

操作系统原理第一次实验

姓名：李哲彦

学号：37220222203675

日期：2025年3月8日

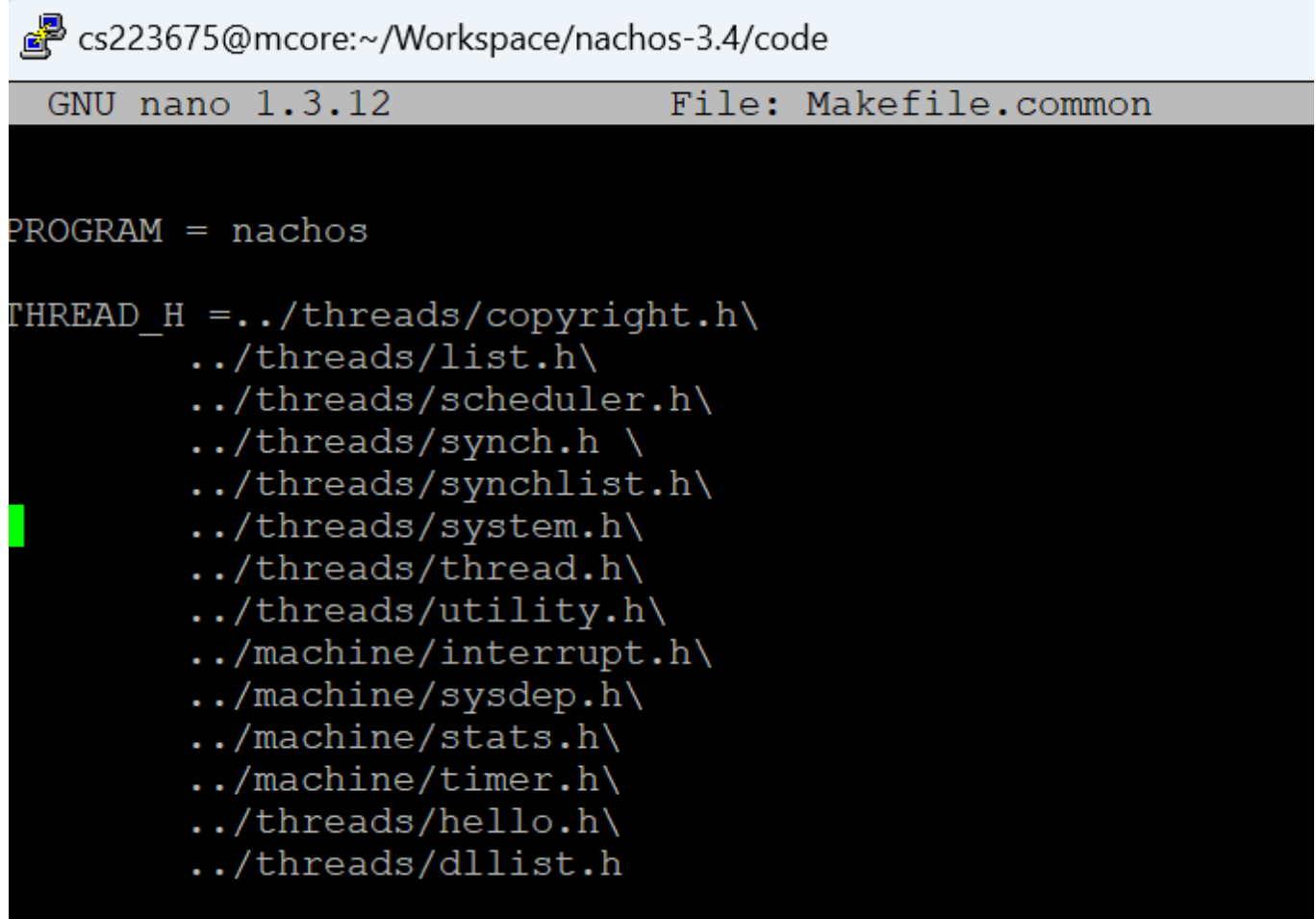
关键代码实现与运行结果

主要函数前添加 "Hello" 内容

文件 hello.c 与 hello.h 中的代码较为简单，在此不赘述。此处的难点是在于 Makefile 相关内容的配置与理解。

Makefile.common 这一文件包含了一些该项目中通用的标识定义以及共享配置文件，该文件被 Threads 目录下的 Makefile 文件所 "include"。因此，在 Makefile.common 中修改相关依赖即可修改 threads 模块的依赖关系。

在 common 文件中，可以看到对 Threads 模块的引用。首先是头文件的引用，在 THREAD_H 后添加 hello.h 的引用。



```
cs223675@mcore:~/Workspace/nachos-3.4/code
GNU nano 1.3.12 File: Makefile.common

PROGRAM = nachos

THREAD_H = ../threads/copyright.h\
          ../threads/list.h\
          ../threads/scheduler.h\
          ../threads/synch.h \
          ../threads/synchlist.h\
          ../threads/system.h\
          ../threads/thread.h\
          ../threads/utility.h\
          ../machine/interrupt.h\
          ../machine/sysdep.h\
          ../machine/stats.h\
          ../machine/timer.h\
          ../threads/hello.h\
          ../threads/dllist.h
```

然后对于 .cc 与 .o 的处理类似。



cs223675@mc core:~/Workspace/nachos-3.4/code

GNU nano 1.3.12

File: Makefile.common

```
THREAD_C = ../threads/main.cc\  
          ../threads/list.cc\  
          ../threads/scheduler.cc\  
          ../threads/synch.cc \  
          ../threads/synchlist.cc\  
          ../threads/system.cc\  
          ../threads/thread.cc\  
          ../threads/utility.cc\  
          ../threads/threadtest.cc\  
          ../machine/interrupt.cc\  
          ../machine/sysdep.cc\  
          ../machine/stats.cc\  
          ../machine/timer.cc\  
          ../threads/hello.c\  
          ../threads/dllist.cc\  
          ../threads/dllist-driver.cc
```

```
THREAD_O = main.o list.o scheduler.o synch.o synchlist.o system.o thread.o \  
          utility.o threadtest.o interrupt.o stats.o sysdep.o timer.o hello.o\  
          dllist.o dllist-driver.o
```

Hello 运行结果:

```
[cs223675@mc core threads]$ make  
g++ -m32 -g -Wall -Wshadow -traditional -I../threads -I../machine -DTHREADS -DHOST_i386 -DCHANGED -c ../threads/hello.c  
g++ -m32 main.o list.o scheduler.o synch.o synchlist.o system.o thread.o utility.o threadtest.o interrupt.o stats.o sysdep.o timer.o hello.o switch.o -o nachos  
[cs223675@mc core threads]$ ./nachos  
Hello Nachos!  
*** thread 0 looped 0 times  
*** thread 1 looped 0 times  
*** thread 0 looped 1 times  
*** thread 1 looped 1 times  
*** thread 0 looped 2 times  
*** thread 1 looped 2 times  
*** thread 0 looped 3 times  
*** thread 1 looped 3 times  
*** thread 0 looped 4 times  
*** thread 1 looped 4 times  
No threads ready or runnable, and no pending interrupts.  
Assuming the program completed.  
Machine halting!  
  
Ticks: total 130, idle 0, system 130, 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...  
[cs223675@mc core threads]$
```

双端链表的本地测试与实现

首先, 根据已给出的 dllist.h 中的函数定义以及文档中的要求, 写出 dllist.cc, 完成双端链表的具体实现。需要注意, dllist.cc 要包含 dllist.h 头文件, 并对其中的函数给出定义即可, 不需要重新定义 Dllist 类。(C++语言知识)

部分代码示例：

C++

```
#include "dllist.h"
DLLElement::DLLElement( void *itemPtr, int sortKey ){
    item = itemPtr;
    key = sortKey;
    next = NULL;
    prev = NULL;
} // initialize a list element
```

接下来是 dllist-driver.cc 的实现。

addNelements: 随机插入 N 个优先级随机的元素。在此我将元素定义为一个字符串，同时加以编号区别。

C++

```
void addNelements(int N, DLList *list){
    srand(time(0));
    for(int i = 0; i < N; i++){
        char* mychar = (char*)malloc(sizeof(char) * 8);
        sprintf(mychar, "Object%d", i);
        // printf("mychar: %s\n", mychar);
        list->SortedInsert((void*)mychar, rand());
        // printf("mychar: %s\n", mychar);
    }
    return ;
}
```

removeNelements: 从头开始删除 N 个元素，同时在屏幕上显示出来。

```

void removeNelements(int N, DLList *list){
    int cnt = 0;
    for(int i = 0; i < N; i++){
        int item_key = 0;
        void* now = list->Remove(&item_key);
        printf("item_key: %d\n", item_key);
        if(now == NULL){
            break;
        }
        else{
            printf("key: %d  str: %s\n", item_key, (char*)now);
        }
    }
    return ;
}

```

为了测试上述代码的正确性，我编写了 dllist-test.cc 用以本地测试。生成并删除10个元素。

```

int main(){
    DLList *list = new DLList();
    addNelements(10, list);
    removeNelements(10, list);
    return 0;
}

```

接下来涉及到 g++ 编译。由于只涉及三个 .cc 文件，直接使用普通编译命令即可：

```

PS F:\OpeSys\nachos-3.4\code\threads> g++ dllist-test.cc dllist-
driver.cc dllist.cc -o dllst-test.exe

```

可以看到上述代码在本地的 Windows 系统上运行正常。

```
PS F:\OpeSys\nachos-3.4\code\threads> ./dllst-test.exe
item_key: 1110
key: 1110 str: Object7
item_key: 3602
key: 3602 str: Object2
item_key: 7408
key: 7408 str: Object1
item_key: 10631
key: 10631 str: Object0
item_key: 14596
key: 14596 str: Object3
item_key: 17073
key: 17073 str: Object8
item_key: 18846
key: 18846 str: Object9
item_key: 23511
key: 23511 str: Object4
item_key: 25632
key: 25632 str: Object6
item_key: 27735
key: 27735 str: Object5
PS F:\OpeSys\nachos-3.4\code\threads> █
```

线程系统修改

阅读 main.cc 以及 threadtest.cc 文件可知，main.cc 会修改 testnum 变量，从而在 threadtest.cc 文件中会根据 testnum 执行对应的操作(可理解为 testcase)。

值得注意的是，main.cc中添加了 `ifdef THREADS` 这一语句，故需要在 makefile 文件中的 cflags 部分添加该标识符。

在 threadtest.cc 中，原本只包含了 Test1 相关的内容。类似编写 Test2 函数即可。

```

void
ThreadTest()
{
    switch (testnum) {
    case 1:
        printf("Entering ThreadTest1\n");
        ThreadTest1();
        break;
    case 2:
        printf("Entering ThreadTest2\n");
        ThreadTest2();
        break;
    default:
        printf("No test specified.\n");
        break;
    }
}

```

首先，模仿在不使用 yield 切换线程的情况下，两个线程会依次执行，且线程0先执行。

错误顺序1： 1写入5个，强制切换0，0写入5个，弹出10个，强制切换1，则1弹出时出错。

```

[cs223675@mc core threads]$ ./nachos -q 2
Hello Nachos! My id is 3675!
Thread Test! testnum: 2
Entering ThreadTest2
Newest!
Thread 1 write out!
Thread 0 write out!
item_key: 457167708
key: 457167708 str: Object4
item_key: 457167708
key: 457167708 str: Object4
item_key: 528355717
key: 528355717 str: Object2
item_key: 528355717
key: 528355717 str: Object2
item_key: 572159311
key: 572159311 str: Object1
item_key: 572159311
key: 572159311 str: Object1
item_key: 1899477202
key: 1899477202 str: Object0
item_key: 1899477202
key: 1899477202 str: Object0
item_key: 1939930220
key: 1939930220 str: Object3
item_key: 1939930220
key: 1939930220 str: Object3
Thread 0 remove out!
item_key: 0
Thread 1 remove out!
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 50, idle 0, system 50, 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...
[cs223675@mc core threads]$

```

错误情况2： 线程1在写入两个物体后，强制切换至线程0，可发现线程0在弹出时会错误弹出线程1写入的物体。（线程0总共只写入了一个 Object1，但是在第一次弹出时，弹出了两个 Object1，故两个 线程的Object1被错误混合在一起。）

```
[cs223675@mcore threads]$ ./nachos -q 2
Hello Nachos! My id is 3675!
Thread Test! testnum: 2
Entering ThreadTest2
Newest!
Thread 1 write 2!
Thread 0 write out!
item_key: 91197419
key: 91197419 str: Object1
item_key: 91197419
key: 91197419 str: Object1
item_key: 256627435
key: 256627435 str: Object4
item_key: 375689181
key: 375689181 str: Object2
item_key: 944657991
key: 944657991 str: Object0
Thread 0 remove out!
Thread 1 write 3!
item_key: 91197419
key: 91197419 str: Object1
item_key: 375689181
key: 375689181 str: Object2
item_key: 944657991
key: 944657991 str: Object0
```

错误情况3： 线程0先写入，然后切换至线程1，线程1再进行 Remove 操作。可以看到由于优先级随机，二者的部分 Object 混入到一起，线程1错误的删除了线程0写入的内容。

```
[cs223675@mcore threads]$ ./nachos -q 2
Hello Nachos! My id is 3675!
Thread Test! testnum: 2
Entering ThreadTest2
Newest!
Thread 0 write out!
Thread 1 write 5!
item_key: 270249794
key: 270249794 str: Object0
item_key: 270249794
key: 270249794 str: Object0
item_key: 1270712601
key: 1270712601 str: Object4
item_key: 1270712601
key: 1270712601 str: Object4
item_key: 1708594754
key: 1708594754 str: Object1
Thread 1 remove 5!
item_key: 1708594754
key: 1708594754 str: Object1
item_key: 1808475856
key: 1808475856 str: Object2
item_key: 1808475856
key: 1808475856 str: Object2
item_key: 2053993992
key: 2053993992 str: Object3
item_key: 2053993992
key: 2053993992 str: Object3
Thread 0 remove out!
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 40, idle 0, system 40, 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
```

错误情况4： 线程 1 先写入5个物体，然后切换至线程0，写入两个物体后切回至线程1，线程1再删除10个物体。可以看到线程0的两个物体(0, 1) 也会被错误地删除。

```

[cs223675@mc core threads]$ ./nachos -q 2
Hello Nachos! My id is 3675!
Thread Test! testnum: 2
Entering ThreadTest2
Newest!
Thread 1 write 5!
Thread 0 write 2!
item_key: 75549284
key: 75549284 str: Object2
item_key: 547732615
key: 547732615 str: Object4
item_key: 1917402534
key: 1917402534 str: Object1
item_key: 1917402534
key: 1917402534 str: Object1
item_key: 1971615275
key: 1971615275 str: Object3
item_key: 1978748956
key: 1978748956 str: Object0
item_key: 1978748956
key: 1978748956 str: Object0
item_key: 0
Thread 1 remove 10!
item_key: 0
Thread 0 remove out!
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 60, idle 0, system 60, 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...

```

遇到的问题以及收获

1、make 的相关问题：在修改 Nachos 的代码时，发现会出现 make 指令无法识别到代码已经被修改，或者重新 "make" 之后，运行结果仍然是老版本代码等情况。在尝试使用 make clean 时，发现会提示没有可生成的 clean 对象。经过查阅，在 Makefile.common 中手动添加下列内容后解决问题。即调用 make clean 命令时，执行指定的 rm 指令，删掉所有 .o 对象文件等中间文件。

```

.PHONY: clean
clean:
    # 删除所有编译产物：对象文件、依赖文件、可执行文件
    rm -f *.o *.d *.a

```

2、关于 THREADS 标志符的设定，以启用 ThreadTest 部分的内容：从 mian.cc 代码中发现，ThreadTest 部分的内容被包括在 `#ifdef THREADS` 内。查询相关资料得知，需在 Makefile 文件中的 Cflags 字段增添该标识。

收获：通过本次实验，我了解了 Nachos 的基本结构以及 Threads 部分代码的基本框架与使用方法。并且，在 Threadstest 这一模块的实验中，我实际构造了4个会导致线程间数据混乱的方法，深刻理解到如果管理不当，线程间数据混乱会造成严重的不良后果。例如，在我构造的两个线程同时对同一个双向链表进行操作的情况下，就会出现一个线程写入的内容错误地被另一个线程弹出的情况。

这也警示我，在日后涉及到多线程编程时，要时刻注意线程间的数据安全。