目录

内容一：总体概述

Lab1 的主要内容主要涉及到 nachos 线程相关知识。将线程这一个抽象的概念具体到每一条代码。本次 lab 内容上可以分成两块，第一块和线程本身相关，包括线程的概念，具体的表示方法，线程的状态，状态转换等。第二块和线程的调度相关，包括线程的调度策略和具体的调度方法。

内容二：任务完成情况

调研

Linux PCB 是一个叫 task_struct 的结构体，这个结构体定义了进程相关的很多信息，包括 task 标识，进程的相关标志位，进程的资源等等。通过 task_struct，linux 对 task 进行维护，调度。由于 nachos 是一个比较简单的操作系统，许多实际运行中所考虑的问题，在这里被简化了。故而，nachos 的 PCB 就比较简单。

关于调度算法，linux 把 task 分成这几类：Stop_ask，Real_Time，Fair，Idle_Task，其优先级由高到低。Fair 进程使用完全公平调度算法 CFS，Real_Time 使用先入先出或者时间片轮转。而不同优先级的进程是可以抢占 CPU 运行的。

任务完成列表（Y/N）

	Exercise1	Exercise2	Exercise3	Exercise4	Exercise5	Challenge
第一部分	Y	Y	Y	Y	Y	Y

具体 Exercise 的完成情况

第一部分

Exercise 1　源代码阅读

整个系统的入口函数是 main()函数，通过这个函数会创建系统的第一个线程，main 线程。通过 main 线程，其他的线程被创建。Main 线程执行体主要的部分是一个 threadtest 函数，使用这个函数，可以测试操作系统中的线程相关的机制。

在 thread.h 和 thread.cc 中，写的是线程的相关代码，包括线程类的基本成员，以及线程相关的操作。这样对系统中的线程就可以具体描述了，就比较重要的是 fork()函数 Yield()函数 Finish()函数，可以用来描述线程的一生。

fork()函数的主要作用是创建新线程，其运行的机制主要是调用 StackAllocate()函数，通过这个函数来为线程分配空间，装填线程运行的代码体。而后将新的创建的线程加入就绪队列。加队列的过程关中断，表示是一个原子操作。

Yield()函数的主要作用是让出 CPU。具体的运行机制是先从就绪队列找寻队头，通过把

队头取出来，如果此时找到了一个队列头中的线程，那么把这个线程换上 CPU。如果没有找到这个线程，那么当前的线程就会继续运行下去。

Finish()函数主要是为线程结束做准备，一个线程需要结束的时候，主要是将它的状态设置为 Finish 状态，并把这个线程记录在变量 threadToBeDestroyed()中，之后进程 sleep()，即不会再被调用。设置成这个状态的线程，将会在每次线程切换的时候（运行 scheduler->Run()）被清理掉。

Exercise 2　扩展线程的数据结构

在进程的数据结构中，增加了两个 int 类型的成员变量，分别是 new_add_pid 和 new_add_uid。同时为这两个变量的维护添加了 get 和 set 方法。其中 uid 现在没有特殊的含义，仅仅作为测试该成员变量有没有被顺利添加到类中。pid 作为可以作为线程的唯一表示符号，用于区分不同线程。pid 的取值和 Exercise3 中全局管理机制中提供的表相关，就是线程在表中的下标。

Exercise 3　增加全局线程管理机制

增加了全局线程管理机制，模仿 XV6 操作系统的思路，建立了一个全局管理器 globalThreadManager，这个类主要维护一个 Thread* 类型的数组，数组大小为 MAXTHREADNUM，该量被定义 127，用于限制系统中存在的线程数量为 128。在数组中，每一个指针都指向一个线程对象。系统中每个线程对象被创建的时候，都会在构造函数中去申请加入该队列，全局管理器会遍历维护的数组，查看数组中有无空余的空间分配给该线程，若没有多空间，会让线程创建失败。

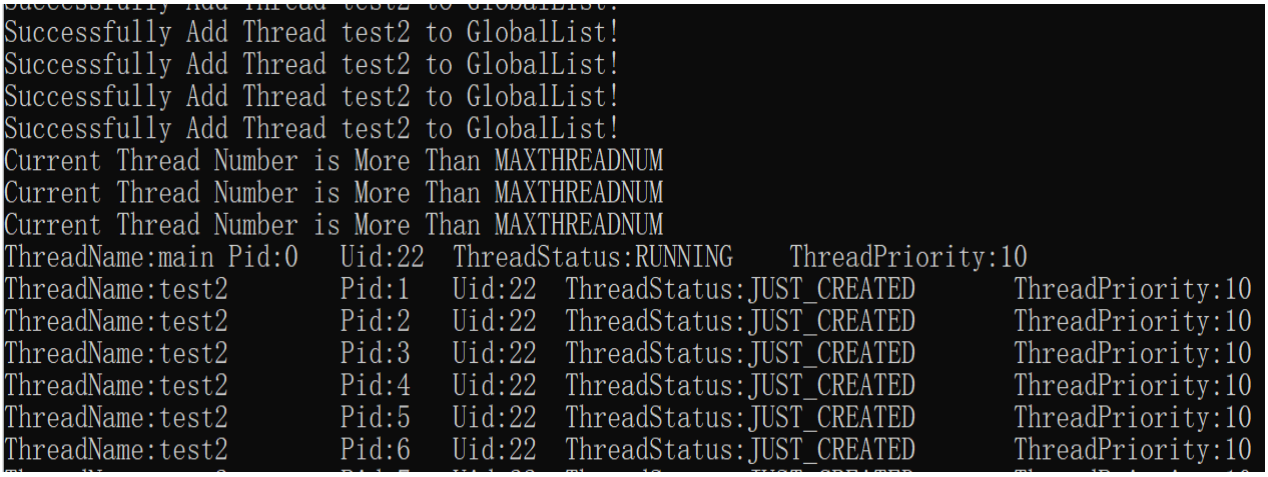
在全局管理器中设置 ShowListInfo()方法，这个方法的会遍历管理器维护的全局线程表，遍历该表，就能取出所有的系统中存在的线程。

下图为实验结果截图，在 Threadtest 中，运行脚本函数，创建了 130 个进程，可以发现，创建成功 127 个，失败了 3 个。为什么会比系统最大线程数少一个，是 main 线程占用了一个位置，所以有三个分配失败。调用 showListInfo 方法可以看见所有被创建的线程以及线程的信息。可见 Exercise2 中设置的成员变量都可以被正确设置和使用。

代码改动：

```
Thread.h    int new_add_pid;
Thread.h    int new_add_uid;
Thread.h    int getPid();
Thread.h    int getUid();//添加变量和维护变量的方法

Schedler.h    class GlobalThreadManager
                GlobalThreadManager();
               ~GlobalThreadManager();
                bool AddNewThreadtoList(Thread *thread);
                bool RemoveThreadFromList(Thread *thread);
                void ShowListInfo();
                Thread * GlobalThreadList [MAXTHREADNUM];
//添加全局管理器以及维护函数。
```



Exercise 4　源代码阅读

Scheduler.h 和 Scheduler.cc 主要是定义了系统中线程的调度器，该调度器维护了一个系统的就绪队列，同时定义了一系列的维护该就绪队列的方法。这一组文件中比较重要的是下面几个函数。

ReadyToRun()函数主要的作用是将进程添加到就绪队列，通过修改其中的函数可以改变每次插入线程在就绪队列中的顺序，当没有修改的时候，该对列就是按照插入时间排序。实现的是一个先入先出的调度算法。

FindNextToRun();找到队首的一个线程，并将它取出来。注意，不要用这个函数判断队空，否则可能丢掉队首的线程。

Timer.h 和 timer.cc 主要是定义系统的计时器。计时器的主要作用是模拟硬件时钟中断。系统并没有办法完全模仿硬件的时钟中断，系统在 interrupt 相关文件中提供了一系列变量记录系统时间，定义了时间片。然而，调用该中断主要还是需要线程主动触发中断函数，用这种方式来让系统时间向前走。后面需要时间片轮转算法的地方，本报告实现该机制时，主要是借助了中断相关的 onetich()函数。

Exercise 5　线程调度算法扩展

为线程添加 int 类型的成员变量 Priority，该成员变量代表线程的优先级，从 1-10 共 10 个优先级，其中优先级 1 表示最高优先级，优先级 10 表示最低优先级。线程调度算法为基于优先级的抢占式调度算法。该算法的要点有两部分，分别为基于优先级和抢占式。故而，在每次线程切换的时候需要调上来优先级最高的进程。这就需要对就绪队列按照优先级进行排序，排序的结果为优先级最高的线程在就绪队列的队首，这样每次就可以取出需要换上 CPU 的线程。实现抢占式的方法主要在 fork()函数中。Nachos 中除了 main 线程，其他线程都是在 main()下调用线程的成员函数 fork()实现线程的创建。在线程创建之初，如果线程的优先级比当前 CPU 执行线程的优先级高的话，会直接抢占 CPU，故而在 fork()函数中添加该机制，用于线程抢占式调度。

下图的测试函数中，分别创建了 t1,t2,t3 三个线程，其优先级分别是 9,8,2。Main 线程的优先级是默认的 10。每次执行完线程函数体后，线程都会让出 CPU 让调度机调度就绪队列优先级最高的线程上 CPU。可以看出，当高优先级线程来后，会抢占 CPU，在相互让权的过程中，最后必然是优先级高的两个进程相互交替执行。

代码改动：

Thread.h　int Priority;

Thread.cc　int getPriority()：

Thread.cc　int setPriority(int)：//添加优先级标志和维护函数

Thread.cc　void Fork() //添加抢占机制

Scheduler.cc Thread * FindNextToRun()//按照优先级找到下一个调度上 CPU 的进程

```
Successfully Add Thread main to GlobalList!
Successfully Add Thread t1 to GlobalList!
Successfully Add Thread t2 to GlobalList!
Successfully Add Thread t3 to GlobalList!
Come into Thread t1
*** thread 1 looped 0 times
Come into Thread t2
*** thread 2 looped 0 times
*** thread 1 looped 1 times
*** thread 2 looped 1 times
*** thread 1 looped 2 times
*** thread 2 looped 2 times
*** thread 1 looped 3 times
*** thread 2 looped 3 times
*** thread 1 looped 4 times
*** thread 2 looped 4 times
Successfully Remove Thread t1 from GlobalList!
Successfully Remove Thread t2 from GlobalList!
Come into Thread t3
*** thread 3 looped 0 times
ThreadName:main Pid:0   Uid:22  ThreadStatus:RUNNING    ThreadPriority:10
ThreadName:t3   Pid:3   Uid:22  ThreadStatus:READY      ThreadPriority:4
Come into Thread t0
*** thread 0 looped 0 times
*** thread 3 looped 1 times
*** thread 0 looped 1 times
*** thread 3 looped 2 times
*** thread 0 looped 2 times
*** thread 3 looped 3 times
*** thread 0 looped 3 times
*** thread 3 looped 4 times
*** thread 0 looped 4 times
Successfully Remove Thread t3 from GlobalList!
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 270, idle 0, system 270, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

***Challenge 线程调度算法扩展（时间片轮转法）**

为线程类添加 int 类型的变量 used_time_slice，记录线程已经运行的时间片个数。定义常量 SWITCHTIMESLICE 表示线程调度的基本单位，每次调度所间隔时间为 3 个系统时间片。在系统时钟运行时，每次执行系统时钟中断，当前线程的已用时间片加 1。一个时间片的时间在系统中被定义为常量 10。在下图实验结果中，可以看见，当系统中只存在一个线程 main 时，它会一直执行，直到有线程进入，且当前线程已经使用时间片大于 3 的时候，它会主动

让权。之后，每个线程会执行三个时间片，而后下 CPU。

```
ThreadName:t1   Pid:1   Uid:22  ThreadStatus:READY      ThreadPriority:10
ThreadName:t2   Pid:2   Uid:22  ThreadStatus:READY      ThreadPriority:10
ThreadName:t3   Pid:3   Uid:22  ThreadStatus:READY      ThreadPriority:10
Come into Thread t0
*** thread 0 looped 0 times     CUR_Thread:main  Thread_used_time_slice:40      CURSystem:40
Come into Thread t1
*** thread 1 looped 0 times     CUR_Thread:t1    Thread_used_time_slice:10      CURSystem:50
*** thread 1 looped 1 times     CUR_Thread:t1    Thread_used_time_slice:20      CURSystem:60
*** thread 1 looped 2 times     CUR_Thread:t1    Thread_used_time_slice:30      CURSystem:70
Come into Thread t2
*** thread 2 looped 0 times     CUR_Thread:t2    Thread_used_time_slice:10      CURSystem:80
*** thread 2 looped 1 times     CUR_Thread:t2    Thread_used_time_slice:20      CURSystem:90
*** thread 2 looped 2 times     CUR_Thread:t2    Thread_used_time_slice:30      CURSystem:100
Come into Thread t3
*** thread 3 looped 0 times     CUR_Thread:t3    Thread_used_time_slice:10      CURSystem:110
*** thread 3 looped 1 times     CUR_Thread:t3    Thread_used_time_slice:20      CURSystem:120
*** thread 3 looped 2 times     CUR_Thread:t3    Thread_used_time_slice:30      CURSystem:130
*** thread 0 looped 1 times     CUR_Thread:main  Thread_used_time_slice:10      CURSystem:140
*** thread 0 looped 2 times     CUR_Thread:main  Thread_used_time_slice:20      CURSystem:150
*** thread 0 looped 3 times     CUR_Thread:main  Thread_used_time_slice:30      CURSystem:160
*** thread 1 looped 3 times     CUR_Thread:t1    Thread_used_time_slice:10      CURSystem:170
*** thread 1 looped 4 times     CUR_Thread:t1    Thread_used_time_slice:20      CURSystem:180
Successfully Remove Thread t1 from GlobalList!
*** thread 2 looped 3 times     CUR_Thread:t2    Thread_used_time_slice:10      CURSystem:200
*** thread 2 looped 4 times     CUR_Thread:t2    Thread_used_time_slice:20      CURSystem:210
Successfully Remove Thread t2 from GlobalList!
*** thread 3 looped 3 times     CUR_Thread:t3    Thread_used_time_slice:10      CURSystem:230
*** thread 3 looped 4 times     CUR_Thread:t3    Thread_used_time_slice:20      CURSystem:240
Successfully Remove Thread t3 from GlobalList!
*** thread 0 looped 4 times     CUR_Thread:main  Thread_used_time_slice:10      CURSystem:260
```

代码改动：

Thread.h　int used_time_slice;//增加记录线程使用时间片的变量

Thread.c　thread();//初始化新增的时间片变量

Yeild();//改动函数，增加按照时间片调度的机制。

Interrupt.cc OneTick();//在记录系统时间的同时，对当前线程已经使用时间片变量进行维护。

内容三：遇到的困难以及解决方法

主要遇到的困难还是一开始上手项目的不熟悉感。需要花大量时间去理解注释，源代码，追踪系统运行过程。这其中看到许多汇编相关的代码，还需要一行一行去查相关寄存器的作用。在理解代码的时候，可能个人的理解关注重点不一样，导致理解视角不够全面，这个时候，和别人交流自己的看法，可以极大地帮助自己补充自己看代码过程中没有理解到的地方。在加代码到项目的过程中，也会遇到很多错误，原因很可能就是之前对代码的理解不够全面，多加断点，多尝试，多调试，亲自动手追踪，能帮助更好地理解系统是如何运行的。

内容四：收获及感想

关于技术方面的感想就是操作系统关于进程的运行和调度不再是那么抽象的概念了，它是一个个鲜活的实体。本质上就是内存和处理机相互配合完成任务的过程。对于进程线程的模拟，调度算法的模拟，让我对这部分的内容有了更加深刻具体的理解。所谓 talk is cheap,show me your code。亲手添加代码到系统中，并且运行起来，让我觉得很有满足感。

之前觉得可能源代码的阅读是一件比较枯燥繁琐没有趣味的过程，亲自尝试后，虽然需要花费巨大的精力，但是收获也很多。

内容五：对课程的意见和建议

暂时没有意见和建议。

内容六：参考文献

[1]《nachos 中文教程》
https://wenku.baidu.com/view/905197a9e209581b6bd97f19227916888486b90b.html
[2]《nachos 学习笔记（二）》https://blog.csdn.net/darord/article/details/83303765