

课程名称：操作系统实践	年级：2023级	上机实践成绩：
指导教师：张民	姓名：李彤	
上机实践名称：修改alarm-priority	学号：10235101500	上机实践日期：2024.10.28
上机实践编号：	组号：	上机实践时间：

实验三 241028-修改alarm-priority

此处省略运行pintos

实验准备阶段——运行测试用例

如在build环境下make之后，运行 **alarm-multiple** 程序，输出正常

```
Boot complete.
Executing 'alarm-multiple':
(alarm-multiple) begin
(alarm-multiple) Creating 5 threads to sleep 7 times each.
(alarm-multiple) Thread 0 sleeps 10 ticks each time,
(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.
(alarm-multiple) If successful, product of iteration count and
(alarm-multiple) sleep duration will appear in nondescending order.
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
```

实验思路

要想将进程调用由队列的FIFO顺序改为按照优先级顺序调用，主要有两种实现方法：

- **顺序插入**：在放入的时候，就按照优先级的大小排好序，取出时只需要弹出list头部的进程即可；
- **尾部插入**：在放入时直接插入list的尾部，在取出时，遍历list搜索优先级最高的进程弹出。

在最开始的实验中，我使用的是 **顺序插入** 法，使得线程在进入队列时就能按照优先级排序，取出时直接从头部取出。

实验步骤

1. 解读list_insert_ordered

首先在list.c文件中找到函数list_insert_ordered，并在这个基础上实现顺序插入。其主要思想是通过for循环将待插入进程elem与进程list中的每一个进程进行优先级比较，找到正确的位置后再跳出循环，并调用list_insert

函数将其插入。

```
void
list_insert_ordered (struct list *list, struct list_elem *elem,
                    list_less_func *less, void *aux)
{
    struct list_elem *e;

    ASSERT (list != NULL);
    ASSERT (elem != NULL);
    ASSERT (less != NULL);

    // e 遍历list中的所有进程，与elem的优先级进行比较
    for (e = list_begin (list); e != list_end (list); e = list_next (e))
        if (less (elem, e, aux))
            // prio_cmp_func返回true的时候，说明elem的优先级比e大，所以此时可以把elem插入进程list中（顺序正确）
            break;
    // 循环结束就说明已经找到了正确的位置，把进程插入
    return list_insert (e, elem);
}
```

2. 实现函数prio_cmp_func

因为要基于优先级比较进行排列插入，所以这里还要实现一个能比较不同thread优先级的函数prio_cmp_func。

注：不要忘记把prio_cmp_func的函数声明添加到list.h文件中（一开始确实忘了。。。）

```
/*compare two threads priority, return true when i is prior to o*/
bool
prio_cmp_func(struct list_elem *elem_i, struct list_elem *elem_o, void *aux)
{
    struct thread *thread_i = list_entry(elem_i, struct thread, elem);
    struct thread *thread_o = list_entry(elem_o, struct thread, elem);

    return thread_i->priority > thread_o->priority;
}

void list_insert_ordered (struct list *, struct list_elem *,
                        list_less_func *, void *aux);

bool prio_cmp_func(struct list_elem *elem_i,
                  struct list_elem *elem_o, void *aux);
```

3. list_insert_ordered函数替换list_push_back函数

将比较函数prio_cmp_func写好之后，就可以使用list_insert_ordered函数实现顺序插入了，其实就是把thread.c文件中的尾插函数list_push_back都替换成函数list_insert_ordered。这里要改的函数分别为**thread_unblock函数、thread_yield函数和init_thread函数**（Ctrl+F输入list_push_back可以快速检索出所有要更改的函数）。下面只给出了thread_unblock函数中的修改。

需要注意的是，**list_push_back和list_insert_ordered形参是不一样的**，所以不能只改一个函数名。在这次实验中list_insert_ordered的**第一个参数是线程list，第二个参数是待插入线程，第三个参数则是比较函数**

`prio_cmp_func`，第四个参数我还没用到，就设置为空。

```
void thread_unblock(struct thread *t)
{
    enum intr_level old_level;

    ASSERT(is_thread(t));

    old_level = intr_disable();
    ASSERT(t->status == THREAD_BLOCKED);
    // list_push_back(&ready_list, &t->elem);
    list_insert_ordered(&ready_list, &t->elem, prio_cmp_func, NULL);
    t->status = THREAD_READY;
    intr_set_level(old_level);
}
```

4. make

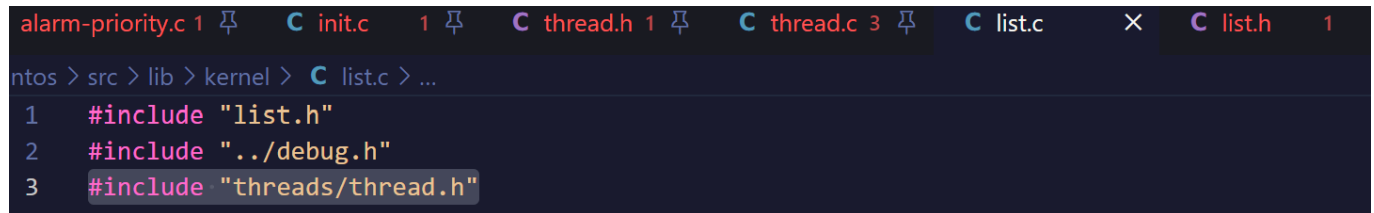
BUT，出现了ERROR。。。来看看怎么回事

```
问题 10 调试控制台 输出 终端 端口
-I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers -MMD -MF lib/kernel/list.d
In file included from ../lib/kernel/list.h:86:0,
      from ../lib/kernel/list.c:1:
../lib/kernel/list.c: In function 'prio_cmp_func':
../lib/stddef.h:5:55: error: dereferencing pointer to incomplete type 'struct thread'
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *) 0)->MEMBER)
                                         ^
../lib/kernel/list.h:110:24: note: in expansion of macro 'offsetof'
- offsetof (STRUCT, MEMBER.next)))
```

粗略看一眼，大概是 结构体 `thread` 的引用有点问题，然后又在 `thread/thread.h` 文件中找到了 `thread` 结构体的定义。

```
C alarm-priority.c 1  C init.c 1  C thread.h 1  C thread.c 3  C list.c 4  C list.h
pintos > src > threads > C thread.h > thread
1  #ifndef THREADS_THREAD_H
77 > /** The 'elem' member has a dual purpose. It can be an
83 struct thread
84 {
85     /* Owned by thread.c. */
86     tid_t tid;                /**< Thread identifier. */
87     enum thread_status status; /**< Thread state. */
88     char name[16];            /**< Name (for debugging purposes). */
89     uint8_t *stack;           /**< Saved stack pointer. */
90     int priority;              /**< Priority. */
91     struct list_elem allelem;  /**< List element for all threads list. */
92
93     /* Shared between thread.c and synch.c. */
94     struct list_elem elem;     /**< List element. */
```

再看list.c引用的头文件，发现确实 **没有包含相关文件**，然后就把thread.h文件加上了



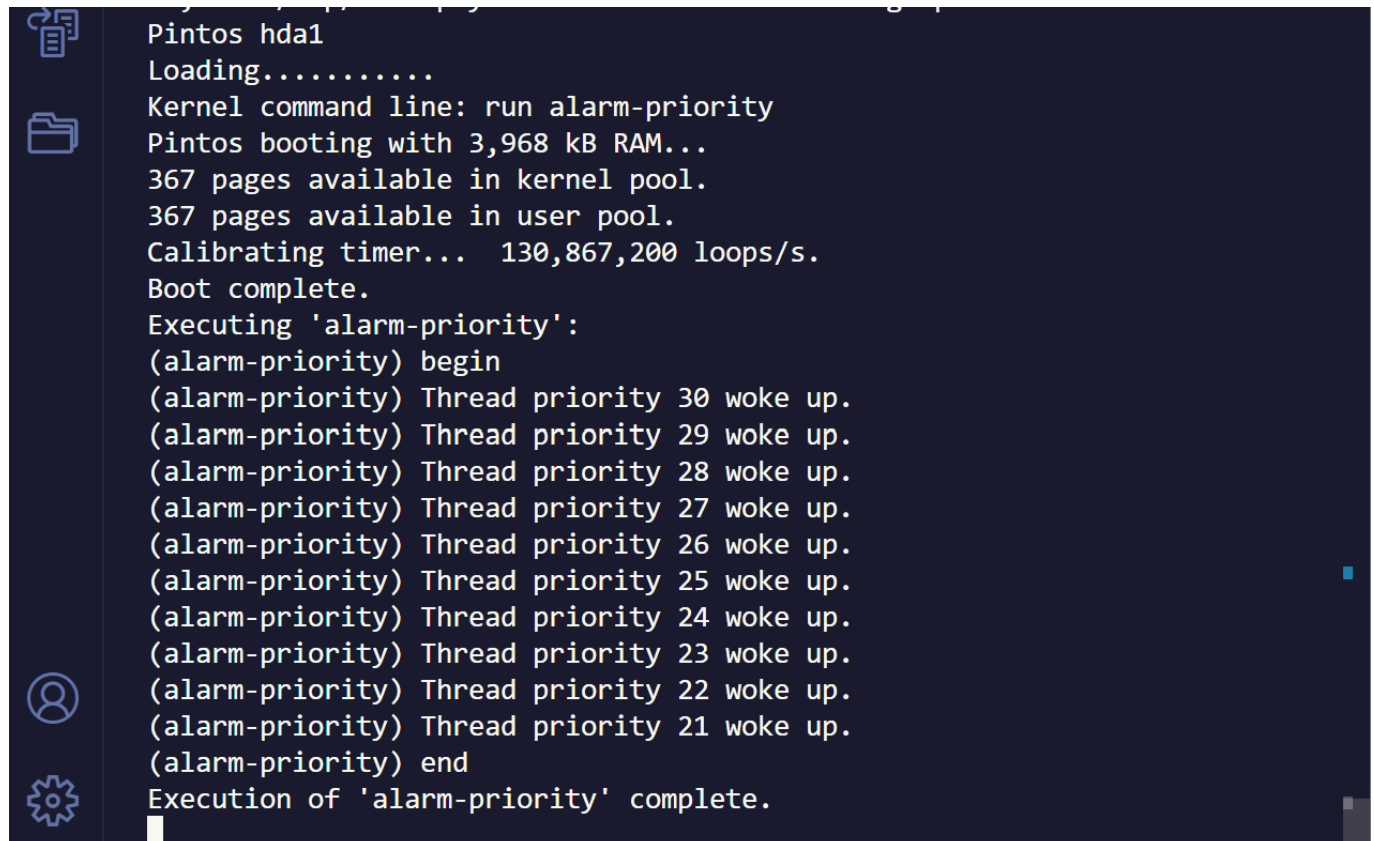
```

alarm-priority.c 1  init.c 1  thread.h 1  thread.c 3  list.c 1  list.h 1
ntos > src > lib > kernel > list.c > ...
1  #include "list.h"
2  #include "../debug.h"
3  #include "threads/thread.h"

```

5. 再次执行make指令

这一次没有报错，于是接着执行 **pintos -- run alarm-priority** 指令，最终打印出的线程优先级由高到低（代表实现了按照优先级大小调度）

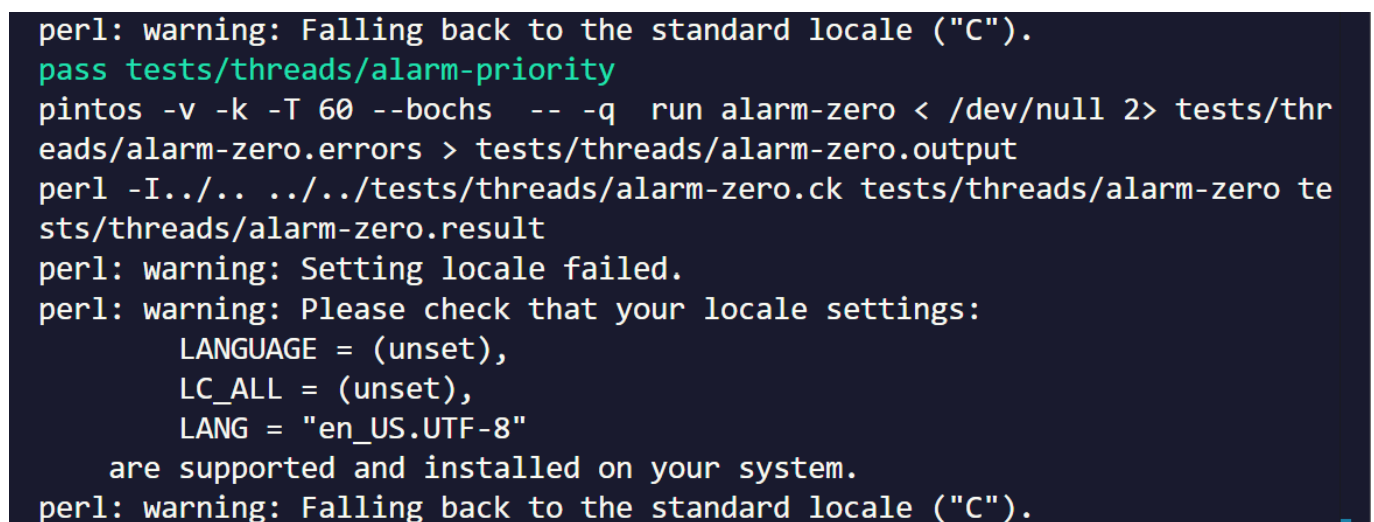


```

Pintos hda1
Loading.....
Kernel command line: run alarm-priority
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 130,867,200 loops/s.
Boot complete.
Executing 'alarm-priority':
(alarm-priority) begin
(alarm-priority) Thread priority 30 woke up.
(alarm-priority) Thread priority 29 woke up.
(alarm-priority) Thread priority 28 woke up.
(alarm-priority) Thread priority 27 woke up.
(alarm-priority) Thread priority 26 woke up.
(alarm-priority) Thread priority 25 woke up.
(alarm-priority) Thread priority 24 woke up.
(alarm-priority) Thread priority 23 woke up.
(alarm-priority) Thread priority 22 woke up.
(alarm-priority) Thread priority 21 woke up.
(alarm-priority) end
Execution of 'alarm-priority' complete.

```

make check



```

perl: warning: Falling back to the standard locale ("C").
pass tests/threads/alarm-priority
pintos -v -k -T 60 --bochs -- -q run alarm-zero < /dev/null 2> tests/threads/alarm-zero.errors > tests/threads/alarm-zero.output
perl -I../.. ../tests/threads/alarm-zero.ck tests/threads/alarm-zero tests/threads/alarm-zero.result
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
    LANGUAGE = (unset),
    LC_ALL = (unset),
    LANG = "en_US.UTF-8"
are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").

```

也pass了。

实验心得

通过这次实验，我更加熟悉 thread 源码，并且对于 pintos 中线程创建及调度方法有更深入的理解，在追溯与 thread 相关的函数的过程中也了解到了很多其他的函数的实现和功能，测试期间通过使用GDB、printf()进行调试，也让我对各种调试操作更加熟悉。