实验报告：Pintos 代码修改——线程优先级调度机制

课程名称：操作系统实践	**年级**：大二	**上机实践成绩**：
指导教师：张民	**姓名**：丁乙	
上机实践名称：Pintos 代码修改——线程优先级调度机制	**学号**：10235101489	**上机实践日期**：2024.10.28
上机实践编号：实验三	**组号**：无	**上机实践时间**：1.5h

一、目的

本次上机实践的主要目的是掌握 Pintos 操作系统的基本代码结构，通过测试驱动开发（TDD）的方法，修改 Pintos 中的线程调度机制，从原有的 FIFO（先进先出）调度改为基于优先级的调度，以通过 `alarm-priority` 的测试用例，并深入理解操作系统中的线程调度原理。

二、内容与设计思想

上机实践内容

环境搭建：配置 *Pintos* 操作系统环境，确保能够编译和运行 *Pintos* 代码。

代码分析：通读 *Pintos* 源代码，特别是与线程调度相关的部分，如 `thread.c` 和 `init.c` 等。

测试准备：运行 `make` 编译 *Pintos* 代码，并进入 `build` 目录，准备运行测试用例。

测试运行：运行 `pintos --run alarm-multiple` 等测试用例，观察测试结果，并分析未能通过测试的原因。

代码修改：修改线程调度机制，将 *FIFO* 调度改为优先级调度，确保线程按照优先级高低进行调度。

测试验证：重新编译并运行 `make check`，验证 `alarm-priority` 测试用例是否通过。

算法设计思想与算法实现步骤

分析现有调度机制：通过阅读和调试 *Pintos* 源代码，了解现有的 *FIFO* 调度机制。

设计优先级调度机制：设计并实现基于优先级的调度机制，包括线程插入就绪队列时的排序和选择下一个执行线程时的逻辑。

修改代码：在 `thread.c` 等文件中修改线程插入和选择逻辑，实现优先级调度。

编写比较函数：编写用于比较线程优先级的函数，以便在插入就绪队列时进行排序。

测试与调试：通过运行测试用例和调试工具，验证修改后的调度机制是否正确。

三、使用环境

本次上机实践所使用的平台是 Linux 操作系统，相关软件包括 Pintos 操作系统源代码、GCC 编译器、GDB 调试器等。

四、实验过程

环境搭建：按照 *PPT* 中的步骤，配置好 *Pintos* 操作系统环境，确保能够顺利编译和运行 *Pintos* 代码。

代码分析：通过阅读和调试 *Pintos* 源代码，特别是与线程调度相关的部分，了解了现有的 *FIFO* 调度机制。

测试准备：运行 `make` 编译 *Pintos* 代码，并进入 `build` 目录，准备运行测试用例。

测试运行：运行 `pintos --run alarm-multiple` 等测试用例，发现 `alarm-priority` 测试用例未能通过。

代码修改：

在 `thread.c` 文件中，将线程插入就绪队列的 `list_push_back` 函数替换为按照优先级插入的函数。

编写比较函数 `list_less_func`，用于比较线程的优先级。

修改选择下一个执行线程的逻辑，确保选择优先级最高的线程执行。

测试验证：重新编译并运行 `make check`，验证 `alarm-priority` 测试用例是否通过。经过多次修改和测试，最终成功通过该测试用例。

在实验过程中，遇到了以下问题：

对 *Pintos* 源代码的理解不够深入，导致在修改代码时遇到困难。

编写的比较函数存在逻辑错误，导致线程插入就绪队列时的排序不正确。

在调试过程中，未能准确定位问题所在，导致调试效率低下。

针对上述问题，采取了以下解决方法：

多次阅读 *PPT* 和 *Pintos* 源代码，加深对代码结构的理解。

重新编写比较函数，并进行多次测试，确保排序正确。

使用 *GDB* 调试器进行调试，逐步定位问题所在，并进行修改。

五、总结

通过本次上机实践，我深入了解了 Pintos 操作系统的基本代码结构和线程调度机制。通过测试驱动开发（TDD）的方法，我成功地将 Pintos 中的线程调度机制从 FIFO 改为基于优先级的调度，并通过了 `alarm-priority` 的测试用例。在实践过程中，我遇到了一些问题，但通过不断学习和调试，最终成功解决了这些问题。

通过本次实践，我不仅掌握了 Pintos 操作系统的基本使用方法，还提高了代码阅读和调试能力。同时，我也深刻认识到了操作系统中线程调度机制的重要性，

六、附录

```
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-z
ero.output
perl -I../.. ../../tests/threads/alarm-zero.ck tests/threads/alarm-zero tests/threads/alarm-zero.result
```

```
Kernel command line: -q run alarm-priority
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  157,081,600 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.
Timer: 531 ticks
Thread: 0 idle ticks, 531 kernel ticks, 0 user ticks
Console: 839 characters output
Keyboard: 0 keys pressed
Powering off...
root@3621b2df82b0:~/pintos/src/threads/build#
```

```
if (cur != idle_thread)
list_insert_ordered(&ready_list, &cur->elem, prio_cmp_fun,NULL);
//list_push_back (&ready_list, &cur->elem);
cur->status = THREAD_READY;
old_level = intr_disable ();
ASSERT (t->status == THREAD_BLOCKED);
list_insert_ordered(&ready_list,&t->elem,prio_cmp_fun,NULL);
//list_push_back (&ready_list, &t->elem);
t->status = THREAD_READY;
```

```
old_level = intr_disable ();
//list_push_back (&all_list, &t->allelem);
list_insert_ordered(&all_list, &t->allelem, prio_cmp_fun,NULL);
intr_set_level (old_level);
```

修改插入逻辑，插入改为有序插入，使得整个链表元素按照优先级高低降序排列，选择是队头元素——即当前链表队伍中优先级最高的元素。所以实现了按照优先级调度

```
uint32_t thread_stack_ofs = offsetof (struct thread, stack);
bool prio_cmp_fun(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;
}
```

优先级比较函数辅助实现优先级插入