。其定义于 proc.c 文件中，除了互斥锁 lock 之外， XV6 系统中允许同时存在的进程数量是有上限的。NPROC 为 64，即 XV6 最多只允许同时存在 64 个进程。该参数定义在 param.h 中，可修改。

同时，在 proc.c 文件中，还定义进程的基本操作：

- Fork 操作，用于创建新的进程；
- Exit 操作，用于退出进程，但该进程并未销毁，直到其父进程通过 wait()检测到该进程已退出；
- Wait 操作，用于等待子进程退出，并返回其进程 ID。若返回-1，则说明该进程无子进程；
- Yield 操作，用于使当前进程放弃 cpu，开始下一轮调度；
- ForkRet 操作，用于当一个新创建的进程第一次被调度时，释放 Ptable 的锁，并初始化用户进程空间；
- Sleep 操作，用于当前进程转化为 SLEEPING 状态，并调用 sched 以释放 CPU；
- Wakeup 操作，用于寻找一个睡眠状态的进程并把它标记为 RUNNABLE；
- Kill 操作，用于杀死进程。

二、讨论总结

什么是进程，什么是线程？操作系统的资源分配单位和调度单位分别是什么？？XV6 中的

进程和线程分别是什么，都实现了吗？

从抽象的意义来说，进程是指一个正在运行的程序的实例，而线程是一个 CPU 指令执行流的最小单位。进程是操作系统资源分配的最小单位，线程是操作系统中调度的最小单位。

从实现的角度上讲，XV6 系统中只实现了进程， 并没有提供对线程的额外支持，一个用户进程永远只会有一个用户可见的执行流。

进程管理的数据结构是什么？在 Windows，Linux，XV6 中分别叫什么名字？其中包含哪些内容？操作系统是如何进行管理进程管理数据结构的？它们是如何初始化的？

进程管理的数据结构被叫做进程控制块(Process Control Block, PCB)。一个进程的 PCB 必须存储以下两类信息：

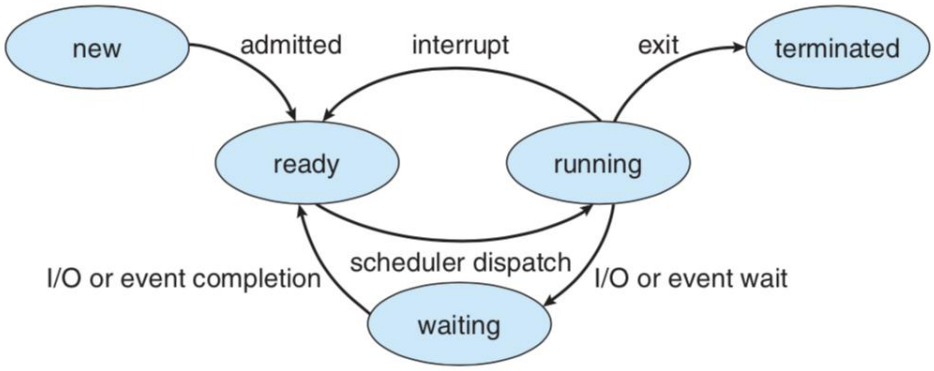
1. 操作系统管理运行的进程所需要信息，比如优先级、进程 ID、进程上下文等
2. 一个应用程序运行所需要的全部环境，比如虚拟内存的信息、打开的文件等。

在 Windows NT 以后的 Windows 系统中，进程用 EPROCESS 对象表示，线程用 ETHREAD 对象表示。Linux 系统的实现中并不刻意区分进程和线程，而是将其一概存储在被称作 task_struct 的数据结构中。在 XV6 操作系统中，与进程有关信息定义在 proc 的数据结构中。

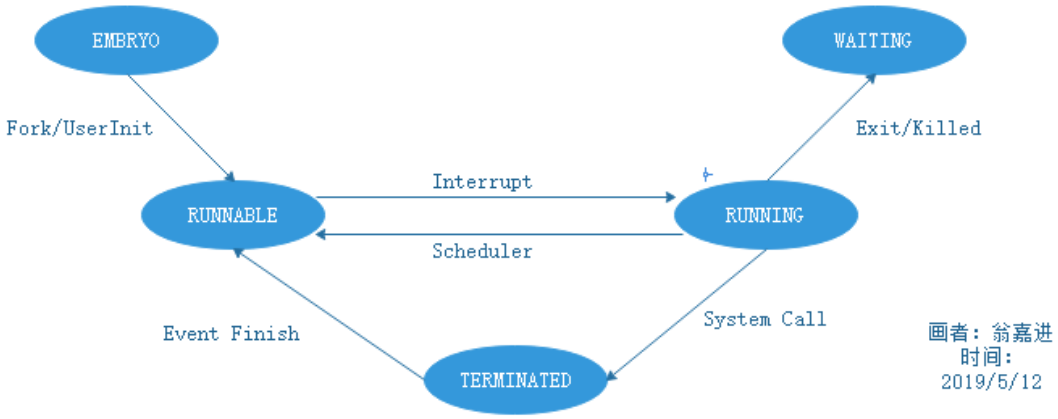
一般操作系统会有统一的 PCB 管理模块，在 XV6 中使用的是 PTable，具体可参考上一节源码阅读部分。

进程有哪些状态？请画出 XV6 的进程状态转化图。在 Linux，XV6 中，进程的状态分别包括哪些

在大多数教科书使用的标准五状态进程模型中，进程分为 New、Ready、Running、Waiting 和 Terminated 五个状态，状态转换图如图所示



XV6 操作系统的进程状态采用的是五状态进程模型。在 XV6 中这五个状态的定义为 EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE。其状态转换图如图所示：



Linux 中的进程转换图与 Xv6 大致相同，但是有如下区别：

1. Linux 中的 Waiting 有两个状态，分别是 Interruptible Waiting 和 Uninterruptible Waiting。
2. Linux 中额外有两个调试用状态 TASK_TRACED 和 TASK_STOPPED。

如何启动多进程（创建子进程）？如何调度多进程？调度算法有哪些？

通过 Fork 操作即可创建子进程。在 XV6 操作系统中，有专门负责进程调度工作的调度器，其定义于 proc.c 文件中。

经典的批处理系统调度算法包括：（1）FCFS-先来先服务；（2）SJF-最短作业优先；（3）SRT-最短剩余时间优先；（4）HRRN-最高相应比优先。这其中，FCFS 相对效率较低，而 SJF 效率最高，但可能出现进程“饥饿”的现象；相比之下，HRRN 则是折衷考虑了等待时间与响应时间。

现今被广发使用的操作系统有 Windows、Linux、UNIX、UNIX(5.3 BSD)，他们的调度算法分别为基于优先级的抢占式多进程调度、抢占式调度、动态优先数法、多级反馈队列法。

操作系统为何要限制一个 CPU 最大支持的进程数？XV6 中的最大进程数是多少？如何执行进程的切换？ 什么是进程上下文？多进程和多 CPU 有什么关系？

因为计算机中 CPU 支持的进程是要受内存大小限制的，每一个进程都拥有独自的进程空间，因而进程的数量一旦超过一定数目，会导致内存溢出等现象。

XV6 中最大进程数为 64 个，它被宏定义与 param.h 中，可修改。

switch.s 文件中定义了线程交换的汇编语言实现，多进程操作系统需要通过时间片分配算法来循环执行进程，当前进程执行一个时间片后会切换到下一个进程。但是，在切换前会保存上一个进程的状态，以便下次切换回这个进程时，可以再次加载这个进程的状态，进程的状态（如寄存器状态，代码执行的位置）即进程上下文。

单 CPU 计算机中的多进程只能是并发，多 CPU 计算机中的多进程可以并行。无论是并发还是并行，使用者来看，看到的都是多进程。

内核态进程是什么？用户态进程是什么？它们有什么区别？

内核态进程，顾名思义，是在操作系统内核态下执行的进程。在内核态下运行的进程一般用于完成操作系统最底层，最为核心，无法在用户态下完成的功能。比如，调度器进程是 Xv6 中的一个内核态进程，因为在用户态下是无法进行进程调度的。

相比较而言，用户态进程用于完成的功能可以多种多样，并且其功能只依赖于操作系统提供的系统调用，不需要深入操作内核的数据结构。比如 init 进程和 shell 进程就是 xv6 中的用户态进程。

进程在内存中是如何布局的，进程的堆和栈有什么区别？

Xv6 进程在虚拟内存中的布局如下。进程的栈用于存放运行时数据和运行时轨迹，包含了函数调用的嵌套，函数调用的参数和临时变量的存储等。栈通常较小，不会在运行时增长，不适合存储大量数据。相比较而言，堆提供了一个存放全局变量和动态增长的数据的机制。堆的大小通常可以动态增长，并且一般用于存储较大的数据和程序执行过程中始终会被访问的全局变量。