XV6 进程进程机制

by 1801210840 姜慧强

XV6 中有关进程的结构体

XV6 中对进程进程调度相关变量在 proc.h 中声明。

使用了 struct cpu 描述 CPU 现有情况，利用 struct context 描述上下文信息， 其中 context 中存储了 5 种通用寄存器值包括栈底指针，下一指令指针，等等。

利用 procstate 规定六种进程状态（未被使用，分配，睡眠，正在运行，可执行，销毁）。

而进程 struct 定义在 struct proc 中，其中使用 pid 作为进程唯一标志号。

除此之外，还有进程占用内存大小，pageTable 路径，栈指针，进程状态，父进程 Struct，系统调用陷入结构，上下文指针，睡眠状态，销毁状态，打开文件的文件描述符列表，I-Node 进程结构，进程名称（最长 16 个字符）。

```
// Per-process state
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)
};
```

XV6 中进程相关实现函数

进程相关 struct 具体是在 proc.c 文件中实现的。

首先维护一个全局的进程表 ptable，ptable 是一个同步表，通过在 spinlock.c 中实现的自旋锁 spinLock 来实现互斥访问及信息同步。

然后还定义了初始主线程，当前分配 PID 号，fork,trap 函数。

XV6 中进程分配 allocproc()

进程分配是在 allocproc 函数中实现的，首先会去获得全局的 ptable 中的自旋锁。然后去遍历 ptable 中所有 proc 找到没有被用的进程 pid。

如果没有找到，则释放锁返回 0.

如果找到空位，则修改相应的 proc 状态，分配 pid 号。

然后分配内核栈，如果可用空间为 0，则释放自旋锁，返回 0。

然后相应的依次分配陷入结构大小，上下文信息区域大小。

最后返回相应的 proc struct。

XV6 中进程初始化 userint()

当 XV6 初始化运行的时候，先去调用 userint()

在 userint()中，先去初始化一个主进程。当主进程分配完毕的时候，会去检查 PageTable 路径地址。

然后分配 trap 结构中一些通用寄存器值。然后初始化进程名称，初始化 I 进程号，初始化状态。

XV6 中内存资源扩充 growproc()

在前面的定义中进程结构体需要事先分配相应的内存大小。如果在使用过程中，内存大小超过分配的内存大小，则需要额外申请空间。

在 XV6 中使用 growproc()实现。

具体实现中去 proc 的 pageTable 路径中查 PageTable，看有没有大小为 n 的空间可用。

该函数支持增量分配和减量分配两种模式。分别调用 vm.c 中 allocuvm 和 deallocuvm 两个函数进行操作。

首先查询所需要的内存大小是否会超过内核初始虚拟地址。

然后从最后一个已经分配的 Page 开始, 每次互斥的分配一个 4096 大小的物理内存（这一部分是在 kalloc.c 中实现的）。需要注意的是这一部分也是通过自旋锁来实现互斥访的。

然后调用 mappages 来分配虚拟地址。将虚拟地址映射到物理地址中。

然后交换全局描述表，把之前扩充的信息写入全局 cpu 中，并给予当前 proc 一个新的地址空间。

XV6 中的进程 Fork

首先分配一个初始化的 proc struct。

然后把原进程的 pgdir 复制给新进程，复制相应的 memory size 赋值给新的 proc。

将新进程的 parent 指向旧进程。赋值相同的 trap frame 值。赋值相同的打开文件列表。相同的 i 进程。 并赋予相同的 name。

并返回子进程的 pid 号。

XV6 中进程退出 Exit()

如果是初始进程，则抛异常。

接下来关闭所有打开的文件。

drop 内存中的 i 进程（该过程也是互斥的，通过 icache 的自旋锁来实现这个过程，确保过程的一致性）

然后获得 ptable 的锁，唤醒父进程。把所有该进程的子进程的父进程都改成初始进程。

如果子进程状态是 ZOMBIE 则唤醒该子进程。

更改进程状态，并调度下一个进程上 cpu。

XV6 中进程等待 Wait()

进程等待这个函数仅用于父进程等待子进程退出这个过程。

父进程完成任务但是因为子进程的原因不能直接退出需要等待子进程完成。

在该函数中，遍历 ptable，查看有没有相应需要等待的子进程，如果没有就直接退出。

否则则 sleep.

而 sleep 是通过在自旋锁上封装了一层睡眠锁来实现的。

QA

什么是进程，什么是进程？操作系统的资源分配单位和调度单位分别是什么

进程是操作系统中程序的一次执行过程，有独立的数据，代码段，有独立的堆栈，共享空间。

而线程则是进程的一个实体，是进程的一个执行分支。

线程从属于进程，线程所拥有的资源继承于进程，线程没有独立的资源。

线程是操作系统的基本调度单位，而进程则是操作系统中基本的资源分配单位。

XV6 中仅实现了进程也就是 proc。因为 proc 是基于 cpu 层面的，拥有独立的资源，有父进程，子进程关系的单元。

在 proc 下面没有更细粒度的线程。

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

XV6 中管理进程是通过一个带自旋锁的全局变量 ptable 来实现的。

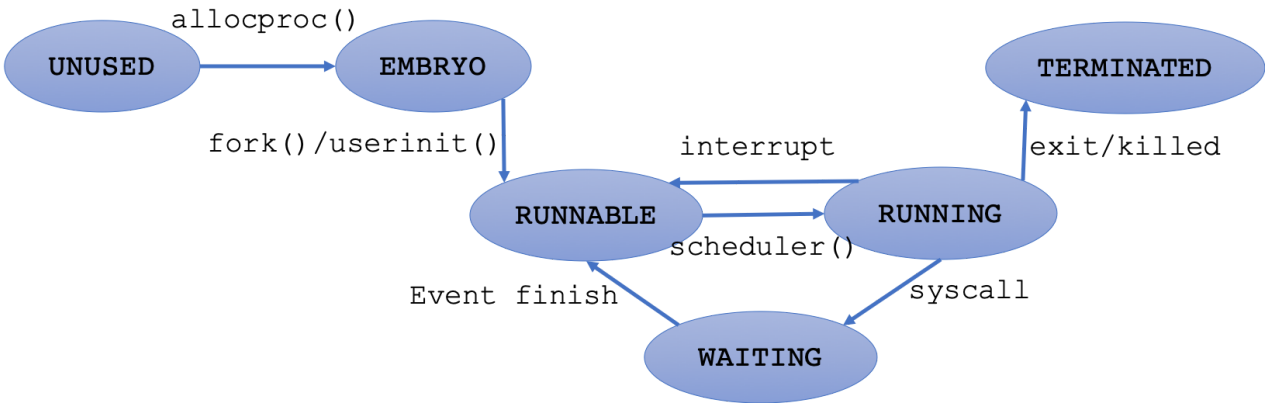
所有进程新建，进程调度，进程分配都需要通过互斥访问 ptable 来实现。

在 window 中，进程用 EPROCESS 对象表示，线程用 ETHREAD 对象表示。在一个 EPROCESS 对象中，包含了进程的资源相关信息，比如句柄表、虚拟内存、安全、调试、异常、创建信息、I/O 转移统计以及进程计时等。每个 EPROCESS 对象都包含一个指向 ETHREAD 结构体的链表。

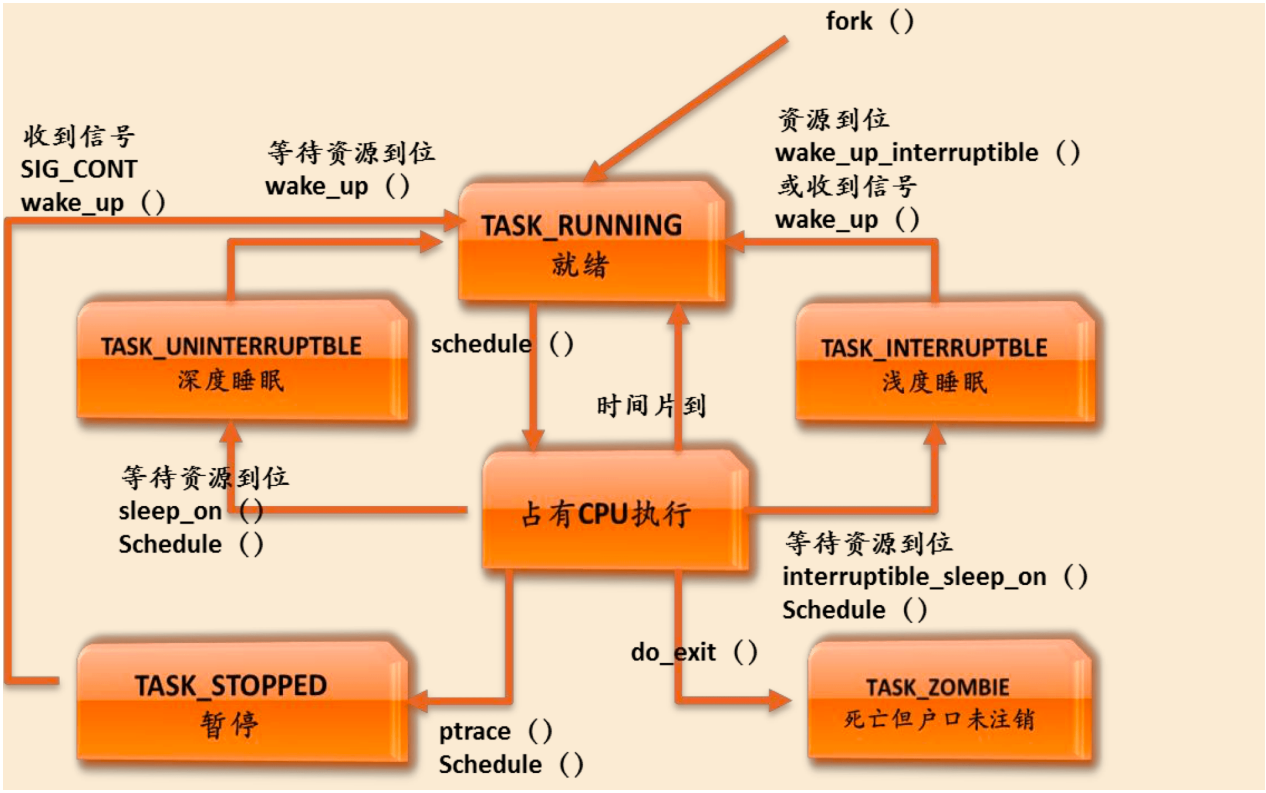
在 linux 中通过维护 task_struct 队列来实现，而 task_struct 队列是双向链表

进程有哪些状态？请画出 XV6 的进程状态转化图。在 Windows，Linux，XV6 中，进程的状态分别包括哪些？你认为操作系统的设计者为什么会有这样的设计思路？

XV6 中进程有五个状态，`enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };`



基本与五状态模型相一致。



对比 Linux, 可以发现其有两个 sleep 状态，一个是 task_uninterruptble 状态，一个是 task_interruptble 状态。两个状态区别于释放资源多少。

操作系统设计进程状态及其调度，个人认为主要是出于对有限资源 CPU 资源的高度利用而进行的规划过程。

如何启动多进程（创建子进程）？最大支持的进程数？如何调度多进程？XV6 中的最大进程数是多少？调度算法有哪些？操作系统为何要限制一个 CPU？XV6 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？

在 XV6 中通过父进程 fork()来完成子进程的启动。

首先分配一个初始进程结构 initproc.

然后把原进程的 pgdir 复制给新进程，复制相应的 memory size 赋值给新的 proc。

将新进程的 parent 指向旧进程。赋值相同的 trap frame 值。赋值相同的打开文件列表。相同的 i 进程。 并赋予相同的 name。

对父进程返回子进程的 pid 号，子进程返回 0.

XV6 中最大进程数为 64.

进程的上下文信息包括通用寄存器值，堆栈信息。

XV6 通过基于自旋锁实现的睡眠锁来实现进程的 sleep 从而换下 cpu，以便有足够的资源用于进程调度.

多进程在单 CPU 模式下也可以实现，多 CPU 中多进程需要更多的注意加锁访问，以防有不同的 CPU 同时获得资源。

内核态进程是什么？用户态进程是什么？它们有什么区别？

内核态是操作系统为了保护自身程序在运行时的转态，设定的具有较高权限的程序运行状态。

当用户程序需要调用具有较高优先级的程序时会去调用相应的内核态指令，通过访管指令来实现系统调用。

用户态一般指的是更接近用户端程序的一些进程，不需要更多的系统权限来完成其所需要实现的功能。

进程在内存中是如何布局的，进程的堆和栈有什么区别?

进程在内存中存放在 ptable 中。ptable 除了保持相应的进程信息列表之外，还保存着一个全局自旋锁，以便在多核心操作系统中能保证互斥的访问 ptable.

内存中分为用户空间和系统空间。每个空间都维护着自己的堆栈，代码，数据。两个空间表示着两个状态，管态，目态。

进程的堆栈都是进程在运行所需保存的一些变量值。堆是自底向上的，其大小是编译时静态分配的。而栈是自顶向下的，其大小是运行时动态分配的。