## Xv6进程与线程阅读报告

1. 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？

**进程**是指一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程。直观地来说，进程是执行中的程序，是一个活动的实体。进程包含了正在运行的一个程序的所有状态信息，通常由程序代码段，数据段，进程栈和堆等，是计算机资源分配的基本单位。当一个可执行文件被载入内存执行时，这个程序就成为一个进程。

**线程**描述指令流执行状态。它是进程中的执行逻辑流的最小单元，在实现了线程的操作系统中，线程是CPU调度的基本单位。线程是进程的一部分，一个进程可以有多个线程，属于同一进程的多个线程共享代码段，数据段等操作系统资源。

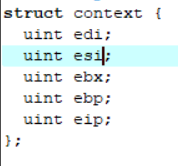
需要注意的是，即使在实现了线程的操作系统中，CPU也不处理进程内部的线程的管理，CPU只给进程进行调度，在进程内部由进程来为线程分配资源，进行管理，而CPU是看不参与部分内容的，即线程对于CPU而言是不可见的。

一个系统的资源分配单位是进程，线程并不独立拥有资源，一个进程中的多个线程共同分享进程中的资源。

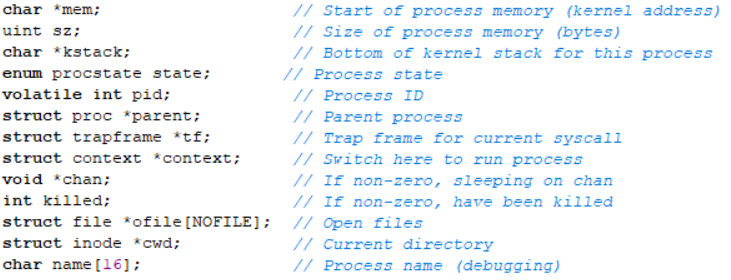
对于实现了线程的操作系统，调度的基本单位是线程，这是在进程内部实现的。而对于不支持线程的操作系统，调度的基本单位是进程。

在XV6中，没有发现与线程有关的内容，因此我们认为xv6没有实现线程。Xv6在proc.h中定义了关于进程的一些数据结构，具体而言：

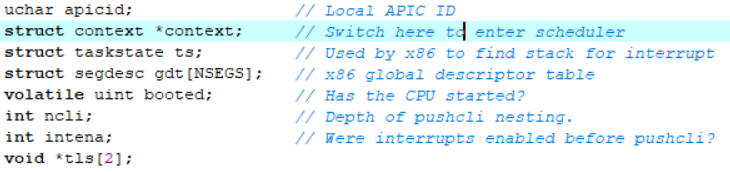
1. context：进程上下文，用于保存在上下文切换时不会被保存的寄存器值，用于再一次调度时的状态恢复。



1. procstate：定义了六个进程状态——UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE。
2. proc：定义了进程控制块的内容，包含程序状态，程序计数器，内存管理信息，IO状态信息等内容，详见下图



1. cpu：定义了关于cpu状态的量



1. 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称为PCB（进程控制块）。具体来说：

1. **Windows**中的进程控制块是**EPROCESS结构,** 线程控制块是ETHREAD结构.

在EPROCESS结构中，保存了进程状态，程序计数器，索引寄存器，堆栈指针等很多信息，鉴于EPROCESS结构内容过于庞大，本报告中不引入关于其内容的具体介绍。

windows创建进程的步骤：

1）.验证参数，将Windows子系统标记和选项转换成native对应的部分。

2）.打开 进程中要被执行的image文件（.exe）。

3）.创建Windows executive进程对象。

4）.创建初始化线程（栈，上下文，Windows executive线程对象）。

5）.初始化进程。

6）.开始执行初始化线程。

7）.完成地址空间的初始化，开始执行程序。

1. **Linux**的进程控制块为一个由结构**task\_struct**所定义的数据结构. 其关键内容如下所示：

struct task\_struct{

...

unsigned short uid;

int pid;

int processor;

...

volatile long state;

long prority;

unsighed long rt\_prority;

long counter;

unsigned long flags;

unsigned long policy;

...

Struct task\_struct \*next\_task, \*prev\_task;

Struct task\_struct \*next\_run,\*prev\_run;

Struct task\_struct \*p\_opptr,\*p\_pptr,\*p\_cptr,\*pysptr,\*p\_ptr;

...

};

(1)unsigned short pid 为用户标识

(2)int pid 为进程标识

(3)int processor标识用户正在使用的CPU,以支持对称多处理机方式；

(4)volatile long state 标识进程的状态，可为下列六种状态之一：

可运行状态(TASK-RUNING);

可中断阻塞状态(TASK-UBERRUPTIBLE)

不可中断阻塞状态(TASK-UNINTERRUPTIBLE)

僵死状态(TASK-ZOMBLE)

暂停态(TASK\_STOPPED)

交换态(TASK\_SWAPPING)

(5)long prority表示进程的优先级

(6)unsigned long rt\_prority 表示实时进程的优先级，对于普通进程无效

(7)long counter 为进程动态优先级计数器，用于进程轮转调度算法

(8)unsigned long policy 表示进程调度策略，其值为下列三种情况之一：

SCHED\_OTHER(值为0)对应普通进程优先级轮转法(round robin)

SCHED\_FIFO(值为1)对应实时进程先来先服务算法；

SCHED\_RR(值为2)对应实时进程优先级轮转法

(9)struct task\_struct \*next\_task,\*prev\_task为进程PCB双向链表的前后项指针

(10)struct task\_struct \*next\_run,\*prev\_run为就绪队列双向链表的前后项指针

(11)struct task\_struct \*p\_opptr,\*p\_pptr,\*p\_cptr,\*p\_ysptr,\*p\_ptr指明进程家族间的关系，分别为指向祖父进程、父进程、子进程以及新老进程的指针。

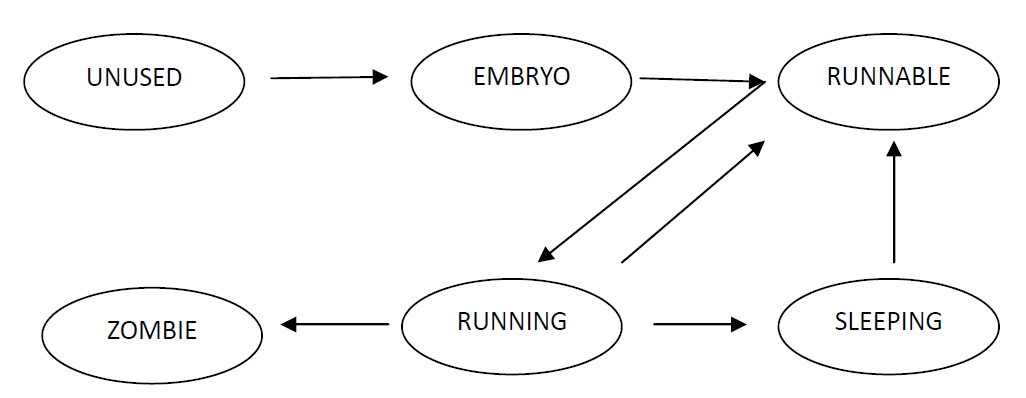
在创建一个新进程时，系统在内存中申请一个空的task\_struct区，即空闲PCB 块，并填入所需信息。同时将指向该结构的指针填入到task[]数组中。

1. **xv6**中的pcb由proc.h中的**proc数据结构**所定义，其内容在1.中已介绍。

XV6 中的proc.c中定义了一系列管理进程的函数。alloc\_proc()函数为进程分配一个未使用的进程号。growproc将进程的内存空间进行扩展。fork（）创建一个与父进程一致的子进程。userinit（）创建第一个用户进程。schedule（）进行cpu的进程调度等。

1. 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？你认为操作系统的设计者为什么会有这样的设计思路？

如前面的1.中所述，proc.h中的procstate结构定义了XV6 进程包括UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE 这6种状态。图示如下：



如前面2.中关于task\_stata的描述，Linux 的进程状态可分为：

可运行状态(TASK-RUNING);

可中断阻塞状态(TASK-UBERRUPTIBLE)

不可中断阻塞状态(TASK-UNINTERRUPTIBLE)

僵死状态(TASK-ZOMBLE)

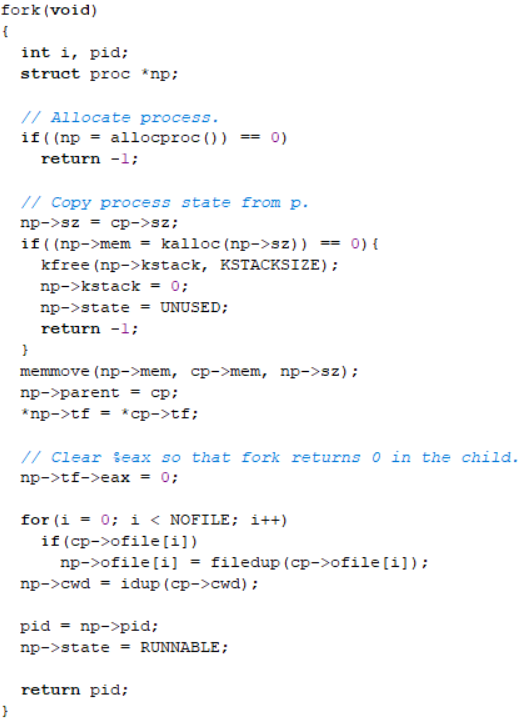
暂停态(TASK\_STOPPED)

交换态(TASK\_SWAPPING)

之所以这样设计主要有两点考虑：首先是为了实现多进程的并发执行，其次是为设计进程调度创造条件。在多进程并发执行的过程中，CPU时间片需要按照一定规则进行调度，保证每个进程都能在合适的时机被分配到运算资源。当RUNNING的进程结束一个时间片而下CPU时，另一个在等待的进程可以从SLEEPING或RUNNABLE状态转换为RUNNING，实现调度。原来的进程则转入RUNNALBE或SLEEPING状态进行等待，直到下一次调度到该进程或等待的资源得到满足，保证并发进程有序进行。另外，进程调度的算法设计需要对进程的状态进行获取，这一模型是一个很好的抽象，抽象化的模型简化了算法的设计过程和设计难度。

1. 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？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？

fork()函数创建一个子进程，其实现如下：



首先，调用allocproc为子进程分配一个proc结构，并用一个指针np指向它。然后，将自身的程序状态拷贝到子进程的proc结构中。然后将父进程的内存空间用memmove函数拷贝到子进程中，将子进程加入调度序列。清空%eax寄存器，使得fork函数在子进程中返回0。最后在父进程中返回子进程pid。

调度方式：每当CPU空闲时，操作系统就必须从就绪队列中选择一个来执行。此时，短程调度程序或CPU调度程序选择一个进程，并为之分配CPU。调度准则包括：CPU使用率，吞吐量，周转时间，等待时间，响应时间，这些都是调度算法的设计依据。

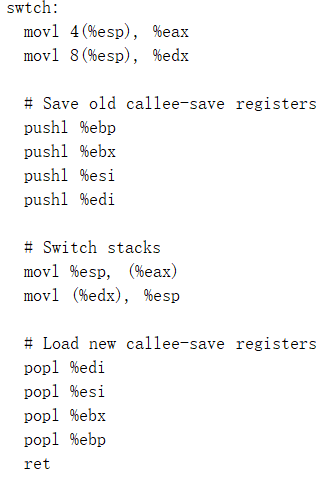
调度算法：先到先服务；最短作业优先，优先级调度，轮转法调度，多级队列调度，多级反馈队列调度等。

操作系统对于最大进程数的限制：首先，进程的数量不能超出pid的表示范围，其次，受系统资源限制，创建一个新进程会消耗系统资源，最主要的就是内存。

xv6最大进程数64，因为在param.h中有如下定义：

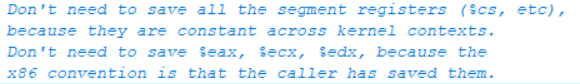


在进程切换时，CPU保存当前进程的的状态，并恢复另一个进程的状态。具体实现在swtch.S中，关键代码如下：



可以看到，swtch函数以两个进程的context数据结构为输入参数，将当前进程的上下文进行保存，调整栈指针的指向位置，并将待调度的进程上下文写入相应的寄存器中。

进程上下文（context）：进程的上下文包括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。当进程切换和发生中断的时候这些信息要保存下来以便于下次运行时使用。在xv6代码中proc.h头文件的context结构记录了需要在进程切换时保存的上下文内容，另外有这样一段说明：



即段寄存器和上述三个通用寄存器不需要在进程切换时显式的保存，因为操作系统会通过某些规则将这些寄存器自动进行保存。

**多进程**是指并发执行，在某段时间内多个进程都在执行，但具体到某个时间点，只有一个进程占用CPU。并发是CPU调度的结果，多个进程不停地上下CPU，制造了一种并发的现象。**多CPU**实现处理器级别的并行，多个CPU分别执行不同的进程，能提高吞吐率，在多CPU的情况下，每一个时间点都有多个进程运行在多个处理器上。但具体到每个处理器，仍旧是前面所说的并发执行的现象。

1. 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**：系统进入内核态之后进行系统操作所产生的进程。拥有较高的特权级别，可以执行特权指令。

**用户态进程**：运行用户程序而创建的进程，特权级别较低，需要通过系统调用转移到内核态才能执行某些特权指令，获取系统服务等。用户态切换为内核态可以通过系统调用，异常和硬件中断三种方式。

它们的主要区别有：处于不同的特权等级，用户态进程执行在用户态，内核态进程执行在内核态；用户态进程可以通过用户运行程序直接进入，而内核态进程需要经过系统调用，异常或硬件终端来调用；另外，内核态进程优先级要高于用户态进程。

1. 进程在内存中是如何布局的，进程的堆和栈有什么区别？

进程包含数据段，代码段，堆和栈，结构如下：

|  |
| --- |
| **栈** |
| **↓**  **↑** |
| **堆** |
| **数据段** |
| **代码段** |

堆和栈的功能和在内存中的区域不同，具体来说：

栈：在内存中向下生长。程序执行前静态分配的内存空间，栈段是存放程序执行时局部变量、函数调用信息、中断现场保留信息的空间。程序执行时，CPU栈段指针会在栈顶根据执行情况进行上下移动。

堆：在内存中向上生长，程序执行时，按照程序需要动态分配的内存空间。