XV6 源码阅读报告

一、系统中进程的描述和重要变量

1. 进程描述

XV6中进程的描述主要通过结构体proc，这个结构体中定义了描述线程的主要变量。如下段代码所示。

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  volatile int pid;            // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

逐个来分析每个变量，观察XV6如何描述进程。

- uint型变量sz表示的进程空间大小，记录该进程所占有的空间大小。
- pgdir变量存储的是进程所占有的页目录，因为进程的空间是按页分配的，故而存储一个指向页目录的指针就可以得到整个进程地址空间的入口。
- kstack指针指向的是进程内核栈的栈底
- state标识进程的状态。该变量是个枚举类型，枚举类型值的集合，该枚举变量的定义如下。可见进程的状态貌似有6种。但是由于UNUSED严格上来说不应该算作是进程的状态，它是描述系统进程表空闲的状态，故而，严格上来说，进程一般在五个状态之间切换。

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

- pid是进程标识符，该标识符用于区分不同的进程。
- parent变量主要指向进程的父进程。由于进程的创建一般需要在另一个进程中调用fork()函数，一般的进程都会有父进程。
- tf变量也是一个指针变量，它主要指向的是该进程当前系统调用的中断帧。其中中断帧保存着一些寄存器的值。中断帧是内核栈的一帧。在系统从用户态跳转到内核态时，中断帧保存了跳转之前的

系统寄存器信息，这样跳转回用户态的时候，系统可以恢复进入内核态之前的状态。

- context指针记录了进程的上下文信息，所谓的上下文信息，查看其定义发现，也是一些寄存器的值。当进程需要运行的时候，就把其上下文信息换到CPU上。

```
struct context {
  uint edi;//目的变址寄存器
  uint esi;//源变址寄存器
  uint ebx;//基地址寄存器
  uint ebp;//当前时刻系统栈顶指针
  uint eip;//cpu指令地址寄存器
};
```

- chan是进程sleep()的一个标志量。之后分析进程sleep状态会详细描述该变量的作用。
- killed进程被kill的标志。进程调用kill()时，并不会马上被干掉，而是先被置成killed状态。处于这个状态的进程将会在其他时间被kill掉。
- ofile是打开文件指针数组，表示打开文件表。打开文件表的每一个指针都指向该进程打开的文件。
- cwd表示进程的当前的工作路径。
- name进程的名称。
  XV6用来表示进程的PCB，就是上述的pro结构体。

2.处理机描述

和pro结构体定义在同一个文件中的还有一个重要的数据结构cpu。XV6是一个支持多处理机的操作系统，它使用cpu这一结构体来描述处理机的状态，通过一个cpu指针列表，维护系统全局的处理机。

```
extern struct cpu cpus[NCPU];
extern int ncpu;
struct cpu {
  uchar id;                    // Local APIC ID; index into cpus[] below
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
  int ncli;                    // Depth of pushcli nesting.
  int intena;                  // Were interrupts enabled before pushcli?

  // Cpu-local storage variables; see below
  struct cpu *cpu;
  struct proc *proc;           // The currently-running process.
};
```

分析两处和进程线程相关的变量。

- id表示处理机id，它的值就是该处理机处在cpu描述表中的下标。

- scheduler指针指向的是当前处理机所处的上下文信息。用于进程切换，可能记录了被切换下进程切换时的状态，下次再次被调上CPU的时候，可以恢复运行状态。

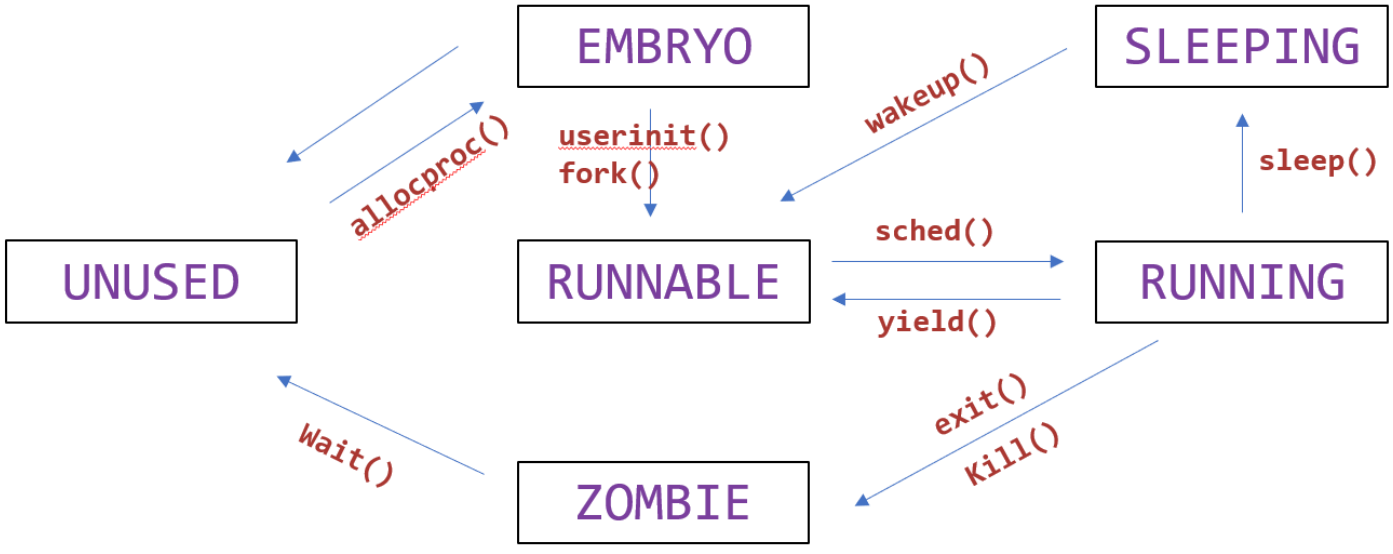
3.进程的全局管理

```
#define NPROC        64  // maximum number of processes
struct {
  struct spinlock lock;//锁
  struct proc proc[NPROC];//进程表
} ptable;
```

XV6将系统中所有进程都管理在进程表中。这个进程表主要依靠结构体ptable来维护。在ptable结构体中，成员变量分别为一个自旋锁变量和一个进程数组变量。由于这个进程数组在整个系统中只存在一份，并且，会有不同的进程调用该资源，所以，在修改进程表的时候需要加互斥锁，用来维护整个进程表的安全。可以看出的是，XV6系统中要求系统的最大进程数为64。每次进程分配，消亡的时候都需要把进程从表中添加和移除。

二、进程状态转换

宏观上来说，XV6的进程状态分别为UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE。其中UNUSED状态主要表示ptable中存储该表项指针的状态。当它指向UNUSED状态的时候，表示该表项可以被占用，可以被分配给新的进程。其他的状态转换如图所示。



分析其中具体关于状态转换的方法。

1.创建

- allocpro()方法
  allocpro方法主要的操作是创建进程，它的执行步骤主要是先查进程的全局管理表，通过这个表查看当前是否有标志位为UNUSED的表项用于分配新的进程。当全局表有空闲空间的时候，它会为该表项分配空间，并且填充它系统栈的内容，将进程状态设置为EMBRYO态。可以看见的新进程的默认函数是forkret()。该函数第一次调用时会初始化log系统，其他时候调用时，不做其他操作。该函数做了进程创建的具体工作，供其他方法调用。
- fork()方法
  fork()方法的主要操作也是创建进程，它调用allocpro()方法，allocpro()方法创建了一个基本的进程，做了进程创建过程的基本工作。fork()更进一步，它将会复制当前进程的地址空间中断帧，打开文件表，进程的工作空间。通过这种方式，进程复制为两个完全相同的进程，连名字也相同，唯一区分两种进程的方法就是根据进程id。分配结束后，fork()方法会将进程的状态设置程RUNNABLE，这样进程就会由EMBRYO状态转化程RUNNABLE。（当页表分配失败时，整个进程会被丢弃，设置成UNUSED）
- yield()方法
  该方法的主要作用是让权，即当前正在占用cpu的进程让出处理机资源。 该方法把正在执行的进程状态设置为RUNNABLE。并且把处理机让给进程调度器进程。

2.Sleep与wakeup

- sleep()和wakeup()方法
  sleep()方法的主要是让进程睡眠。设置一个标志量，通过改变进程pro中的这个标志量，让某进程睡眠在其上，当标志量改变时，所有等在这个标志量上的进程都会被唤醒。sleep()将进程从RUNING状态转换为SLEEPING状态，而wakeup()会将SLEEPING状态的进程转换为RUNNABLE。

3.消亡

- exit()和wait()方法
  exit()方法的主要操作是退出进程，具体来说它会将这个进程设置为ZOMBIE状态，将该进程的子进程的父进程设置为初始进程，移交控制权。而wait()方法是一个死循环，它会不断检测ptable中进程的状态，遇到僵尸进程时，它会把该进程清理，完成善后工作。
- kill()方法
  kill方法将给定id的进程杀死，主要通过设置指定进程中killed的标志量来标志该进程是否被kill。如果进程正在休眠，它会将该进程唤醒。

三、进程调度方式

XV6进程调度的主要方法依靠两个函数sched()和scheduler()。这两个函数都执行了一次进程切换。但是具体的进程如何切换，这两个函数有所区别。

- switch.s
  该文件为汇编文件，这个文件定义了上下文切换的具体步骤。主要是讲上下文相关的寄存器从cpu上换下来压栈，再将新进程的上下文换上cpu执行。
- sched()和scheduler方法的主要功能
  sched和scheduler()方法主要实现了进程之间的调度。其中sched()函数主要做的工作是将当前cpu执行的进程上下文换下cpu，将scheduler也就是调度器的上下文切换上cpu，scheduler方法是一个无限循环，它每次循环都将从patable中选出一个非空进程，换上cpu，直到该进程执行完毕。这就是一个按照patable下标优先的调度算法。表面上来看，进程先进入，会优先执行，然而实际上，可能后来的进程会被分配到下标靠前的位置。