

XV6 源码第一次阅读报告

进程与线程部分

目录

[一、进程模型介绍](#page3) [2](#page3)

[二、XV6 源码阅读](#page4) [3](#page4)

[基本头文件](#page4) [3](#page4)

[进程相关部分](#page5) [4](#page5)

[三、Xv6 系统实现原理](#page6) [5](#page6)

[1．进程和线程](#page6) [5](#page6)

[2.进程管理](#page7) [6](#page7)

[3.进程状态](#page7) [6](#page7)

[4.多进程调度](#page8) [7](#page8)

[5.内核态与用户态](#page8) [7](#page8)

[6.进程与内存](#page8) [7](#page8)

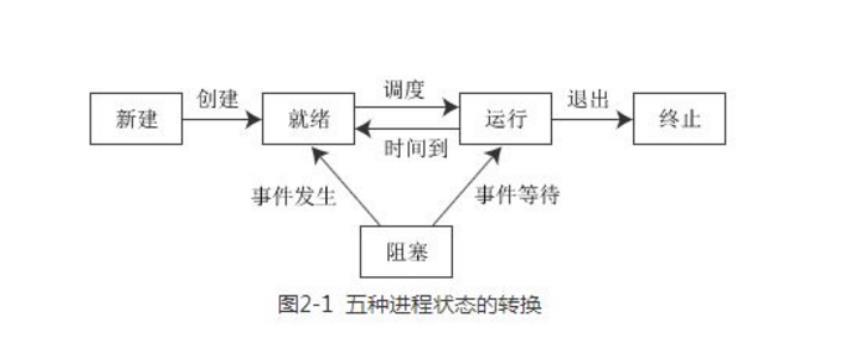
[7.fork 系统调用](#page9) [8](#page9)

第 1 页

一、进程模型介绍

进程是多道程序设计的核心，是计算机科学中最重要和最成功的的概念之一。进程概念的诞生是为了提高系统效率，操作系统允许程序并发执行，多道程序同时执行，间歇占用 CPU。

* 进程是操作系统对一个正在运行程序的抽象，进程的内容包括程序的代码段，数据段和进程控制块。
* 进程利用不同的状态反映程序对系统资源的占用情况，一般包括创建、就绪、运行、阻塞和终止。



* 进程控制块 PCB 是操作系统表示进程一个专门的数据结构，用于记录进程的各种属性，描述进程的动态变化过程，是系统感知进程存在的唯一标识.



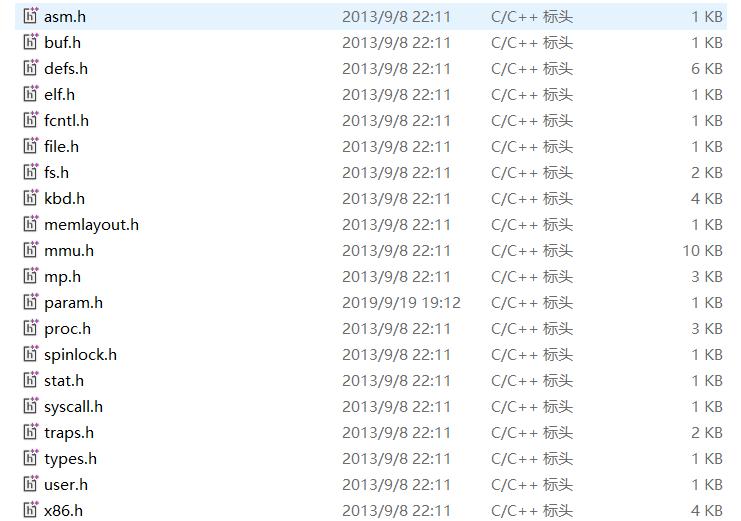
第 2 页

Xv6 使用页表（由硬件实现）来为每个进程提供其独有的地址空间，进程状态包括 unused（未使用）、embryo（初始态）、sleep（等待）、runnable（就绪）、run（运行）、zombie（僵尸），六种状态。

二、XV6 源码阅读

基本头文件

在 xv6 源码中，首先定义的是一系列的.h 文件，定义了基本的数据类型、数据结构、以及函数名。



Types.h 定义无符号变量的别名；

param.h 定义了操作系统的一些基本指标的数量，如最大进程个数，打开文件的个数，CPU 的最大数量，进程能打开的文件描述符数等；

memlayout.h 主要用于声明一些和内存与地址相关的常量与方法，包括虚拟内存转物理内存的方法以及物理内存转虚拟内存的方法等。防止内存泄漏；

第 3 页

defs.h 是函数和结构体定义文件，内部包含了许多类的前向声明，同时声明了许多全局函数，这些全局函数的实现在具体使用的.c 文件模块定义

x86.h 让 c 代码使用特殊的 x86 汇编的一些函数，并声明从用户态陷入内核态的数据结构 trapframe；

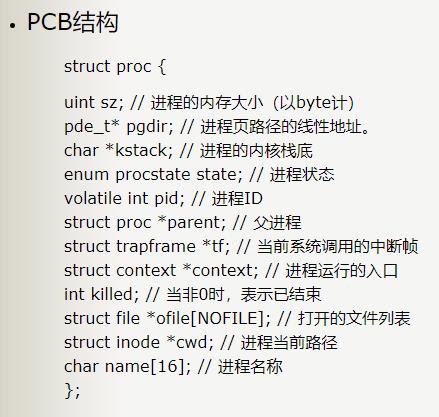
Asm.h 汇编语言的宏定义；

mmu.h 定义 x86 内存管理单元，进程地址空间详细数据结构，包括控制寄存器 CR 上的不同位代表的含义；

elf.hELF 可执行文件的格式，包括 ELF 头的数据结构等。

进程相关部分

Proc.h 头文件，定义了 cpu、上下文 context 和 proc 进程结构体 PCB，其中进程结构体包括进程占用的内存大小，页表，进程栈空间的基址，进程状态，进程 id，父进程，中断陷入等，如 p->state 指示了进程的状态：新建、准备运行、运行、等待 I/O 或退出状态中。p->pgdir 以 x86 硬件要求的格式保存了进程的页表。xv6 让分页硬件在进程运行时使用 p->pgdir。进程的页表还记录了保存进程内存的物理页的地址。



第 4 页

**Proc.c** 是声明关于进程核心函数的代码，**ptable** 是进程表数据结构，可以被lock 和释放。**Alloproc** 是建立进程的函数，首先在进程表中寻找没有被使用过的进程，如果找到则将进程状态切换为 EMBRYD,并尝试分配内核栈，如果分配失败则函数返回。若分配成功，则为系统调用预留空间，并初始化进程上下文。**Userinit** **函数**，建立第一个用户进程的初始化，并设置进程表和中断陷入。**Growproc** **函数**，为进程增加内存空间。**Fork** **函数**，用于子进程的创建，将父进程的信息都复制给子进程。

**Exit 函数**：退出进程并关闭所有打开的文件，再将子进程的父进程均改为初始进程，当前进程状态变为 ZOMBIE 僵尸进程。并调用 sched。**Wait** **函数**：等待子进程退出，处理僵尸子进程，将僵尸进程的所有内容重置，最后返回该僵尸进程的 pid。 **Scheduler 函数**：从就绪队列中选择runnable可运行的进程，将进程状态改为运行。**Sched** **函数**，切换到内核。**Yield** **函数**，放弃 cpu 占用。Forkret 函数，fork 返回时调用。**Sleep** **函数**：使进程睡眠，并设置 chan 进程睡眠队列。Wakeup 将睡眠队列中所有进程唤醒。**Kill** **函数**，将进程 PCB 中的 kill 值置为 1，即进程被杀死。 **Procdump 函数**，将进程列表运行中的打印出来。

其中 acquire 和 release 两个函数给资源加解锁，避免死锁问题。

Vm.c 是关于 cpu 虚拟内存管理函数的代码，包括页表申请，页表释放，页目录的切换，cpu 与进程的绑定等

Kalloc.c 物理内存的申请与释放。

Switch.s 内部是上下文切换的汇编指令，判断要执行的机器指令，保存换出进程的现场，加载换入进程的现场，进程上下文切换。

三、Xv6 系统实现原理

1．进程和线程

进程是具有独立功能的程序关于某个数据集合的一次运行活动，是系统进行资源分配和调度的独立单位。

线程是进程中关于 cpu 资源分配的部分，是系统资源分配的最小单位。

XV6 中进程是通过 proc 结构体来记录，用 ptable 记录所有进程。XV6 中没有实现线程。

第 5 页

2.进程管理

进程管理的数据结构是 PCB 进程控制块，在 windows 中 EPROCESS 、KPROCESS、PEB，在 linux 中为 task\_struct，在 xv6 中为 proc 结构体。在 proc 结构体中，包含 sz 进程占用的内存空间规模；pgdir 进程页表；kstack 进程在内存栈中的基地址；enum procstate state，进程的状态;pid 进程的标识符；proc patent:父进程;trapframe \*tf 进程中断保存的寄存器;context \*context 进程上下文；\*chan 睡眠进程列表；killed 若不为 0，进程被杀死；\*ofile[NOFILE]进程打开的文件列表； inode\*cwd 进程当前的目录；name[16]进程的名字。

操作系统通过不同的函数控制进程状态的切换，改变 PCB 中相应的数据达到管理进程管理数据结构的目的。

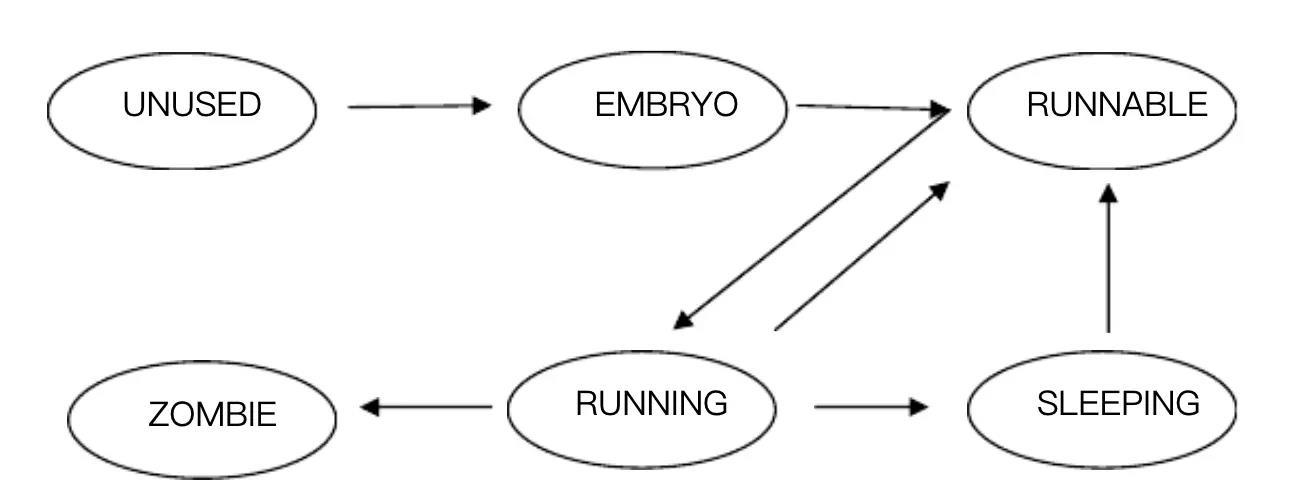
在 xv6 中进程的初始化会调用 allocproc()函数，首先定义进程 p，然后将进程列表上锁，在进程列表中选出尚未被使用的进程，如果没有则释放进程列表返回 0。将未使用进程的状态切换为 EMBRYO,如果物理空间开辟失败，则重置进程状态为UNUSED,返回 0.再分配陷阱帧。

3.进程状态

进程状态一般有创建状态、退出状态、运行状态、就绪状态、阻塞状态、阻塞挂起状态、就绪挂起状态。

在 Linux 中进程状态有就绪、深度睡眠、浅度睡眠、占有 CPU 执行、暂停和死亡但户口未注销。

在 XV6 中，进程状态包括 unused（未使用）、embryo（初始态）、sleep（等待）、runnable（就绪）、run（运行）、zombie（僵尸），六种状态。



操作系统利用对进程不同状态的控制来实现 cpu 和系统资源的分配，占用不同的资源对应不同的进程状态，并实现操作系统的并发，即多道程序的交叉运行。

第 6 页

4.多进程调度

在 XV6 中，利用 allocproc 函数改变进程状态，利用父进程调用 fork()函数创建子进程。在 scheduler 函数中寻找进程队列中可运行的进程，用 switchnvm 切换到进程的页表，将进程状态改为运行中，并用 swtch 切换到该进程中运行。此外，在中断陷入后也会进行进程的切换。

调度算法有：先来先服务、短作业优先算法、最高响应比优先算法、时间片轮转算法、高响应比优先算法、多级反馈队列算法，公平共享调度算法等。

操作系统设置最大进程数的原因是因为内存空间资源有限，需要限制进程创建的数量，其次太多的进程也会导致 CPU 调度难以进行。XV6 中最大进程数为 NPROC=64 个。进程的切换有 scheduler 函数，从就绪队列中选择进程切换运行，以及中断和陷入导致的进程切换。

进程上下文在 XV6 中是当前进程的程序计数器 PC 和当前运行的 CPU 中各个寄存器的内容。包括程序状态寄存器、栈指针、通用寄存器和其他控制寄存器的值。将 CPU 硬件状态从一个进程换到另一个进程的过程称为上下文切换。

5.内核态与用户态

内核态进程：是操作系统程序运行的状态，一个用户进程也可以因系统调用陷入内核代码中执行时。内核态进程的特权级是最高的，可以执行专门的内核操作指令。

用户态进程：一个进程执行用户自己代码时处于用户态，特权级最低

区别在于：内核态和用户态是两种不同的进程运行状态，拥有不同的操作权限。

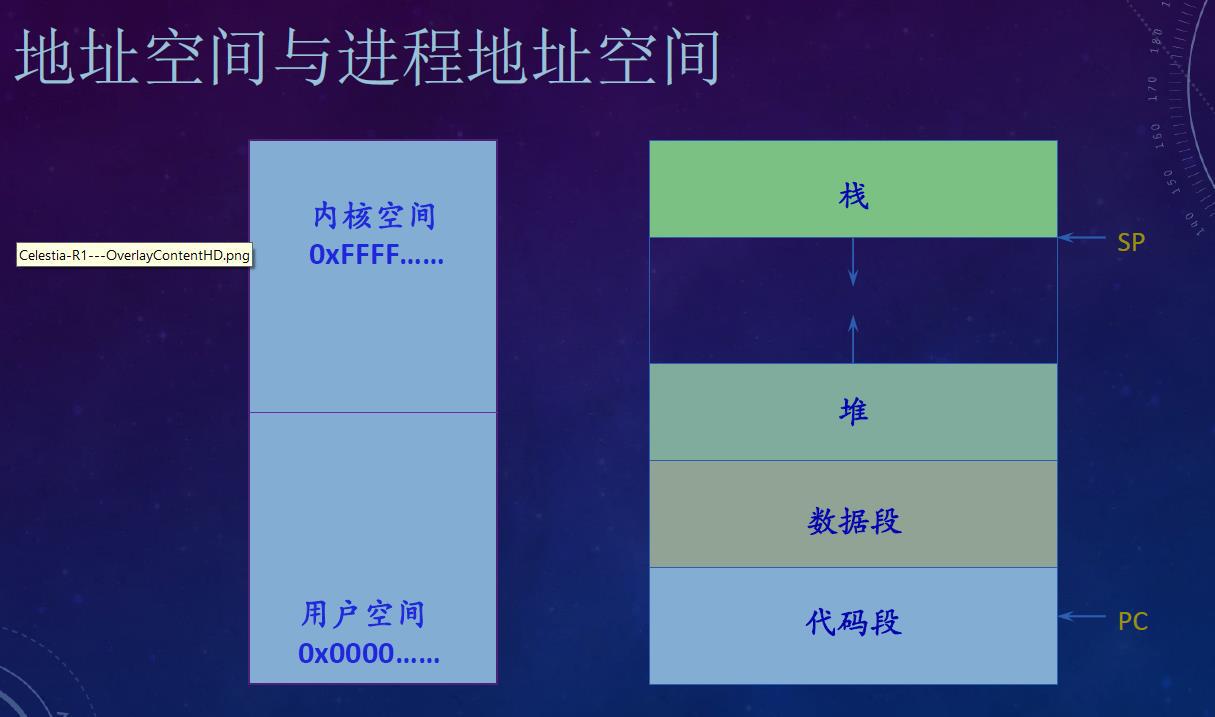
6.进程与内存

操作系统给每个进程都分配了一个地址空间，其中内核态进程一般在内核空间，用户态进程一般在用户地址空间。进程在内存中占用的空间包括栈、堆、数据段和代码段。

栈是程序执行前静态分配的地址空间，栈会向下增长。

第 7 页

堆是程序执行时，根据需要动态分配的空间，包括 malloc、calloc、realloc 函数分配的空间。堆内存由所有线程共享，并会向上增长。



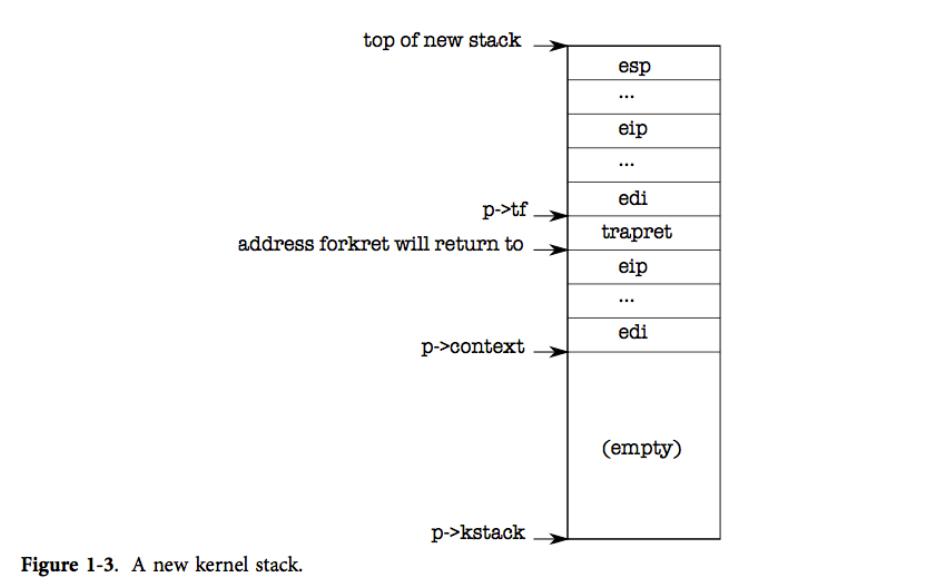
7.FORK 系统调用

一个进程可以通过系统调用 fork 来创建一个新的进程，这个新进程被称为子进程，子进程的内存内容同创建它的进程（父进程）一样。

Fork 函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。Fork 函数在子进程和父进程中都会被执行，并且在父进程、子进程中都返回（一次调用返回两次），子进程中，fork 函数返回 0，在父进程中，fork 返回新创建子进程的 pid。

fork 函数的执行过程中将子进程的返回值存放在子进程的寄存器 eax 中，将子进程 pid 直接 return 返回给父进程。而子进程执行时，会返回 eax 寄存器中的值 0.

第 8 页



eax 寄存器，当作变量通用的返回值寄存器，默认函数返回值都通过 eax 来传递。将子进程中断陷入帧的 eax 寄存器设为 0，子进程执行会默认返回 0。

第 9 页