Nachos 源码解读

山东大学软件学院

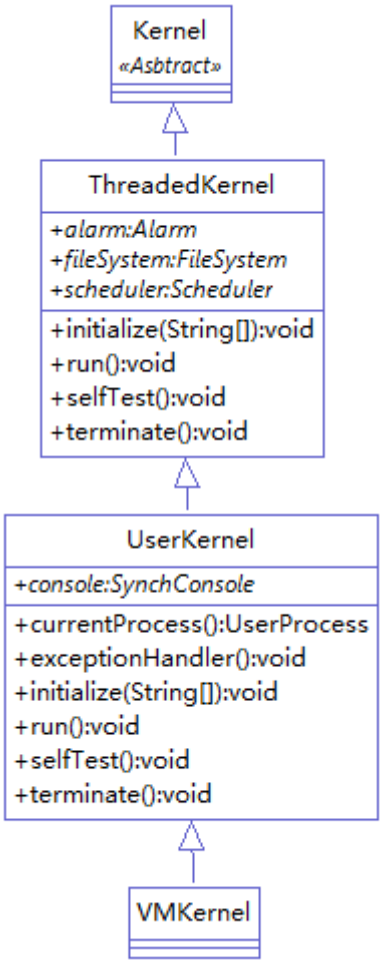
2013 级软件工程

哈根

1、 从 Machine.java 开始——Nachos 内核启动

Nachos 的程序执行从 Machine.java 的 main 方法开始。主要进行的是处理启动参数、载入配置文件、设置工作目录、安装安全管理器、创建设备、并启动第一块 TCB 等操作，在 TCB 启动时会调用 AutoGrader 的 start 方法，其中启动了内核。

在 AutoGrader 的 start 方法执行时，AutoGrader 首先会解析启动命令传入的参数，接着执行初始化操作，然后从配置文件中读取 Kernel.kernel 的值，构造内核，并且执行内核的初始化方法。紧接着，执行 run 方法，内含 kernel 的自检、run 方法以及最后的终止。



由于不同 project 使用的内核不同，所以各个内核的效果也是不一样的。下面以 project1 的内核为例简要说明。

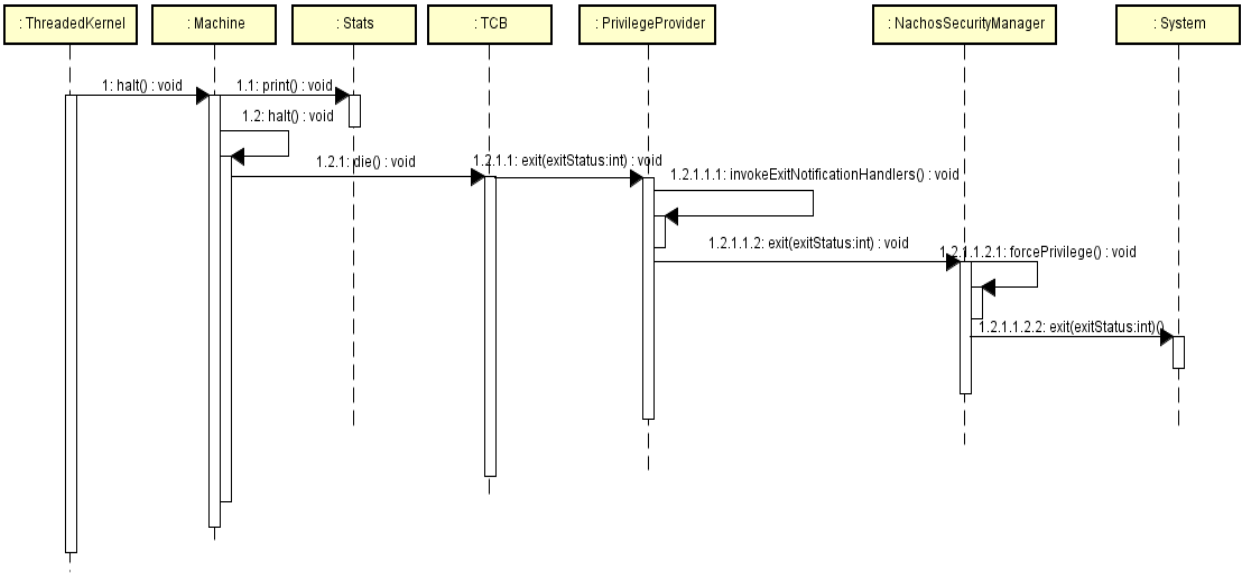
在 Kernel 抽象类中，会检查 Kernel 中的静态成员变量 kernel 是否为空，并把当前对象赋值给 Kernel，这也决定了内核可以通过 Kernel.kernel 调用。在构造器结束后，AutoGrader 会执行 Kernel 的 initialize 方法，对于 phase1 的 ThreadedKernel 而言，在该方法中初始化了调度器，文件系统，alarm 等。在执行完 initialize 方法后，会进入 AutoGrader 的 run 方法，依次执行内核自检、运行、停机操作。

对于 phase1 的 ThreadedKernel 而言，run 方法是空的，也就是执行完开机自检后就自动停机了。开机自检依次 KThread、Semaphore 以及 SynchList 的自检操作。

在phase2中，启动的是UserKernel，由UML图继承关系知，该类继承了ThreadedKernel，在该类的方法体中也多见**super**关键字，即ThreadedKernel做的事情，UserKernel也做了。初始化过程中，UserKernel还初始化了一个控制台，用来接收屏幕中用户的输入，并且还设置processor的异常处理器（exceptionHandler），用来处理指令周期中中断周期的各类系统调用以及其他中断异常。自检（selfTest）过程中还会接收用户字符，来判断控制台是否正常。run

方法会初始化一个用户进程，并启动它。该用户进程是一个UserProcess类或其子类，构造哪个类由Kernel.processClassName决定，载入哪个进程由Machine.shellProgramName决定，而该参数的默认值为启动nachos时-x后参数的值，如果没有-x参数，会读取配置文件中Kernel.shellProgram的值（详见Machine.getShellProgramName和参数解析的方法），也就是说我们可以通过2种方法告诉nachos默认的启动程序是什么，在proj2/nachos.conf中默认是halt.coff，调用halt系统调用实现了关机操作，当phase2完成时，我们可以运行sh.coff,在nachos上启动一个简单的shell。

当Kernel.run运行完毕，会执行Kernel.terminate，而该操作执行的Machine.halt，期间向屏幕输出了当前nachos的状态，并执行Machine.terminate,而该方法调用了TCB.die，而TCB.die通过特权调用，最终调用了nachos.security.NachosSecurityManager下的exit方法退出，期间调用了Privilege的invokeExitNotificationHandler方法，执行各类退出通知。



2、 Nachos 参数解析、设备创建和 Debug 方法

1) Nachos 启动参数解析

Nachos 启动参数处理主要集中在 Machine.processArgs 方法。

首先处理的是-d 参数，该参数与 nachos debug 信息密切相关。Nachos 会把-d 的值传入 Lib.enableDebug,然后开启测试开关；接着是-h，表示帮助参数，会打印帮助文档，然后退出；-m 参数指定物理页数目，然后赋值给 Machine.numPhysPages。

-s 会指定随机数种子，赋值给 Machine.randomSeed;-x 指定内核启动的用户程序名称，赋值给 shellProgramName;-z 会输出版权信息，然后退出；-[]后跟配置文件名称，赋值给 configFileName,如果不存在默认为 nachos.conf,--后的参数会赋值给 autoGraderClassName。接着设置 Lib 中随机数种子，初始化随机数。

2) Nachos 配置文件

在解析完参数后，会根据当前 Machine 中 configFileName 的值读取配置文件。主要处理的方法是 Config 类的 load 方法。会读取对应的配置文件，然后将其 key 与 value 放入一个 map 对象中。在需要时通过 Config.get 方法获取参数。

3) Nachos 设备创建

设备创建主要是 Machine 的 createDevices 方法。依次创建中断、定时器等，并且会根据配置文件中是否指定 Machine.bank、Machine.processor、Machine.cosole、Machine.stubFileSystem、Machine.networkLink 来依次创建对应的对象。部分参数是由传入参数和配置文件参数以及系统默认值综合指定，比如 numPhysPages，会先检查是否有传入参数，如果没有会读取配置文件中的参数。

4) Nachos Debug 方法

在 Nachos 系统中，随处可见到的是 Lib.assertTrue()方法以及 Lib.debug()方法，其中 Lib.asserTrue()是断言，为了保证系统执行的正确性而存在。而 Lib.debug 方法则是方便用户进行代码调试。

在 Lib 类里，有一个长度为 0x80 的 boolean 数组，大小正好与 ASCII 码数目相同。当 Machine 参数-d 解析时，会调用 Lib.enableDebugFlags()方法,解析-d 后的字符串，对字符串中的每个人 char，将对应数组中的该为置为真。在执行 debug 时会检测该位是否开启，进而输出语句。

比如在UserProcess类中，常常可以见到Lib.debug(dbgProcess,xxx)，而dbgProcess为一个char型常量，不同的类中一般都有这样的一个常量符号作为调试操作符，这里dbgProcess的值为’a’,是UserProcess的调试符。机器传入-d a时，数组中第’a’位值为真，会将xxx输出。如果-d后面没有字母为a,则xxx不输出。需要注意的是，debug方法中通过test检测对应位是否开启，阅读test方法发现，如果

-d传入了’+’，则所有调试信息都会输出。

3、 Nachos 中断简述

在 Nachos 的 Machine 启动时，有一步是创建各种设备，中断和定时器首当其冲。而这两个设备也与系统时间的增加、中断的触发等密切相关。

1) PendingIntterrupt 中断处理与 Intterupt 增加中断调度

PendingInterrput 对中断做了包装，内含一个 Runnable 对象（实际要调度的中断）、调度时间、中断类型、和中断 id。中断 id 随着 pendingIntterupt 对象的增加而增加，由 numPendingIntteruptsCreated 决定，这个对象不是 static 的，说明允许有多个中断管理器，只需每个中断管理的中断处理器 id 不同即可，每构造一个对象，该中断的 numPendingIntteruptsCreated 加 1。

当添加一个新中断时，默认调用 IntteruptPrivilege 的 schedule 方法，该方法实际上调用的是 Intterput 的私有 schedule 方法，该方法将含中断处理代码的 Runnable 对象 handler 包装为 PendingIntterupt 对象，并储存在 Intterupt 中断对象的 pending 队列里。每调度一次 pending 队列中满足时间要求的对象会清空，对于部分需要反复执行的中断处理程序，其 handler 里往往含有再次调度的代码，保证每隔一定时间进行调度。

2) Intterupt 中断查询

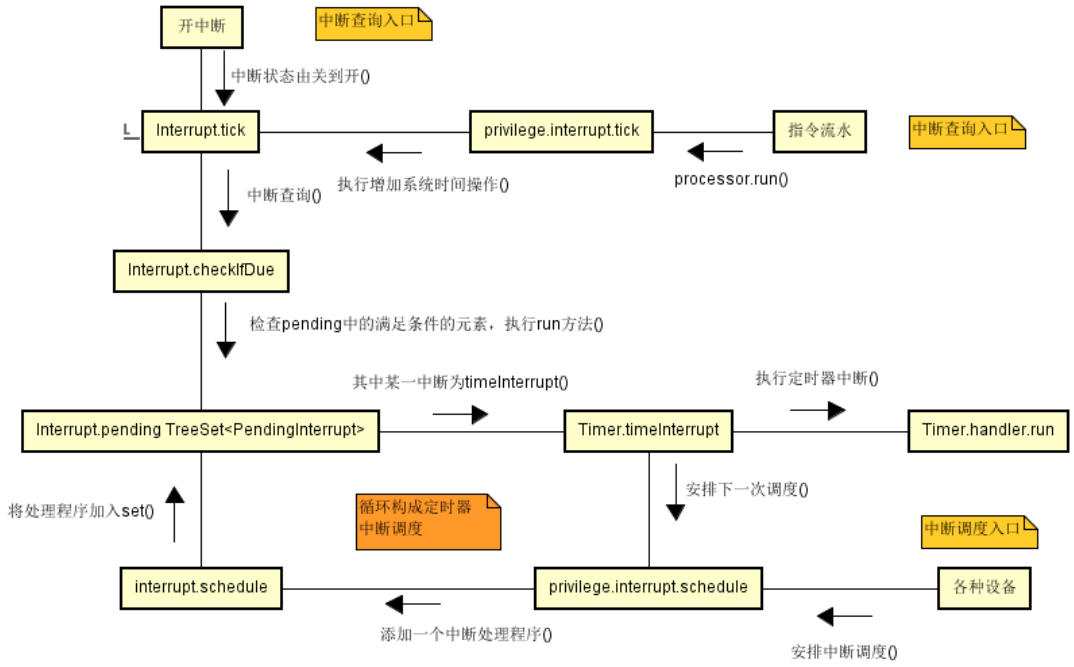
中断查询主要是通过开中断操作以及指令流水中的中断周期来进行。其核心代码是 intterupt.tick，其他程序性通过 privilege.interrupt.tick()调用。该方法根据用户模式还是系统模式增加不同的时间，调用 checkIfDue 来执行中断查询。

每次中断查询时，从 pending 中将里面小于当前时间的 pendingIntterupt 对象依次取出，然后执行。即完成了一次中断查询。对于反复执行的某些中断，一般在执行开始后或结束前会在 handler 中安排下一次调度，即下次执行时间。

3) Timer 计时器类

该类是计时器类，控制着系统时间和计时器中断。其核心是 Timer.scheduleInterrupt() 与 Timer.timerInterrupt()相互作用维持计时器中断。Timer.scheduleInterrupt()中获取

Stats.TimerTicks 来安排未来的调度时间（也就是调度间隔）并利用随机数制造调度时间的不稳定性（但该随机数的平均水平并不维持在 0，也就是平均调度间隔不等于 Stats.TimerTicks），当 timerInterrput()被执行，会再次执行 Timer.scheduleInterrupt()安排下次的调度，而 timerInterrrput()中会执行 Timer.handler 的 run 方法，这个 handler 是由 setHandler 传入的定时器中断。至于 AutoGraderInterrupt，虽然 delay 为 1，但是由于其 scheduleInterrupt 安排在 timerIntterupt 中，也是每 Stats.TimerTicks 执行一次，该中断处理程序会对权限进行检查，保证调度器有权限来调度。



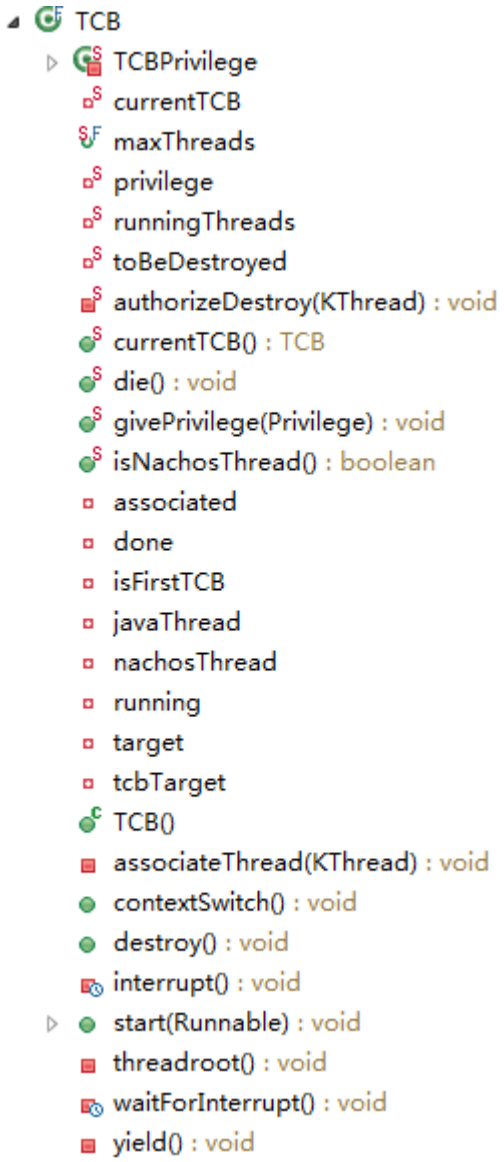
4) Alarm 类

该类负责向 Timer 设置定时器中断(Timer.handler)，还提供一个方法让线程等待一定时间运行。而一段时间后的唤醒正是在定时器中断中完成的。可以模拟 Interrupt 与 PendingIntterupt 的关系，在 alarm 中用类似的方法实现该功能。

4、 Nachos 内核线程及调度算法简述

1) 漫谈 TCB

TCB 全称为 Thread Control Block，即线程控制块。对每个 TCB 而言，都管理着许多与线程相关的参数。Nachos 中有 TCB 类，每个 nachos 线程都对应着一个 TCB，负责处理 Nachos 线程调度的部分底层细节，并对特定线程做标记。

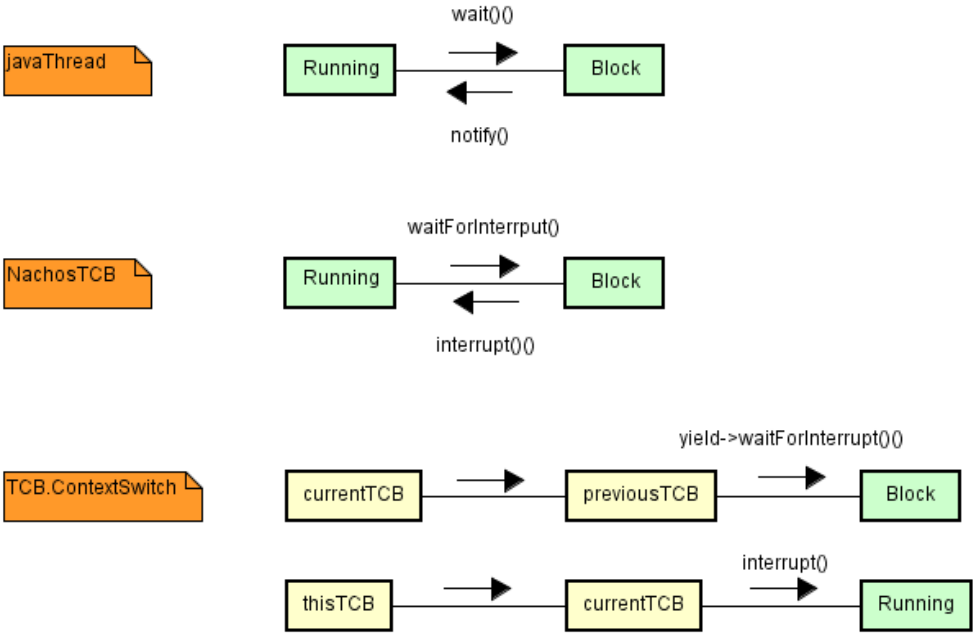


通过 TCB 的方法和属性列表可知，TCB 有 currentTCB、privilege、runningThreads、toBeDestroyed 等静态成员变量。其中 privilege 对象赋给 TCB 类执行特权操作的权利。这些特权操作包括退出系统、以及创建一个 java 底层的线程等。runningThreads 中存储着当前所有运行着的 TCB，此处的 running 意为“启动但未终止”，并非线程的“running”态。而 currentTCB 记录着当前运行的 TCB，toBeDestroyed 表示一个即将销毁的 KThread 线程。这个线程的销毁过程在 KThread.restoreState 中被调用，而该方法在 KThread 的构造方法、KThread 第一次被调度（runThread 方法）和 run 方法中都有应用。也就是刚刚结束的线程是在即将调度到的线程中被销毁的。

对于每一个 TCB，start 方法会判断其是否是第一个 TCB，对于第一块 TCB，直接执行 threadroot 方法，对于其他 TCB，利用特权构造一个 java 线程，在线程中执行 threadroot 方法。

注意在 threadroot 中，新构造的线程被阻滞了。在 javaThread.start 之前，当前 TCB 会暂时关闭，在 threadroot 中被唤醒，然后新构造的线程 waitForInterrupt，并且在 KThread 中，新构造的线程会进入就绪队列，等待被调度。对第一个线程而言，在 threadroot 中会给 currentTCB 以及 running 赋值。

同时，TCB 还负责着线程的等待与唤醒工作。等待（waitForInterrupt）对应 java 线程的 wait 操作，唤醒（interrupt）对应 java 的 notify 操作。对应关系如图。



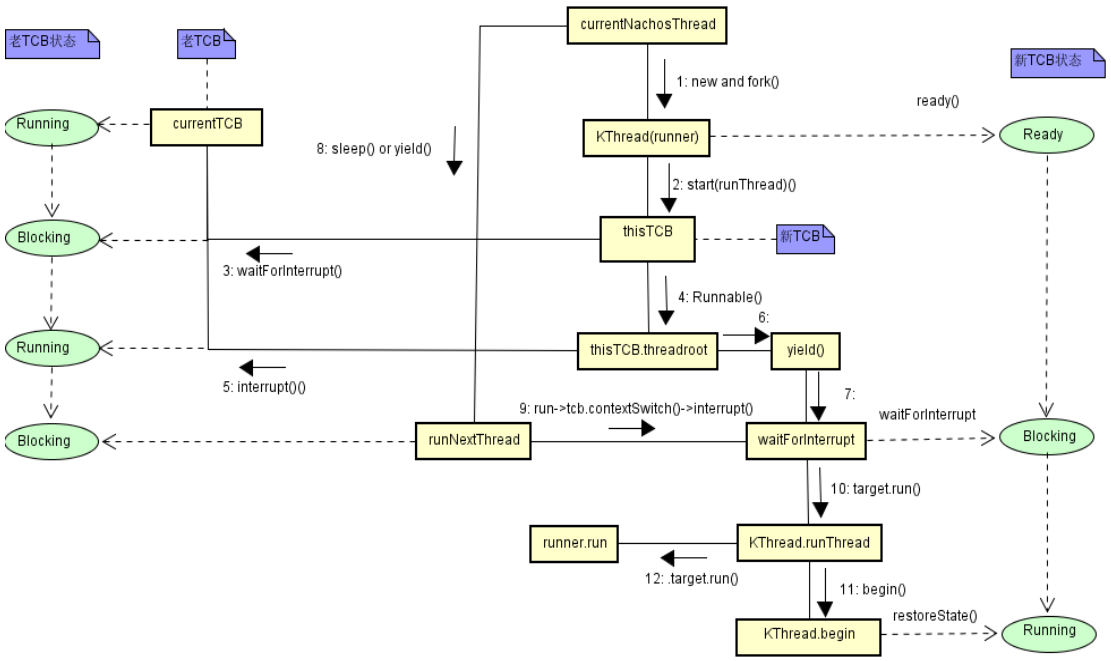
其中，thisTCB.contextSwitch 方法使 currentTCB 进入等待状态，并且使调用该方法的 TCB 成为 currentTCB 并开始运行。

2) 论一个内核线程（KThread）的初始化和 fork

当一个 KThread 初始化时，会判断 KThread 静态成员变量 currentThread 的值，进而确定是否是第一次初始化。对于首次初始化的 KThread，会初始化全局就绪队列、为 tcb 和 currentThread 赋值、修改当前 KThread 状态为 running、创建空闲线程（idleThread）等操作。对于后续构造的 KThread，每次 new 一个属于自己的 TCB。除此之外，会把传入的 Runnable 实例赋给 target。

当一个 KThread 执行 fork 操作时，会调用初始化时的 tcb（第一个 KThread 是不被 fork 的，因为第一个 KThread 和第一个 TCB 均独立创建）。此时传入 tcb 的 Runnable 接口执行的操作时 runThread，即在第一次被调度的时候，会依次执行 begin、target.run、与 finish。begin 主要负责检查 toBeDestoryed 变量，如果变量非空，销毁该变量代表的线程，target.run 方法即要运行的 Runnable 实例中的程序，finish 做一些善后工作，包括取出等待队列中的线程进入就绪队列，将当前线程设置到 toBeDestroyed 令下一个被调度的线程销毁。

注意，执行完 fork 操作，该线程在就绪队列中，而未马上被调度。



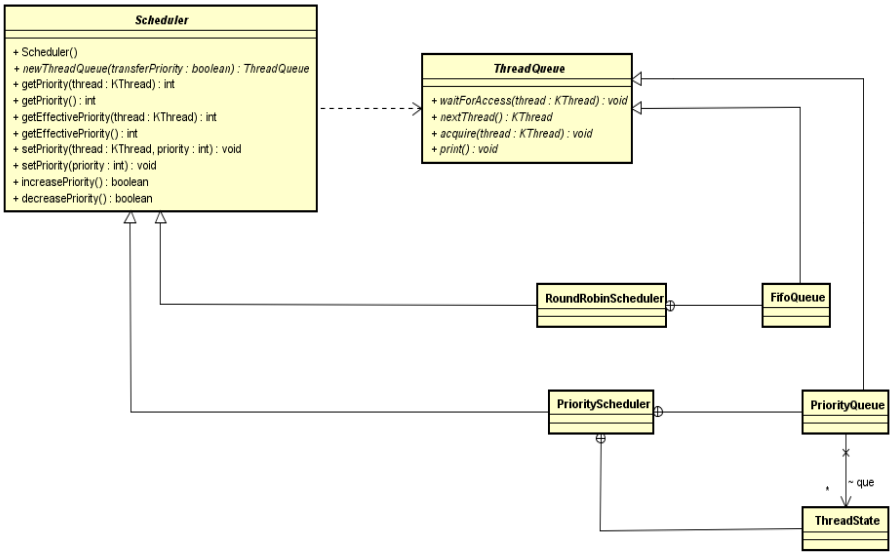
3) ThreadQueue 线程队列及调度算法关系

在 ThreadedKernel 中初始化了系统调度器，所有的调度都实现了 Scheduler 提供的接口。系统提供的默认调度算法是 RoundRobinScheduler，即轮转法调度。

在 Scheduler 中，newThreadQueue 会返回一个新的线程队列。对于任何一个调度器，都对应一个 ThreadQueue 子类，即线程队列。无论是线程的全局就绪队列，还是实现 join 方法时的等待队列，其队列都应该从调度器中获取。例如轮转法调度对应着一个 FifoQueue，即先进先出队列，其内部实现为一个 LinkedList 链表。

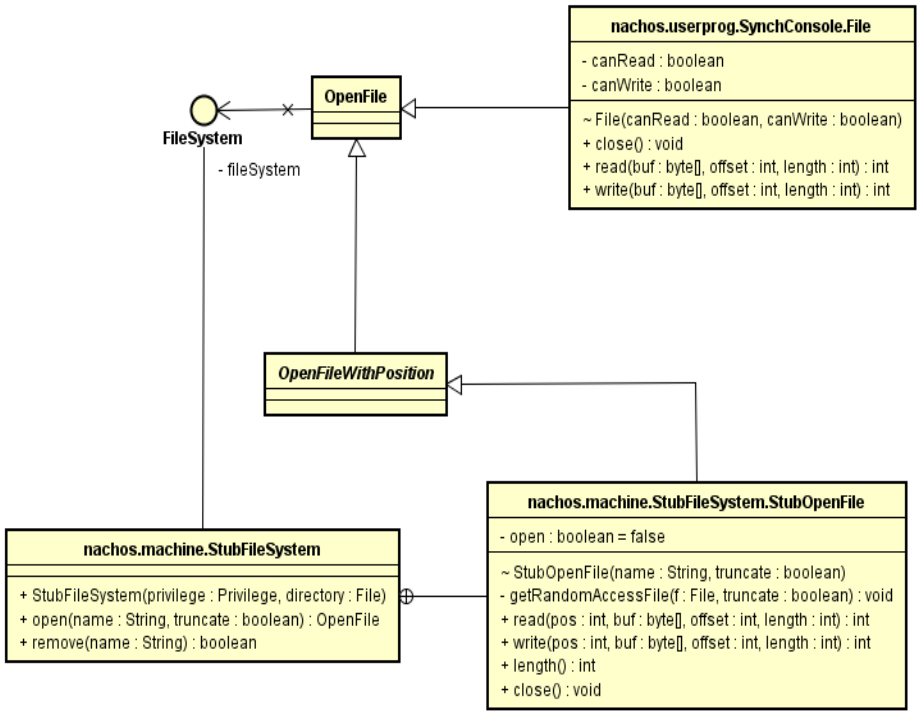
对于任何一个队列，都有入队和出队操作。ThreadQueue 中，waitForAccess 是入队，但是读代码发现，需要我们人工执行 sleep 操作。nextThread 为出队操作，出队后可执行 run 方法运行。

对于 PriorityScheduler，还提供了改变优先级相关的方法，优先级通过 ThreadState 包装 KThread 实现，增加优先级这一属性，在题目中实现自己的优先队列 PriorityQueue，通过对 PriorityQueue 选取合适的数据结构，可以让每次出队时间变为 O(logn),而传统的线性队列为 O(n)。



5、 Nachos 文件系统简述

Nachos 文件系统都实现了 FileSystem 接口，提供打开文件和删除文件的方法。所有的文件都要实现 OpenFile 类，作为文件系统返回的文件。OpenFile 中提供了文件基本的读写操作。



在 nachos 的 SynchConsole 中，把控制台也当作文件处理，每个用户进程的 0 号和 1 号

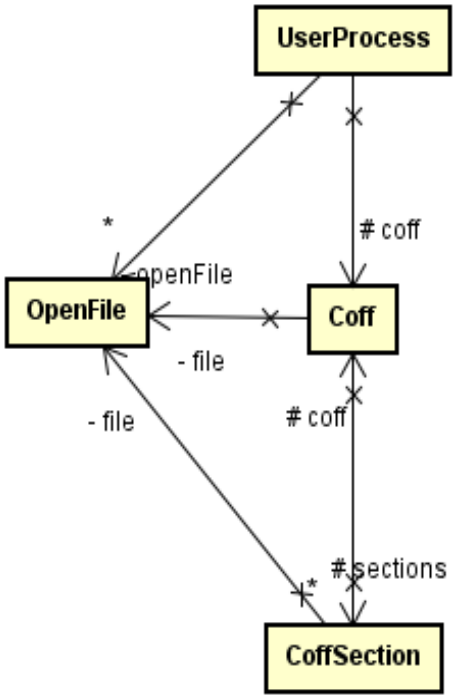
文件描述符分别代表标准输入和标准输出。他们的文件同样继承自 OpenFile，有独立的实现。

在 StubFileSystem、StubOpenFile 等类中，其利用特权操作，依赖 java 对本地文件的访问，实现了一套集文件打开、创建、读取/写入指定位置、关闭、删除等操作的文件系统，或者说，对 java 文件操作进行包装，向上提供为 StubFileSystem 的接口。

在 SynchConsole 中，依靠 SerialConsole 接口、StandardConsole 的实现和中断控制，对 System.in 和 System.out 进行包装，向上提供文件接口，实现了将输入输出当作文件的能力。

6、 Nachos 用户进程、处理器和指令简述

1) Nachos 用户程序解析



通过阅读 nachos.userprog 部分的代码发现，用户进程是通过 UserProcess.load 方法将程序载入内存的。而每个可执行文件用一个 Coff 对象包装。对传入的每个文件名，load 方法先根据文件名读取该文件，然后将文件作为构造方法参数构造一个 Coff 对象，在 Coff 对象中，会对文件进行一个整体的解析，获取代码段（CoffSection）数目、程序入口等信息，并构造代码段。然后 load 方法会获取该程序的每段代码段，统计页的数目和参数的数目，参数会占一页内存，进而将代码段和参数按页载入内存。

这里包含了一个参数数组存入内存的方法。这种方法在书写 exec 系统调用时可能用到，在书写系统调用时可能会读取内存中存入的参数，这里简述一下 load 方法中存入字符串数组的方法。设置 2 个变量 entryOffset 和 striingOffset，分别表示下一个要存储的字符串地址的地址和字符串的地址。entryOffset 初值为当前页页首，stringOffset 初值为 entryOffset 的值向后偏移 4*args.length。由于内存中每个地址代表的单元最小为一个 byte 型数据，4 个 byte 型数据可以组装成一个整型数据。所以对于每个字符串的地址，nachos 采用一个整型来存。即在该页的头 4*args.length 中，每 4 个字节存储着一个地址，可以读取该地址然后利用该地址读取传入的参数；4*args.length 之后的地址空间，依次写入一个字符串和空终止字符 0 作为每个字符串的结束标识。

将程序体写入内存的方法为 UserProcess.loadSections.在该方法中会完成进程页表和内存物理页间的映射，并调用 CoffSection 的 loadPage 将该页载入内存，loadPage 中会统计页长度，对于页长度不足 pageSzie 的，会在载入完指定长度后，页的剩余部分填 0.

当内存载入完毕后，就会 fork 一个 UThread，然后开始执行，而 UThread 的 run 方法中，是依次调用注册寄存器、恢复状态、Processor.run 操作。Processor.run 会依次执行内存中写入的二进制机器代码，执行程序。

2) Nachos 处理器和指令简述

Nachos 内部模拟了 MIPS 指令集，每条指令封装成一个 Mips 对象。然后指令从取值到执行的阶段封装为一个 Instruction 对象。处理器为 Processor 对象，用 byte 数

组模拟内存，int 数组模拟寄存器组，共 0x26 个（38 个），包括一些特殊功能的寄存器。执行指令的过程即使 PC 寄存器不断改变，然后取指、解析、执行的过程，通过捕捉异常（MipsException）来执行系统调用和其他中断。

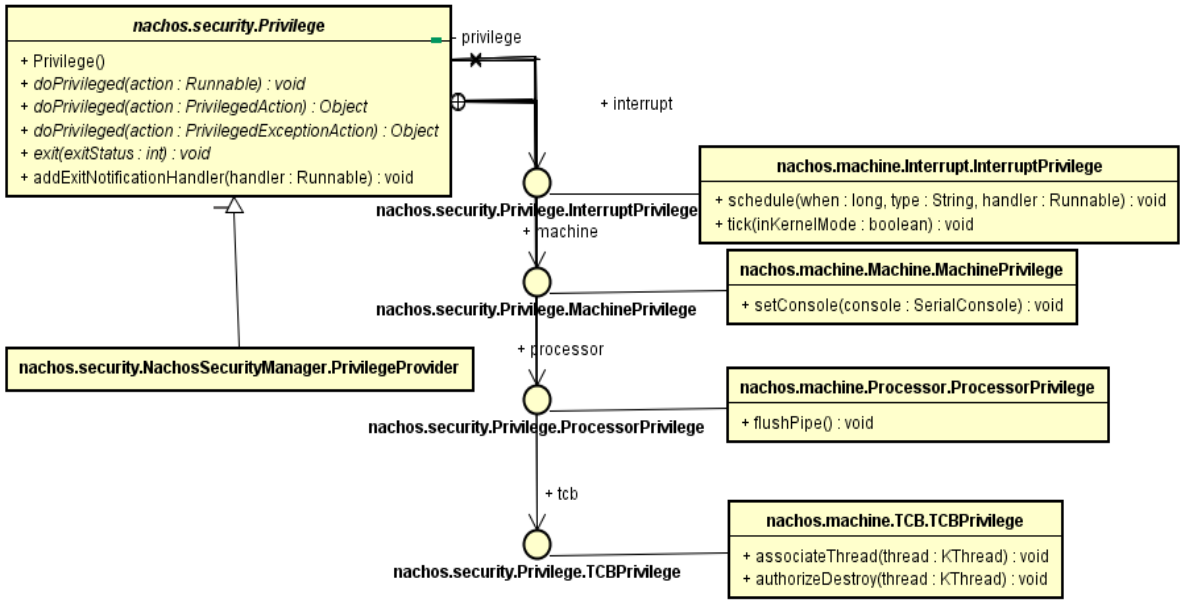
3) Nachos 系统调用

在 Nachos 中，36 号寄存器为 Cause，在每条指令执行结束后，该寄存器中会存入触发中断的类型。对于系统调用而言，会存入 0。在 V0 寄存器中存入系统调用的编号。

在 syscall.h 中，会看到不同系统调用的头文件，且不同系统调用都会在 UserProcess.handleSyscall 处理，传入的值为 A0-A3 寄存器中的值，作为系统调用的参数。至于传入这三个参数的原因，与函数调用约定有关，在一般情况下，系统会把函数的前 4 个参数的地址写入 A0-A3 寄存器，后面的参数存入堆栈中，函数返回后，将返回值或其地址存入 V0 寄存器。

7、 Nachos 安全管理简述

在 Nachos 的 NachosSecurityManager 中定义了 checkPermission 方法，该类中定义了一些受限的权限。



对于受限的权限，可通过 privilege 来调用实现。Privilege 的具体实现为 nachos.security.NachosSecurityManager.PrivilegeProvider，一个 Privilege 中定义了 TCB、Machine、Processor、Interrupt 的几个特权接口，并且也持有这些接口的引用。接口的实现在各个虚拟设备的类中。通过这些特权类，可以让其他设备或用户程序调用特权中对应的引用，实现相关特权操作。