

线程机制实习报告

姓名
日期

马少鹏
2018.3.13

学号

1500012893

目录

内容一：总体概述.....	3
内容二：任务完成情况.....	3
任务完成列表（Y/N）	3
具体 Exercise 的完成情况.....	3
内容三：收获及感想.....	6
内容四：对课程的意见和建议.....	6
内容五：参考文献.....	6

内容一：总体概述

1. 通过对 Linux 系统中进程控制块（PCB）相关资料的查阅，以及对 Linux 下 sched.h 源码文件的分析，了解了它的基本实现方式。通过阅读 Nachos 源代码，了解了 Nachos 中线程控制的实现方式。并对比了解了二者的异同。
2. 通过对 Nachos 中 threads 文件夹内相关代码的阅读，理解了 Nachos 现有的线程机制。
3. 在理解的基础上，自己为 Nachos 的线程管理机制增加了“用户 ID、线程 ID”两个数据成员，并为其增加了维护机制。
4. 在 Nachos 中增加了对线程数量的限制，并增加 TS（Threads Status）命令，显示系统中所有线程的信息和状态。

内容二：任务完成情况

任务完成列表（Y/N）

	Exercise1	Exercise2	Exercise3	Exercise4
Lab1	Y	Y	Y	Y

具体 Exercise 的完成情况

Exercise1 调研

Linux 中的进程管理

Linux 内核通过一个被称为进程描述符的 `task_struct` 结构体来管理进程，这个结构体包含了一个进程所需的所有信息，定义在 `include/linux/sched.h` 文件中。

`Task_struct` 是一个非常复杂的结构体，它的成员主要有：

1. 进程的状态。 `volatile long state`。
2. 进程的唯一标识 `pid` 和 线程组的领头线程的 `pid` 成员的值 `tgid`。
3. 进程的标记。 `unsigned int flags` 表示一些更加细节的状态信息。
4. 进程之间的亲属关系。 `struct task_struct *parent` 等。
5. 进程调度信息。如进程的优先级、调度策略等信息。
6. `Ptrace` 系统调用。
7. 时间数据成员。如进程当前的运行时间、进程的开始执行时间等。
8. 信号处理信息。
9. 文件系统信息。

Nachos 中的线程管理

Nachos 中的线程管理数据结构为类 `Thread`，定义在 `thread.h` 中，主要的成员有：

1. 运行时栈顶的指针 `stackTop`。运行时栈底指针 `stack`。

2. 机器状态信息 `machineState[MachineStateSize]`。
3. 线程状态信息 `status`，可取 `ready`、`running`、`blocked` 和 `just_created` 中的一种。
4. 线程名字 `name`。

二者的异同

Linux 进程管理和 Nachos 线程管理的相同之处是都使用一个数据结构记录了进程或线程的状态信息和机器状态的信息。

二者的不同之处：

1. 因为 Linux 是真实的操作系统，所以它的 PCB 要比 Nachos 的 TCB 复杂的多。如进程调度信息、信号处理信息、文件系统信息等成员信息，Nachos 的 TCB 中并没有记录。
2. Linux 的 PCB 中记录了进程之间的亲属关系，多个 PCB 实际上是由双向链表组织起来的。而 Nachos 的 TCB 中并没有记录其他线程的信息，即它们是零散的。

Exercise2 源代码阅读

`code/threads/main.cc`

首先调用 `Initialize(argc,argv)` 进行初始化，该函数首先进行参数的解析，参数有“-d”和“-rs”，然后初始化 `stats`，`interrupt`，`scheduler` 等信息。然后创建一个名字为“main”的线程，并把线程状态设置为 `running`。然后该线程根据不同的参数执行。运行结束后，调用 `Finish()`。值得注意的是，`Finish()` 并不直接杀死线程，而是设置一个全局变量 `threadToBeDestroyed` 为当前线程，然后 `sleep()`。阅读代码文件 `scheduler.cc` 可以发现，直到执行下一个线程的时候，才杀死上一个线程。

`code/threads/threadtest.cc`

主要注意 `SimpleThread()` 和 `ThreadTest1()` 这两个函数。前者循环五次，每次打印线程信息并挂起。后者首先创建一个 `SimpleThread` 的线程，然后自身执行函数 `SimpleThread()`。所以整个的运行过程就是父线程和子线程交替打印信息，然后挂起，直到循环结束。

`code/threads/thread.h` 和 `code/threads/thread.cc`

这两个代码文件中主要定义了 TCB 的数据结构和成员函数。通过阅读可以看到，线程状态一共有四种，通过成员函数 `setStatus()` 设置。上文中提到的 `Finish()` 并不直接杀死线程，也是通过阅读 `thread.cc` 代码文件得出的。此外，通过阅读代码，我理解了线程操作 `yield` 和 `sleep` 的区别：`yield` 是挂起线程，设置线程状态为 `ready`。而 `sleep` 把线程状态设置为 `blocked`，等待中断唤醒该线程，如果没有其他线程执行，则 CPU 空闲，等待一个中断发生。

Exercise3 扩展线程的数据结构

在 `Thread` 类中增加：

Private:

`Int user_id;`

`Int thread_id;`

Public:

`Int get_user_id(){ return user_id; }` //返回用户 ID

`Int get_thread_id(){ return thread_id; }` // 返回线程 ID

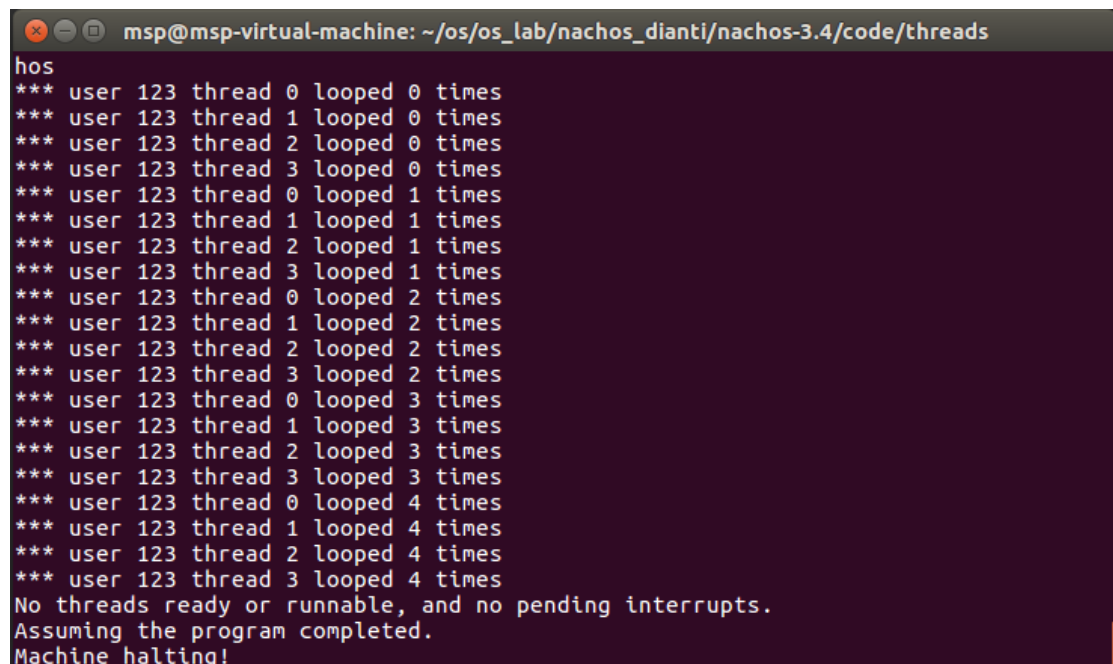
`Void set_user_id(int id){ user_id = id; }` // 设置用户 ID

对线程 ID 的维护，在 `system.cc` 中定义一个全局数组 `Thread_id[]`，考虑到之后对线程数

目的限制，数组大小定为 128。数组索引为线程 ID，Thread_id[i]=0，表示线程 ID i 并未被占用，Thread_id[i]=1 则表示线程 ID i 已经被占用。主线程 main 的线程 ID 默认为 0。每次创建新线程时，遍历该数组，找到第一个未被占用的线程 ID 分配给新线程，并把 Thread_id[i] 设置为 1，需要修改类 Thread 中的构造函数 Thread()，该函数的功能是新建线程时初始化一些成员变量，只需要在该函数中加上为线程分配线程 ID 的代码即可。

此外，需要在 Finish() 中，增加一句代码：Thread_id[currentThread->thread_id] = 0; 表示该线程执行结束时，该线程 ID 重新空闲，可以分配给其他线程。

运行结果：

A terminal window titled 'msp@msp-virtual-machine: ~/os/os_lab/nachos_dianti/nachos-3.4/code/threads' shows the output of a program. The output consists of 20 lines of status messages, each preceded by '***'. The messages follow a repeating pattern: 'user 123 thread 0 looped 0 times', 'user 123 thread 1 looped 0 times', 'user 123 thread 2 looped 0 times', 'user 123 thread 3 looped 0 times', then the same pattern with '1 times', '2 times', '3 times', and finally '4 times'. After the 20th line, the program prints 'No threads ready or runnable, and no pending interrupts.' and 'Assuming the program completed.' followed by 'Machine halting!' on the final line.

```
msp@msp-virtual-machine: ~/os/os_lab/nachos_dianti/nachos-3.4/code/threads
hos
*** user 123 thread 0 looped 0 times
*** user 123 thread 1 looped 0 times
*** user 123 thread 2 looped 0 times
*** user 123 thread 3 looped 0 times
*** user 123 thread 0 looped 1 times
*** user 123 thread 1 looped 1 times
*** user 123 thread 2 looped 1 times
*** user 123 thread 3 looped 1 times
*** user 123 thread 0 looped 2 times
*** user 123 thread 1 looped 2 times
*** user 123 thread 2 looped 2 times
*** user 123 thread 3 looped 2 times
*** user 123 thread 0 looped 3 times
*** user 123 thread 1 looped 3 times
*** user 123 thread 2 looped 3 times
*** user 123 thread 3 looped 3 times
*** user 123 thread 0 looped 4 times
*** user 123 thread 1 looped 4 times
*** user 123 thread 2 looped 4 times
*** user 123 thread 3 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

其中，默认设置用户 ID 为 123，主线程 main 线程 ID 为 0，在函数 ThreadTest1 中，创建了三个线程，线程 ID 分别为 1、2、3。

Exercise 4 增加全局线程管理机制

在 Nachos 中增加对线程数量的限制，最大数量为 128。

定义全局变量：

```
Int thread_count;
```

```
Const int MAX_THREAD_COUNT=128;
```

在类 Thread 的构造函数中，对线程数进行判断，若大于 128，则不进行线程 ID 的分配。在成员函数 Fork() 中进行同样的判断，如果线程数大于 128，直接 return，不为其申请空间。

增加 TS 功能。利用参数 “-ts” 实现。

定义数据结构：

```
Struct Whole_thread
```

```
{
```

```
    Int Thread_id;
```

```
    Thread* thread;
```

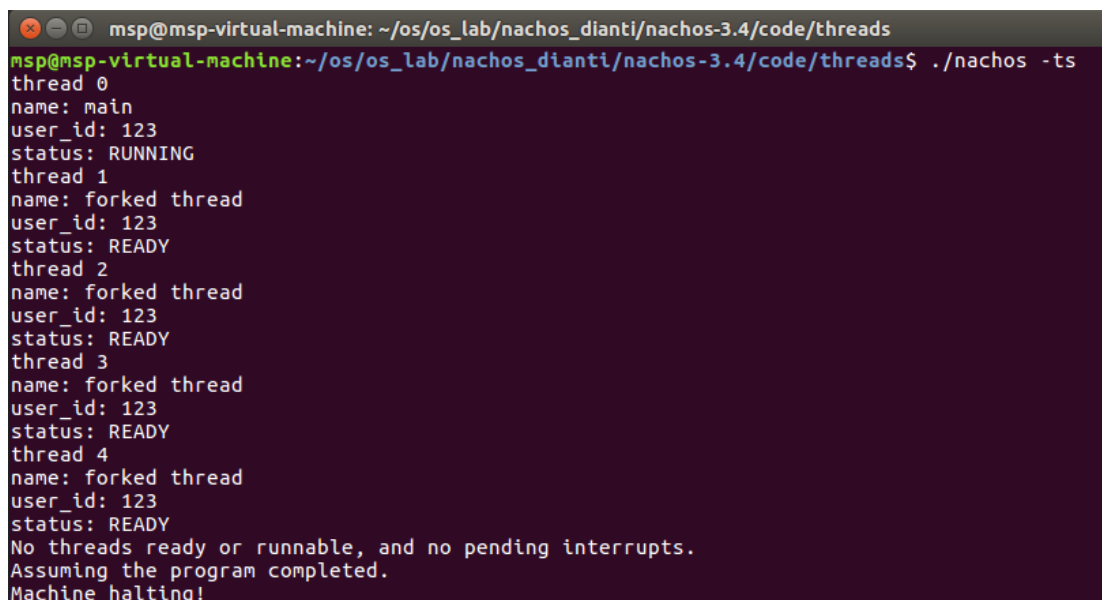
```
};
```

```
Whole_thread whole_thread[MAX_THREAD_COUNT];
```

用来记录 `thread_id` 的分配情况和系统中存在的线程的 `Thread` 数据结构的指针，即这是一个索引，根据这个索引便可以找到系统中存在的所有线程的信息并打印出来。

在 `main.cc` 中解析参数时，增加一条“-ts”，当参数为“-ts”时，`ThreadTest()`执行函数 `ThreadTest2()`。

`ThreadTest2()`为写在 `threadtest.cc` 中的函数，该函数功能是打印线程信息和状态。运行结果如下：



```
msh@msh-virtual-machine: ~/os/os_lab/nachos_dianti/nachos-3.4/code/threads
msh@msh-virtual-machine:~/os/os_lab/nachos_dianti/nachos-3.4/code/threads$ ./nachos -ts
thread 0
name: main
user_id: 123
status: RUNNING
thread 1
name: forked thread
user_id: 123
status: READY
thread 2
name: forked thread
user_id: 123
status: READY
thread 3
name: forked thread
user_id: 123
status: READY
thread 4
name: forked thread
user_id: 123
status: READY
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

内容三：收获及感想

通过第一次线程机制实习，我对 Linux 中进程管理有了更加深刻的理解，对 Nachos 的线程管理有了深刻的理解。

在实际动手写代码之前，我和我的讨论组队友进行了讨论，对 Nachos 线程管理机制发表了各自的看法。我觉得，经常和别人交流，非常有利于自身与他人的学习与进步。

内容四：对课程的意见和建议

希望可以在源代码的基础上增加一些文档，因为代码很多，经常发现代码调用了之前没有遇到的函数，找到这个函数的时候又发现它调用了其他的函数，找来找去很是麻烦。

内容五：参考文献

[1] Linux include/linux/sched.h 源代码