中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

实验课程:	操作系统原理实验
实验名称:	内核线程
专业名称:	计算机科学与技术
学生姓名:	数据删除
学生学号:	数据删除
实验成绩:	
报告时间:	2025 年 4 月 19 日

1. 实验要求

Assignment 1 printf 的实现

学习可变参数机制，然后实现 printf，你可以在材料中的 printf 上进行改进，或者从头开始实现自己的 printf 函数。结果截图并说说你是怎么做的。

Assignment 2 线程的实现

自行设计 PCB，可以添加更多的属性，如优先级等，然后根据你的 PCB 来实现线程，演示执行结果。

Assignment 3 线程调度切换的秘密

操作系统的线程能够并发执行的秘密在于我们需要中断线程的执行，保存当前线程的状态，然后调度下一个线程上处理机，最后使被调度上处理机的线程从之前被中断点处恢复执行。现在，同学们可以亲手揭开这个秘密。

编写若干个线程函数，使用 gdb 跟踪 c_time_interrupt_handler、asm_switch_thread 等函数，观察线程切换前后栈、寄存器、PC 等变化，结合 gdb、材料中“线程的调度”的内容来跟踪并说明下面两个过程。

- 一个新创建的线程是如何被调度然后开始执行的。
- 一个正在执行的线程是如何被中断然后被换下处理器的，以及换上处理机后又是如何从被中断点开始执行的。

通过上面这个练习，同学们应该能够进一步理解操作系统是如何实现线程的并发执行的。

Assignment 4 调度算法的实现

在材料中，我们已经学习了如何使用时间片轮转算法来实现线程调度。但线程调度算法不止一种，例如

- 先来先服务。
- 最短作业（进程）优先。

➢ 响应比最高者优先算法。

➢ 优先级调度算法。

➢ 多级反馈队列调度算法。

此外，我们的调度算法还可以是抢占式的。

现在，同学们需要将线程调度算法修改为上面提到的算法或者是同学们自己设计的算法。然后，同学们需要自行编写测试样例来呈现你的算法实现的正确性和基本逻辑。最后，将结果截图并说说你是怎么做的。

2. 实验过程

Assignment 1 printf 的实现

printf 的基本框架流程为:

1. 函数接收一个格式化字符串 format 和一个可变参数列表。使用 va_list 类型来处理可变参数，并通过 va_start 宏初始化参数列表。
2. 遍历格式化字符串 format。对于每个字符，判断其是否为格式控制符 %
3. 格式控制符处理
    a) 对于整数输出（%d 和 %x），分别调用 write_int 和 write_hex 函数，将整数转换为字符串并输出。
    b) 对于字符串输出（%s），直接获取字符串指针，然后逐字符调用 putchar 函数输出。
    c) 对于字符输出（%c）和输出百分号本身（%%），均直接调用 putchar 输出。
4. 转义字符处理：特别处理了换行符\n 和制表符\t。通过调用 getCursor 获取当前光标位置，并根据换行和制表符的规则移动光标位置，并通过 moveCursor 移动光标。同时，如果光标超出屏幕范围，则调用 rollUp 实现向上滚动屏幕的功能。

write_int 函数采用递归而非字符缓冲区的方式处理多位数输出，write_hex 函数则支持十六进制输出。

Assignment 2 线程的实现

通过 ProgramManager 类管理线程，负责线程的创建、调度和销毁，主要由以下几部分组成：

1. 其内部成员有：
    a) allPrograms：存储所有状态的线程/进程
    b) readyPrograms：存储就绪状态的线程
    c) running：指向当前执行的线程
2. 创建线程方法，流程为：
    a) 分配空闲 PCB
    b) 初始化线程相关信息（名称、优先级等）
    c) 设置线程执行函数和参数
    d) 设置线程栈空间
    e) 将线程加入就绪队列
    f) 返回线程 ID（pid）
3. 内部的释放线程方法，流程为：
    a) 通过线程指针反推线程在线程池中的下标位置
    b) 将对应位置的标志位置为空闲
4. 线程调度方法，使用时间片流转，每次调度时如果当前线程还在运行，则将其放回队列末尾，重新从队列头取出一个新的线程执行。

使用 8259A 时钟中断触发线程调度，达到线程切换的效果。

测试运行：创建两个线程 first_thread 和 second_thread

Assignment 3 线程调度切换的秘密

对 Assignment 2 的代码进行修改，将两个线程改为了易于追踪中断前位置的代码。

原理上，函数调用之前会将下一条指令的地址压入栈中，这样函数返回后就能找到调用前的地址。通

过交换两个线程的栈，就可以在 asm_switch_thread 结束后来到另一个线程的地址空间。

对 asm_switch_thread、schedule_time_interrupt_handler、programManager.schedule 分别下断，观察其执行前后的行为。

Assignment 4 调度算法的实现

基于 RR 实现了四个调度算法：FCFS、SJF、PR、MLFQ，实现均为非抢占式。

FCFS 只需要将调度函数的逻辑改为只有线程结束才切换任务；SJF 和 PR 的实现几乎完全一致，因为没有任务量估算算法的情况下，执行时间和优先级在非抢占式调度下没什么区别。MLFQ 实现了 5 层队列，为每层队列赋了不同的时间片权值。

3. 关键代码

Assignment 1 printf 的实现

两个输出整数的函数：

```
void STDIO::write_int(int x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x >= 10) write_int(x / 10), x %= 10;
    putchar(char(x + '0'));
}

void STDIO::write_hex(int x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x >= 16) write_hex(x / 16), x %= 16;
    if (x < 10) putchar(char(x + '0'));
    else putchar(char(x - 10 + 'A'));
}
```

printf 实现：

```
void printf(const char* format, ...) {
    va_list args;
    va_start(args, format);
    for (const char* p = format; *p != '\0'; ++p) {
        if (*p == '%') {
            ++p;
            if (*p == 'd') {
               int num = va_arg(args, int);
               stdio.write_int(num);
            }
            else if (*p == 'x') {
               int num = va_arg(args, int);
               stdio.write_hex(num);
            }
            else if (*p == 's') {
               const char* str = va_arg(args, const char*);
               while (*str != '\0') {
                   stdio.putchar(*str++);
               }
            }
            else if (*p == 'c') {
               char ch = (char)va_arg(args, int);
               stdio.putchar(ch);
            }
            else if (*p == '%') {
               stdio.putchar('%');
            }
        }
        else if (*p == '\n') { // 换行
            uint cursor = stdio.getCursor();
            cursor += 80 - (cursor % 80);
            if (cursor >= 25 * 80) {
               stdio.rollUp();
               cursor = 24 * 80;
            }
            stdio.moveCursor(cursor);
        }
        else if (*p == '\t') { // 制表符
            uint cursor = stdio.getCursor();
            cursor += 4 - (cursor % 4);
            if (cursor >= 25 * 80) {
               stdio.rollUp();
               cursor = 24 * 80;
            }
            stdio.moveCursor(cursor);
        }
        else {
            stdio.putchar(*p);
        }
    }
```

```
}
```

Assignment 2 线程的实现

PCB 定义:

```
struct PCB {
    int* stack;                      // 栈指针，用于调度时保存 esp
    char name[MAX_PROGRAM_NAME + 1]; // 线程名
    ProgramStatus status;             // 线程的状态
    int priority;                    // 线程优先级
    int pid;                         // 线程 pid
    int ticks;                       // 线程时间片总时间
    int ticksPassedBy;                // 线程已执行时间
};
```

基础设施，用于 PCB 管理:

```
const int PCB_SIZE = 4096;                         // PCB 的大小，4KB。
char PCB_buffer[PCB_SIZE * MAX_PROGRAM_AMOUNT]; // 存放 PCB 的数组，预留了
MAX_PROGRAM_AMOUNT 个 PCB 的大小空间。
bool PCB_used[MAX_PROGRAM_AMOUNT];                // PCB 的分配状态，true 表示已经分配，false
表示未分配。
int PCB_free_list[MAX_PROGRAM_AMOUNT];            // PCB 的空闲链表，存放未分配的 PCB 的索引。
int PCB_free_count;

int getPCBIndex(PCB* program) {
    return ((int)program - (int)PCB_buffer) / PCB_SIZE;
}

PCB* getPCB(int pid) {
    PCB* program = (PCB*)(PCB_buffer + pid * PCB_SIZE);
    return program;
}
PCB* ProgramManager::allocatePCB() {
    if (PCB_free_count == 0) {
        return nullptr;
    }
    int index = PCB_free_list[--PCB_free_count];
    PCB_used[index] = true;
    return (PCB*)(PCB_buffer + index * PCB_SIZE);
}

void ProgramManager::releasePCB(PCB* program) {
    int index = getPCBIndex(program);
    PCB_used[index] = false;
    PCB_free_list[PCB_free_count++] = index;
}
```

创建线程：

```
int ProgramManager::executeThread
       (ThreadFunction function, void* parameter, const char* name, int priority) {
    // 关中断，防止创建线程的过程被打断
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    // 分配一页作为 PCB
    PCB* thread = allocatePCB();

    if (!thread)
        return -1;

    // 初始化分配的页
    memset(thread, 0, PCB_SIZE);

    for (int i = 0; i < MAX_PROGRAM_NAME && name[i]; ++i) {
        thread->name[i] = name[i];
    }

    thread->status = ProgramStatus::READY;
    thread->priority = priority;
    thread->ticks = priority;
    thread->ticksPassedBy = 0;
    thread->pid = getPCBIndex(thread); // 线程的 pid 为其在 PCB_buffer 中的索引

    // 线程栈
    thread->stack = (int*)((int)thread + PCB_SIZE);
    thread->stack -= 7;                        // 栈顶从高地址向低地址移动 7x4 字节，存放
    thread->stack[0] = 0;                      // 保存 esi
    thread->stack[1] = 0;                      // 保存 edi
    thread->stack[2] = 0;                      // 保存 ebx
    thread->stack[3] = 0;                      // 保存 ebp
    thread->stack[4] = (int)function;          // 保存线程函数地址
    thread->stack[5] = (int)program_exit; // 保存线程退出函数地址
    thread->stack[6] = (int)parameter;         // 保存线程函数参数

    allPrograms.push(thread->pid);    // 将线程加入所有线程列表
    readyPrograms.push(thread->pid); // 将线程加入就绪线程列表

    // 恢复中断
    interruptManager.setInterruptStatus(status);
    return thread->pid;
}
```

线程调度：

```
void ProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyPrograms.size() == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    // printf("schedule\n");
    // printf("pid %d name \"%s\"\n", running->pid, running->name);

    if (running->status == ProgramStatus::RUNNING) {
        running->status = ProgramStatus::READY;
        running->ticks = running->priority;
        readyPrograms.push(running->pid);
    }
    else if (running->status == ProgramStatus::DEAD) {
        releasePCB(running);
    }

    int item = readyPrograms.front();
    PCB* next = getPCB(item);
    PCB* cur = running;
    next->status = ProgramStatus::RUNNING;
    running = next;
    readyPrograms.pop();

    asm_switch_thread(cur, next);

    interruptManager.setInterruptStatus(status);
}
```

两个测试线程工作函数，作用是不断输出同一个字符串：

```
void first_thread(void* arg) {
    char* msg = (char*)arg;
    while (true) {
        printf("from %s: %s\n", programManager.running->name, msg);
    }
}

void second_thread(void* arg) {
    char* msg = (char*)arg;
    while (true) {
        printf("from %s: %s\n", programManager.running->name, msg);
    }
}
```

创建线程:

```
char first_thread_para[] = "Hello World 1!";
    char second_thread_para[] = "Hello World 2!";
    int pid1 = programManager.executeThread(first_thread, first_thread_para, "first
thread", 1);
    int pid2 = programManager.executeThread(second_thread, second_thread_para,
"second thread", 1);
    if (pid1 == -1 || pid2 == -1) {
        printf("can not execute thread\n");
        asm_halt();
    }
    int item = programManager.readyPrograms.front();
    PCB* firstThread = getPCB(item);
    firstThread->status = ProgramStatus::RUNNING;
    programManager.readyPrograms.pop();
    programManager.running = firstThread;
    asm_switch_thread(nullptr, firstThread);
```

Assignment 3 线程调度切换的秘密

两个调度函数:

```
void first_thread(void*) {
    for (int i = 0;; i++) {
        printf("from %s: %d, %d \n", programManager.running->name, i, 1);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 2);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}
```

Assignment 4 调度算法的实现

为了尽可能做到代码复用，将 ProgramManager 部分方法移到了全局。

使用宏指向真正的 ProgramManager:

```
#include "rr.hpp"
#include "fcfs.hpp"
#include "sjf.hpp"
#include "ps.hpp"
#include "mlfq.hpp"

#define ProgramManager MLFQProgramManager
```

先来先服务调度，只在线程结束才切换：

```
void FCFSProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyPrograms.size() == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    if (running->status == ProgramStatus::DEAD) {
        releasePCB(running);
        int item = readyPrograms.front();
        readyPrograms.pop();
        PCB* next = getPCB(item);
        PCB* cur = running;
        next->status = ProgramStatus::RUNNING;
        running = next;
        asm_switch_thread(cur, next);
    }

    interruptManager.setInterruptStatus(status);
}
```

非抢占式最短作业优先调度，为了方便直接用 `priority` 作为时长：

```
void SJFProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyCount == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    if (!running || running->status == ProgramStatus::DEAD) {
        if (running) releasePCB(running);
        int best = -1;
        for (int i = 0; i < readyCount; i++) {
            PCB* program = getPCB(readyPrograms[i]);
            if (best == -1 || program->priority < getPCB(best)->priority) {
                best = readyPrograms[i];
            }
        }

        if (best != -1) {
            readyPrograms[best] = readyPrograms[--readyCount];
            PCB* next = getPCB(best);
            PCB* cur = running;
            next->status = ProgramStatus::RUNNING;
            running = next;
            asm_switch_thread(cur, next);
        }
    }

    interruptManager.setInterruptStatus(status);
}
```

优先级调度，和 SJF 几乎一致：

```
void PSProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyCount == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    if (!running || running->status == ProgramStatus::DEAD) {
        if (running) releasePCB(running);
        int best = -1;
        for (int i = 0; i < readyCount; i++) {
            PCB* program = getPCB(readyPrograms[i]);
            if (best == -1 || program->priority > getPCB(best)->priority) {
                best = readyPrograms[i];
            }
        }

        if (best != -1) {
            readyPrograms[best] = readyPrograms[--readyCount];
            PCB* next = getPCB(best);
            PCB* cur = running;
            next->status = ProgramStatus::RUNNING;
            running = next;
            asm_switch_thread(cur, next);
        }
    }

    interruptManager.setInterruptStatus(status);
}
```

非抢占式多级反馈队列调度算法：

定义五层队列及其时间片权重：

```
class MLFQProgramManager {
public:
    FlatQueue<int, MAX_PROGRAM_AMOUNT> readyPrograms[5]; // 就绪队列
    static constexpr int weight[5] = { 1, 2, 4, 8, 16 }; // 各个优先级的权重
    PCB* running;                                       // 当前执行的线程
    int running_layer;                                  // 当前执行的线程所在的层级
```

调度算法：

```
void MLFQProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (running && running->status == ProgramStatus::RUNNING) {
        int next_layer = std::min(4, running_layer + 1);
        running->status = ProgramStatus::READY;
        running->ticks = running->priority * weight[next_layer];
        readyPrograms[next_layer].push(running->pid);
    }
    else if (running && running->status == ProgramStatus::DEAD) {
        releasePCB(running);
    }

    for (int i = 0; i < 5; i++) {
        if (readyPrograms[i].empty()) continue;
        int item = readyPrograms[i].front();
        PCB* next = getPCB(item);
        PCB* cur = running;
        next->status = ProgramStatus::RUNNING;
        running = next;
        printf("switch from %s(layer %d) to %s(layer %d)\n", cur->name, running_layer,
next->name, i);
        running_layer = i;
        readyPrograms[i].pop();
        asm_switch_thread(cur, next);
        break;
    }
    interruptManager.setInterruptStatus(status);
}
```

测试线程，一个在有限时间内完成，一个无限循环：

```
void first_thread(void*) {
    for (int i = 0; i < 1000000000; i++) {}
}
```

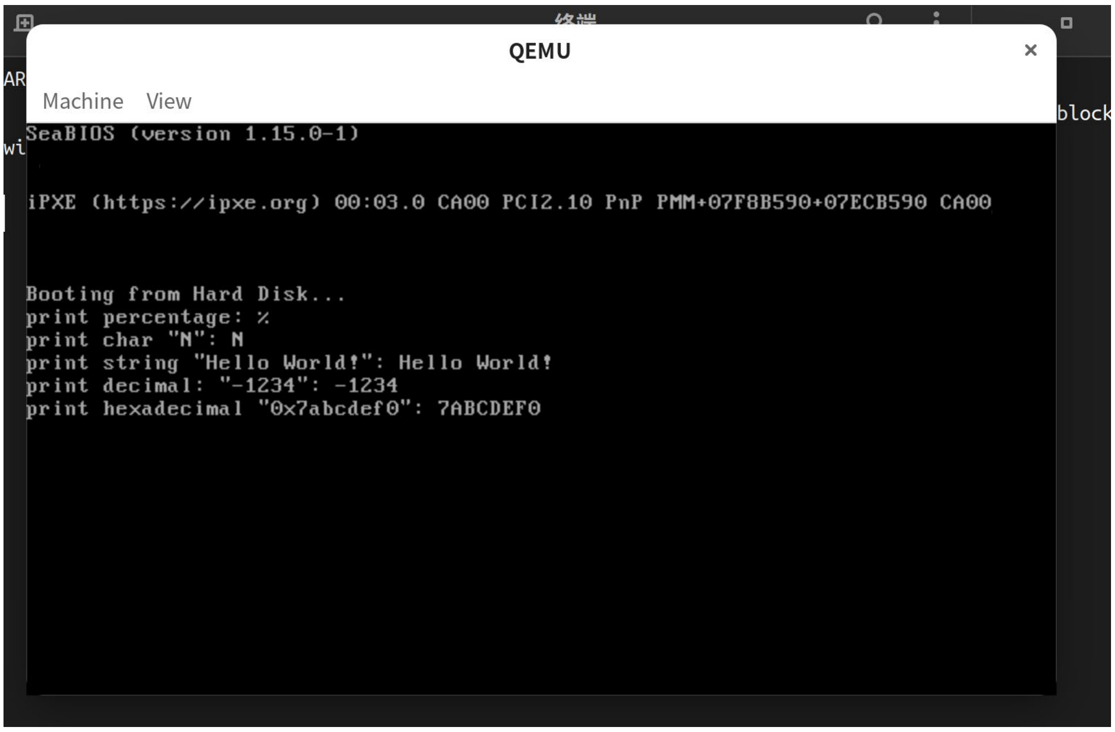
```
void second_thread(void*) {
    asm_halt();
}
```

修改 schedule_time_interrupt_handler 以记录线程切换:

```
void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0) {
        printf("before schedule: %s\n", programManager.running->name);
        programManager.schedule();
        printf("after schedule: %s\n", programManager.running->name);
    }
}
```

4. 实验结果

Assignment 1 printf 的实现

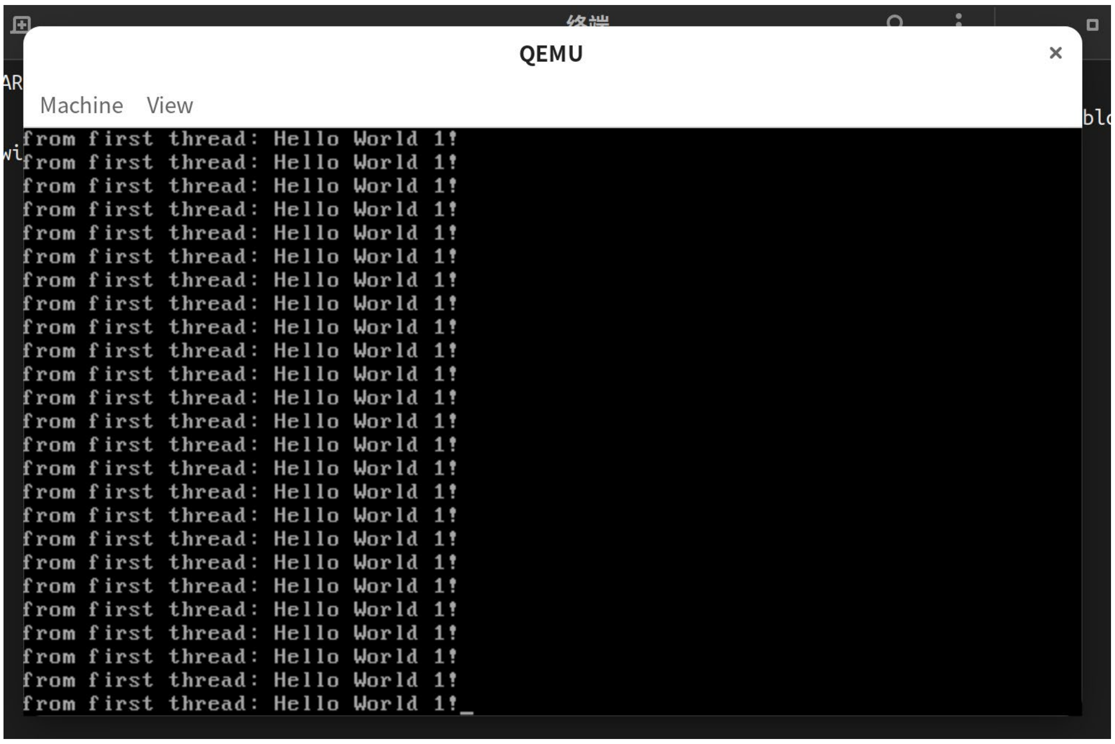


```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00


Booting from Hard Disk...
print percentage: %
print char "N": N
print string "Hello World!": Hello World!
print decimal: "-1234": -1234
print hexadecimal "0x7abcdef0": 7ABCDEF0
```

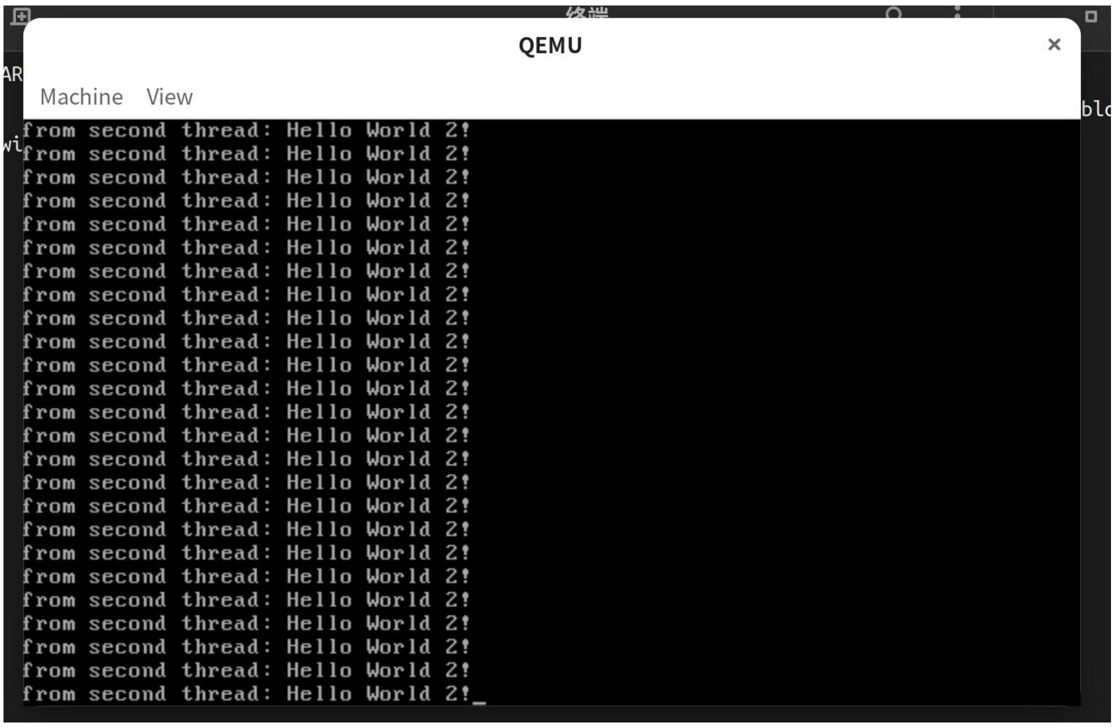
Assignment 2 线程的实现

假设能看见动图。

效果是每个线程分别轮流占据一定长度的时间片，在时间片内输出自己的字符串。

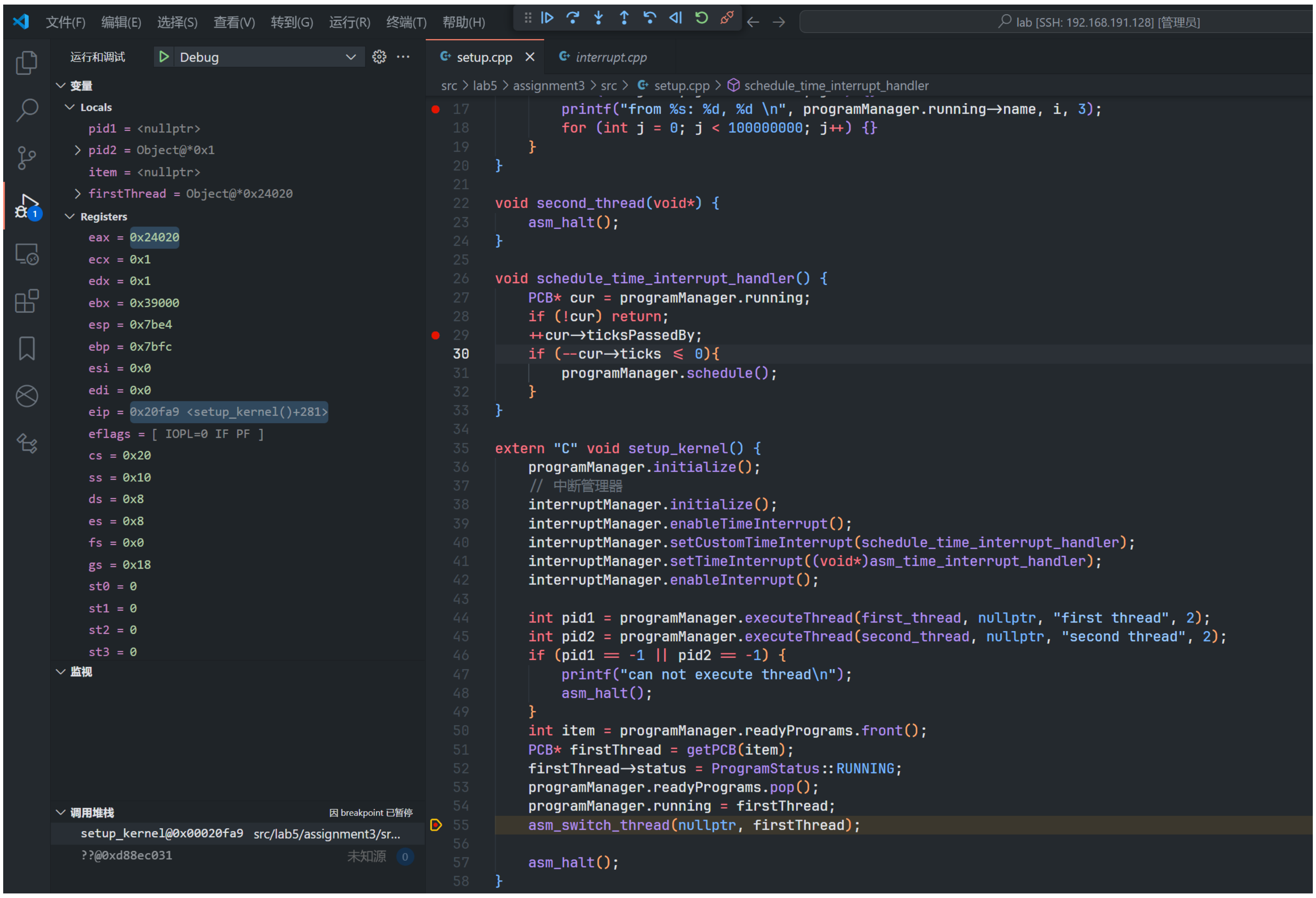


```
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!
from first thread: Hello World 1!_
```



```
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!
from second thread: Hello World 2!_
```

Assignment 3 线程调度切换的秘密

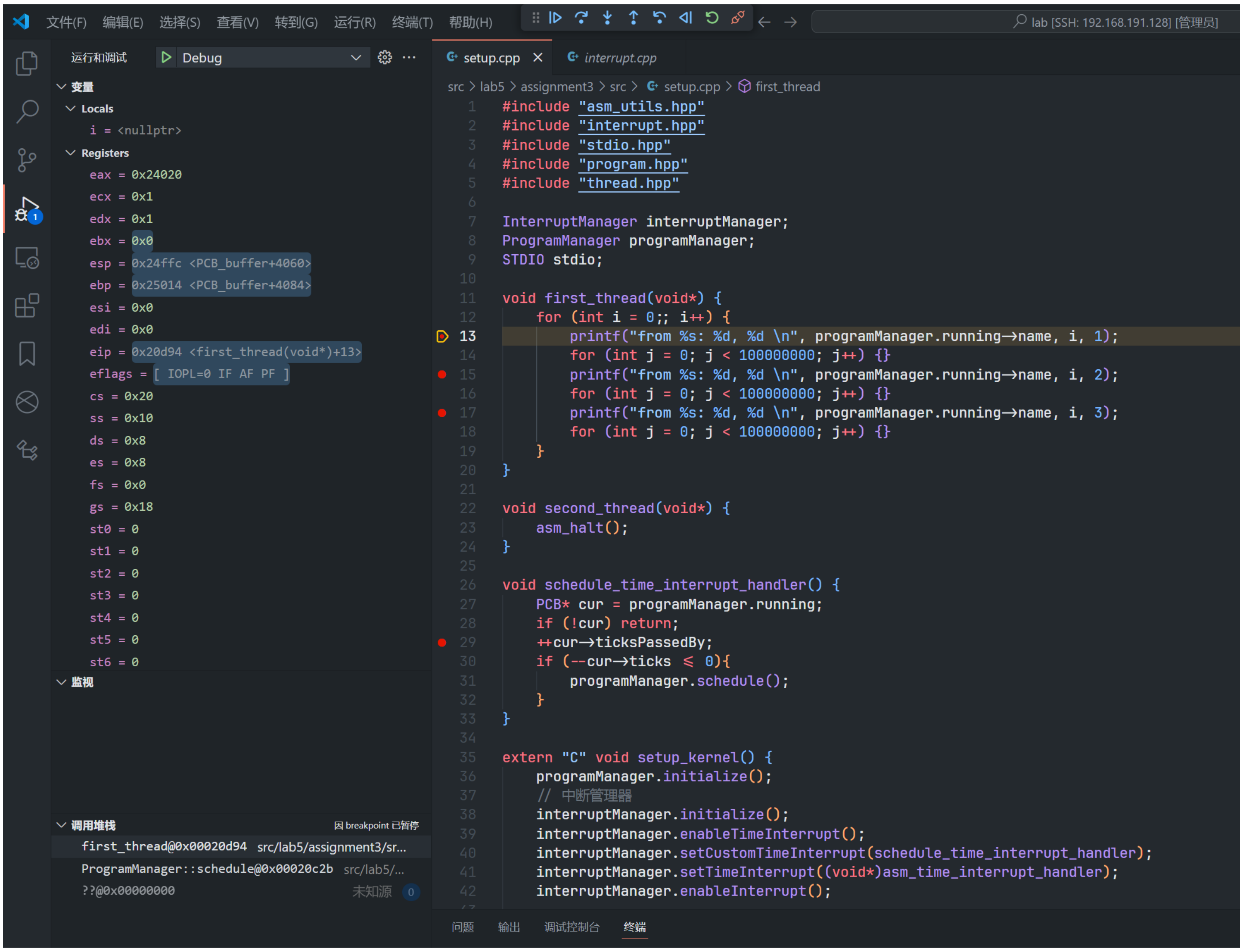

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
interrupt.cpp
src > lab5 > assignment3 > src > setup.cpp > schedule_time_interrupt_handler
变量
监视
调用堆栈
breakpoint 已暂停
未知源

```
Locals
  pid1 = <nullptr>
> pid2 = Object@*0x1
  item = <nullptr>
> firstThread = Object@*0x24020
Registers
  eax = 0x24020
  ecx = 0x1
  edx = 0x1
  ebx = 0x39000
  esp = 0x7be4
  ebp = 0x7bfc
  esi = 0x0
  edi = 0x0
  eip = 0x20fa9 <setup_kernel()+281>
  eflags = [ IOPL=0 IF PF ]
  cs = 0x20
  ss = 0x10
  ds = 0x8
  es = 0x8
  fs = 0x0
  gs = 0x18
  st0 = 0
  st1 = 0
  st2 = 0
  st3 = 0
```

```
setup_kernel@0x00020fa9   src/lab5/assignment3/sr...
??@0xd88ec031
```

```cpp
        printf("from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}

void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0){
        programManager.schedule();
    }
}

extern "C" void setup_kernel() {
    programManager.initialize();
    // 中断管理器
    interruptManager.initialize();
    interruptManager.enableTimeInterrupt();
    interruptManager.setCustomTimeInterrupt(schedule_time_interrupt_handler);
    interruptManager.setTimeInterrupt((void*)asm_time_interrupt_handler);
    interruptManager.enableInterrupt();

    int pid1 = programManager.executeThread(first_thread, nullptr, "first thread", 2);
    int pid2 = programManager.executeThread(second_thread, nullptr, "second thread", 2);
    if (pid1 == -1 || pid2 == -1) {
        printf("can not execute thread\n");
        asm_halt();
    }
    int item = programManager.readyPrograms.front();
    PCB* firstThread = getPCB(item);
    firstThread->status = ProgramStatus::RUNNING;
    programManager.readyPrograms.pop();
    programManager.running = firstThread;
    asm_switch_thread(nullptr, firstThread);

    asm_halt();
}
```

在 setup_kernel 中的 asm_switch_thread 下断，可见从队列取出的线程 pid（item）为 0，即 pid1 的值。

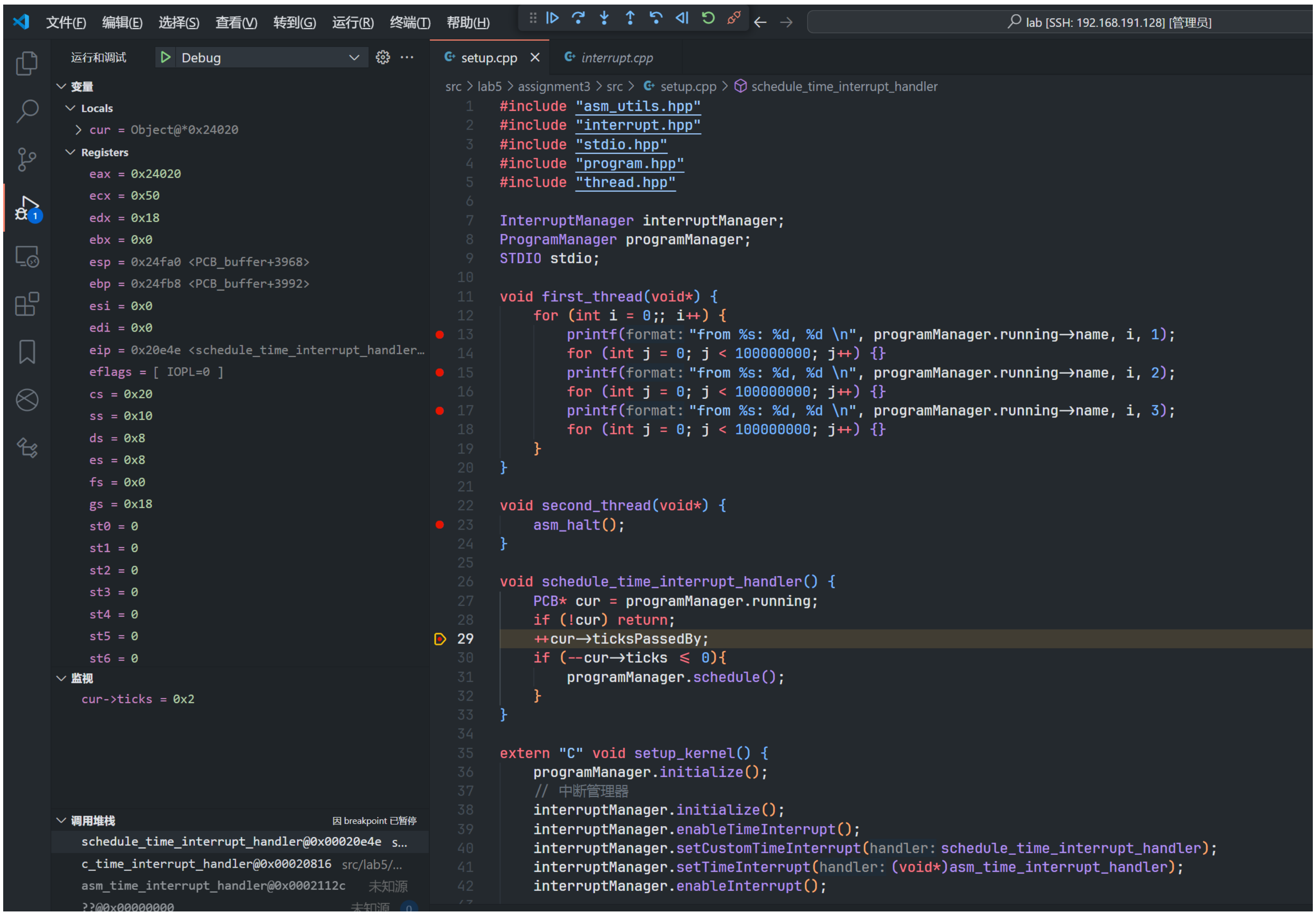

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
interrupt.cpp
src > lab5 > assignment3 > src > setup.cpp > first_thread
问题
输出
调试控制台
终端

```
变量
  Locals
    i = <nullptr>
  Registers
    eax = 0x24020
    ecx = 0x1
    edx = 0x1
    ebx = 0x0
    esp = 0x24ffc <PCB_buffer+4060>
    ebp = 0x25014 <PCB_buffer+4084>
    esi = 0x0
    edi = 0x0
    eip = 0x20d94 <first_thread(void*)+13>
    eflags = [ IOPL=0 IF AF PF ]
    cs = 0x20
    ss = 0x10
    ds = 0x8
    es = 0x8
    fs = 0x0
    gs = 0x18
    st0 = 0
    st1 = 0
    st2 = 0
    st3 = 0
    st4 = 0
    st5 = 0
    st6 = 0
监视

调用堆栈                                   因 breakpoint 已暂停
  first_thread@0x00020d94   src/lab5/assignment3/sr...
  ProgramManager::schedule@0x00020c2b   src/lab5/...
  ??@0x00000000                                 未知源
```

```cpp
#include "asm_utils.hpp"
#include "interrupt.hpp"
#include "stdio.hpp"
#include "program.hpp"
#include "thread.hpp"

InterruptManager interruptManager;
ProgramManager programManager;
STDIO stdio;

void first_thread(void*) {
    for (int i = 0;; i++) {
        printf("from %s: %d, %d \n", programManager.running->name, i, 1);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 2);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}

void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0){
        programManager.schedule();
    }
}

extern "C" void setup_kernel() {
    programManager.initialize();
    // 中断管理器
    interruptManager.initialize();
    interruptManager.enableTimeInterrupt();
    interruptManager.setCustomTimeInterrupt(schedule_time_interrupt_handler);
    interruptManager.setTimeInterrupt((void*)asm_time_interrupt_handler);
    interruptManager.enableInterrupt();
```

继续执行，程序如预期在 first_thread 断下。

若干次在 first_thread 断下后，时钟中断触发：

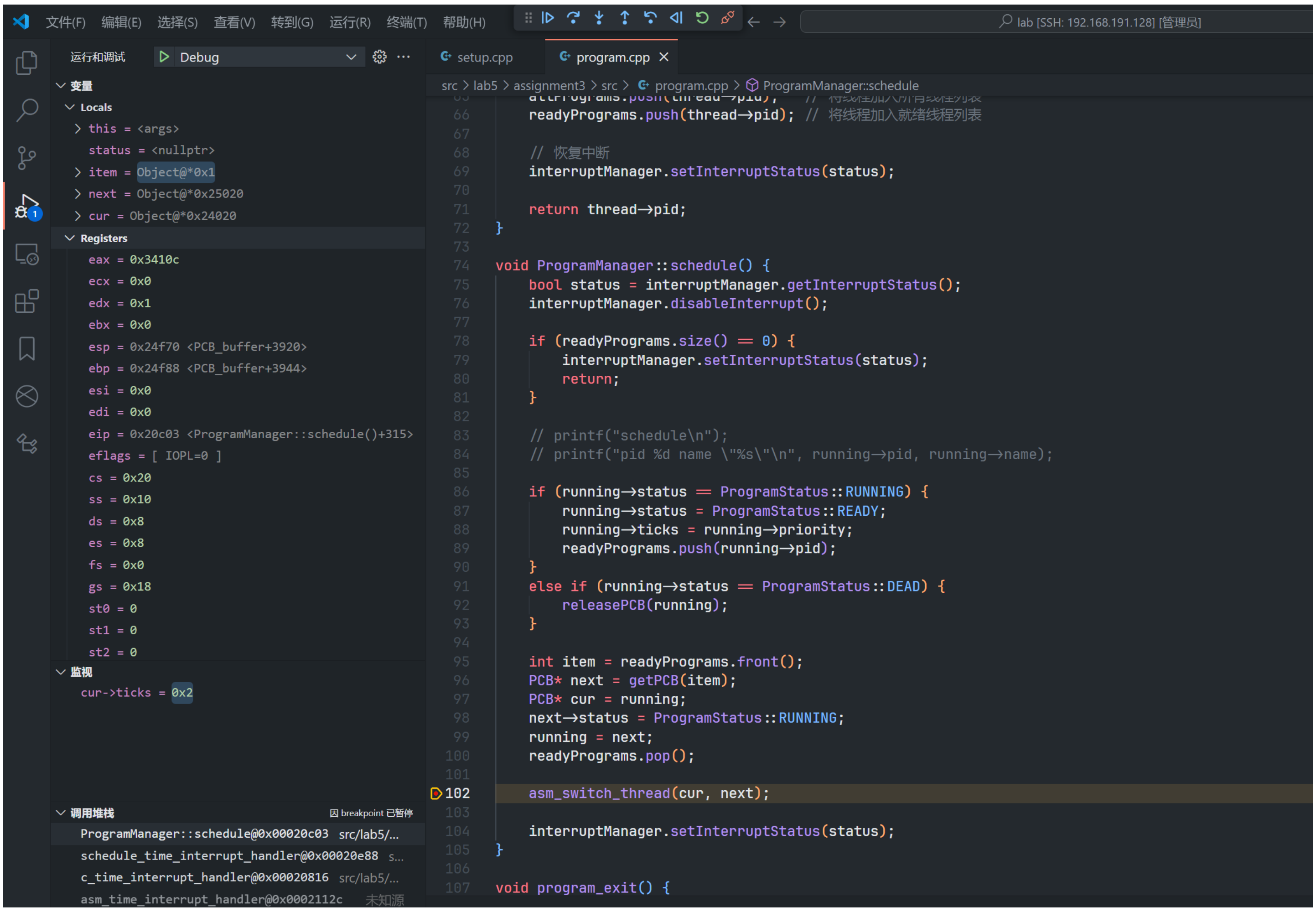

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
interrupt.cpp
src > lab5 > assignment3 > src > setup.cpp > schedule_time_interrupt_handler

```
变量
  Locals
    > cur = Object@*0x24020
  Registers
      eax = 0x24020
      ecx = 0x50
      edx = 0x18
      ebx = 0x0
      esp = 0x24fa0 <PCB_buffer+3968>
      ebp = 0x24fb8 <PCB_buffer+3992>
      esi = 0x0
      edi = 0x0
      eip = 0x20e4e <schedule_time_interrupt_handler…
      eflags = [ IOPL=0 ]
      cs = 0x20
      ss = 0x10
      ds = 0x8
      es = 0x8
      fs = 0x0
      gs = 0x18
      st0 = 0
      st1 = 0
      st2 = 0
      st3 = 0
      st4 = 0
      st5 = 0
      st6 = 0
监视
    cur->ticks = 0x2

调用堆栈                                   因 breakpoint 已暂停
    schedule_time_interrupt_handler@0x00020e4e  s…
    c_time_interrupt_handler@0x00020816  src/lab5/…
    asm_time_interrupt_handler@0x0002112c   未知源
    ??@0x00000000                          未知源
```

```cpp
#include "asm_utils.hpp"
#include "interrupt.hpp"
#include "stdio.hpp"
#include "program.hpp"
#include "thread.hpp"

InterruptManager interruptManager;
ProgramManager programManager;
STDIO stdio;

void first_thread(void*) {
    for (int i = 0;; i++) {
        printf(format: "from %s: %d, %d \n", programManager.running->name, i, 1);
        for (int j = 0; j < 100000000; j++) {}
        printf(format: "from %s: %d, %d \n", programManager.running->name, i, 2);
        for (int j = 0; j < 100000000; j++) {}
        printf(format: "from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}

void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0){
        programManager.schedule();
    }
}

extern "C" void setup_kernel() {
    programManager.initialize();
    // 中断管理器
    interruptManager.initialize();
    interruptManager.enableTimeInterrupt();
    interruptManager.setCustomTimeInterrupt(handler: schedule_time_interrupt_handler);
    interruptManager.setTimeInterrupt(handler: (void*)asm_time_interrupt_handler);
    interruptManager.enableInterrupt();
```

此时 cur->ticks 等于 2，因此不会触发线程调度，返回到原来的位置继续执行。

第二次进入 schedule_time_interrupt_handler 后，调用 program.schedule()：



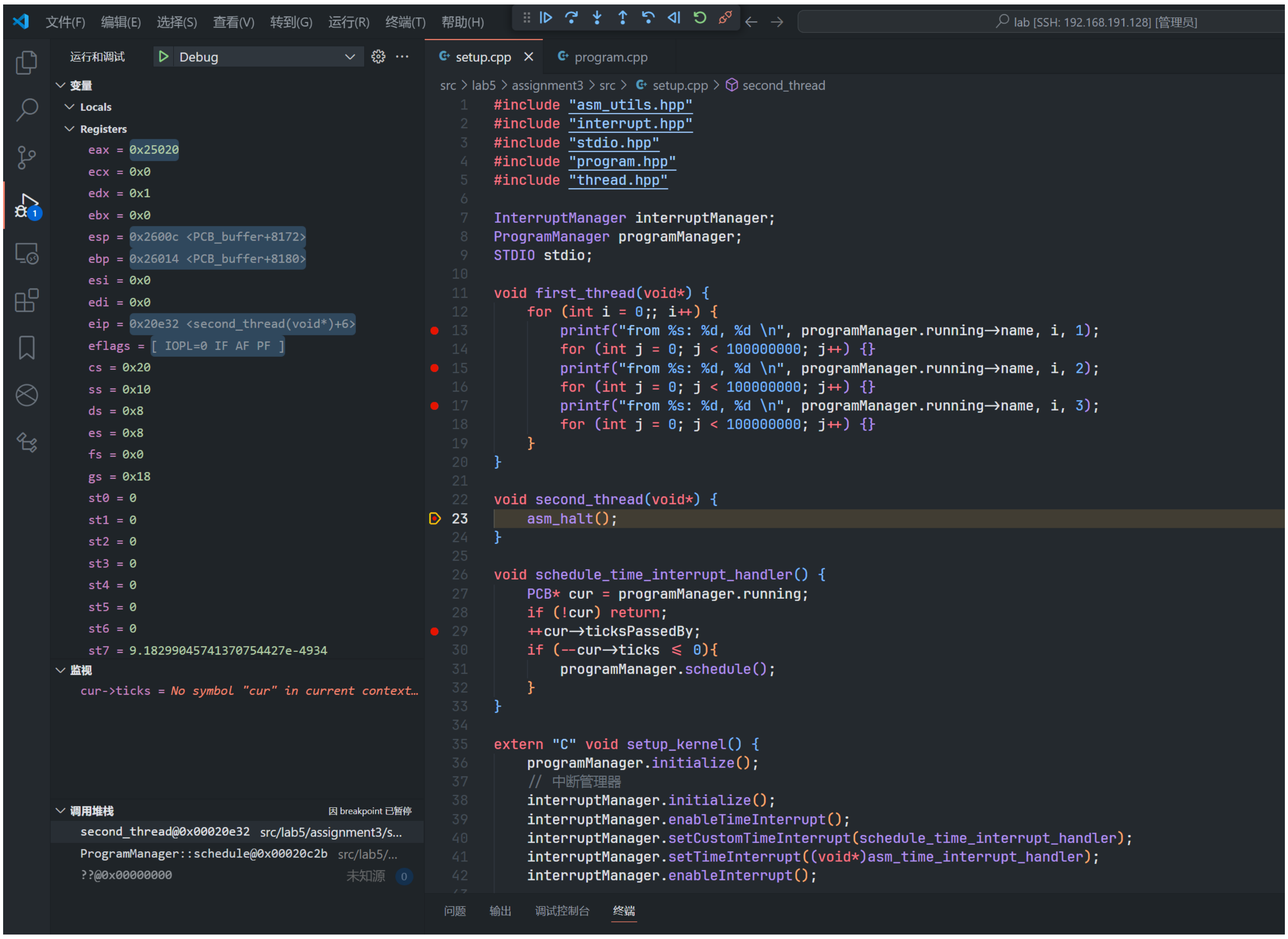
```
变量
  Locals
    > this = <args>
      status = <nullptr>
    > item = Object@*0x1
    > next = Object@*0x25020
    > cur = Object@*0x24020
  Registers
      eax = 0x3410c
      ecx = 0x0
      edx = 0x1
      ebx = 0x0
      esp = 0x24f70 <PCB_buffer+3920>
      ebp = 0x24f88 <PCB_buffer+3944>
      esi = 0x0
      edi = 0x0
      eip = 0x20c03 <ProgramManager::schedule()+315>
      eflags = [ IOPL=0 ]
      cs = 0x20
      ss = 0x10
      ds = 0x8
      es = 0x8
      fs = 0x0
      gs = 0x18
      st0 = 0
      st1 = 0
      st2 = 0
监视
    cur->ticks = 0x2
调用堆栈                                    因 breakpoint 已暂停
    ProgramManager::schedule@0x00020c03  src/lab5/...
    schedule_time_interrupt_handler@0x00020e88  s...
    c_time_interrupt_handler@0x00020816  src/lab5/...
    asm_time_interrupt_handler@0x0002112c  未知源
```

src > lab5 > assignment3 > src > program.cpp > ProgramManager::schedule

```cpp
    allPrograms.push(thread->pid);   // 将线程加入所有线程列表
    readyPrograms.push(thread->pid); // 将线程加入就绪线程列表

    // 恢复中断
    interruptManager.setInterruptStatus(status);

    return thread->pid;
}

void ProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyPrograms.size() == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    // printf("schedule\n");
    // printf("pid %d name \"%s\"\n", running->pid, running->name);

    if (running->status == ProgramStatus::RUNNING) {
        running->status = ProgramStatus::READY;
        running->ticks = running->priority;
        readyPrograms.push(running->pid);
    }
    else if (running->status == ProgramStatus::DEAD) {
        releasePCB(running);
    }

    int item = readyPrograms.front();
    PCB* next = getPCB(item);
    PCB* cur = running;
    next->status = ProgramStatus::RUNNING;
    running = next;
    readyPrograms.pop();

    asm_switch_thread(cur, next);

    interruptManager.setInterruptStatus(status);
}

void program_exit() {
```

由变量窗口可知此时从队列里取出的线程 pid 为 1，即 second_thread。

继续执行，线程断在 second_thread 的 asm_halt 上：



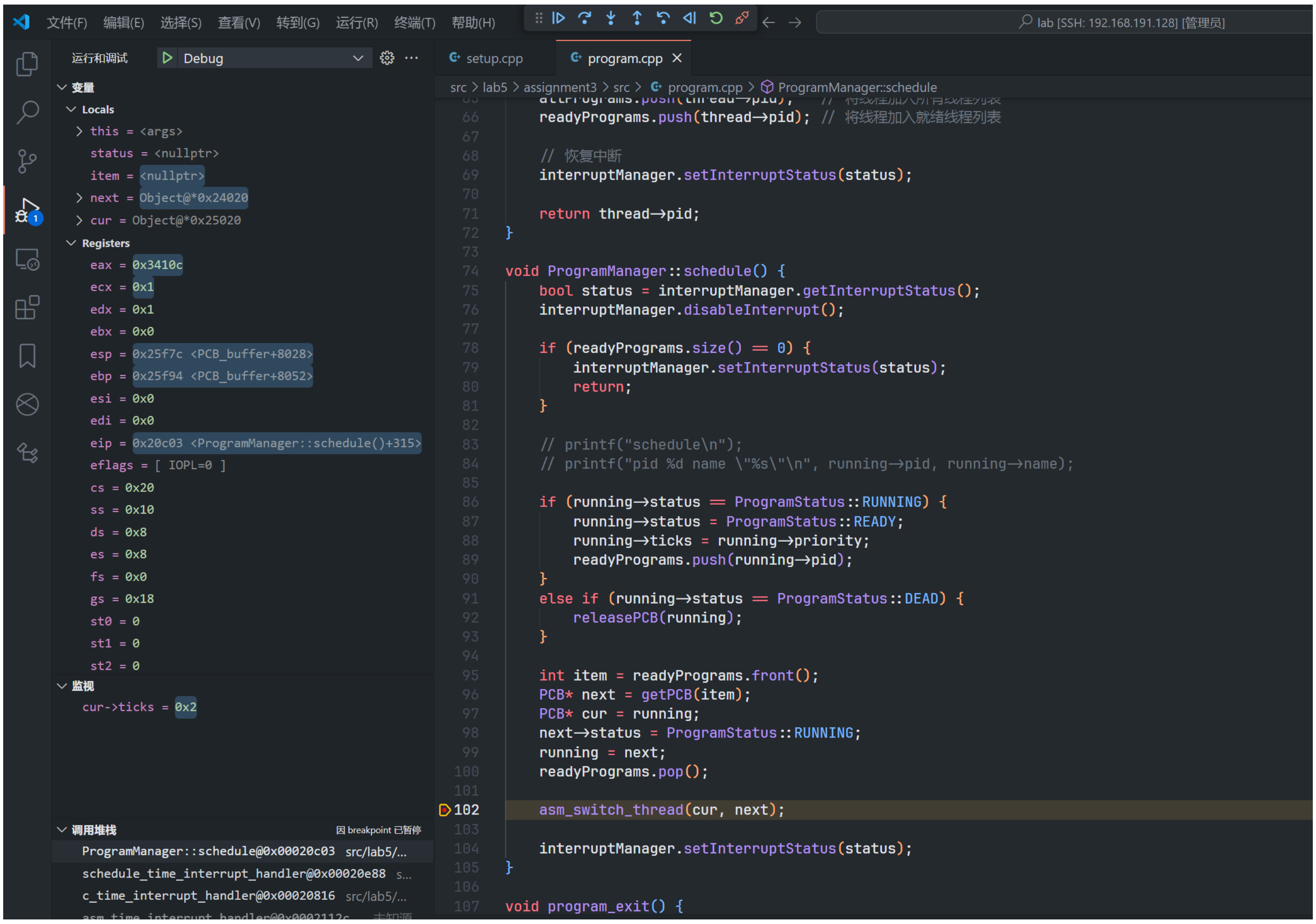
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
program.cpp
src > lab5 > assignment3 > src > setup.cpp > second_thread
问题
输出
调试控制台
终端

```
变量
  Locals
  Registers
    eax = 0x25020
    ecx = 0x0
    edx = 0x1
    ebx = 0x0
    esp = 0x2600c <PCB_buffer+8172>
    ebp = 0x26014 <PCB_buffer+8180>
    esi = 0x0
    edi = 0x0
    eip = 0x20e32 <second_thread(void*)+6>
    eflags = [ IOPL=0 IF AF PF ]
    cs = 0x20
    ss = 0x10
    ds = 0x8
    es = 0x8
    fs = 0x0
    gs = 0x18
    st0 = 0
    st1 = 0
    st2 = 0
    st3 = 0
    st4 = 0
    st5 = 0
    st6 = 0
    st7 = 9.18299045741370754427e-4934
监视
  cur->ticks = No symbol "cur" in current context…

调用堆栈                                   因 breakpoint 已暂停
  second_thread@0x00020e32   src/lab5/assignment3/s...
  ProgramManager::schedule@0x00020c2b   src/lab5/...
  ??@0x00000000                          未知源  0
```

```cpp
#include "asm_utils.hpp"
#include "interrupt.hpp"
#include "stdio.hpp"
#include "program.hpp"
#include "thread.hpp"

InterruptManager interruptManager;
ProgramManager programManager;
STDIO stdio;

void first_thread(void*) {
    for (int i = 0;; i++) {
        printf("from %s: %d, %d \n", programManager.running->name, i, 1);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 2);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}

void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0){
        programManager.schedule();
    }
}

extern "C" void setup_kernel() {
    programManager.initialize();
    // 中断管理器
    interruptManager.initialize();
    interruptManager.enableTimeInterrupt();
    interruptManager.setCustomTimeInterrupt(schedule_time_interrupt_handler);
    interruptManager.setTimeInterrupt((void*)asm_time_interrupt_handler);
    interruptManager.enableInterrupt();
```

对比两个函数的 ebp，恰好相差 0x1000=4096，即一个 PCB 的大小。

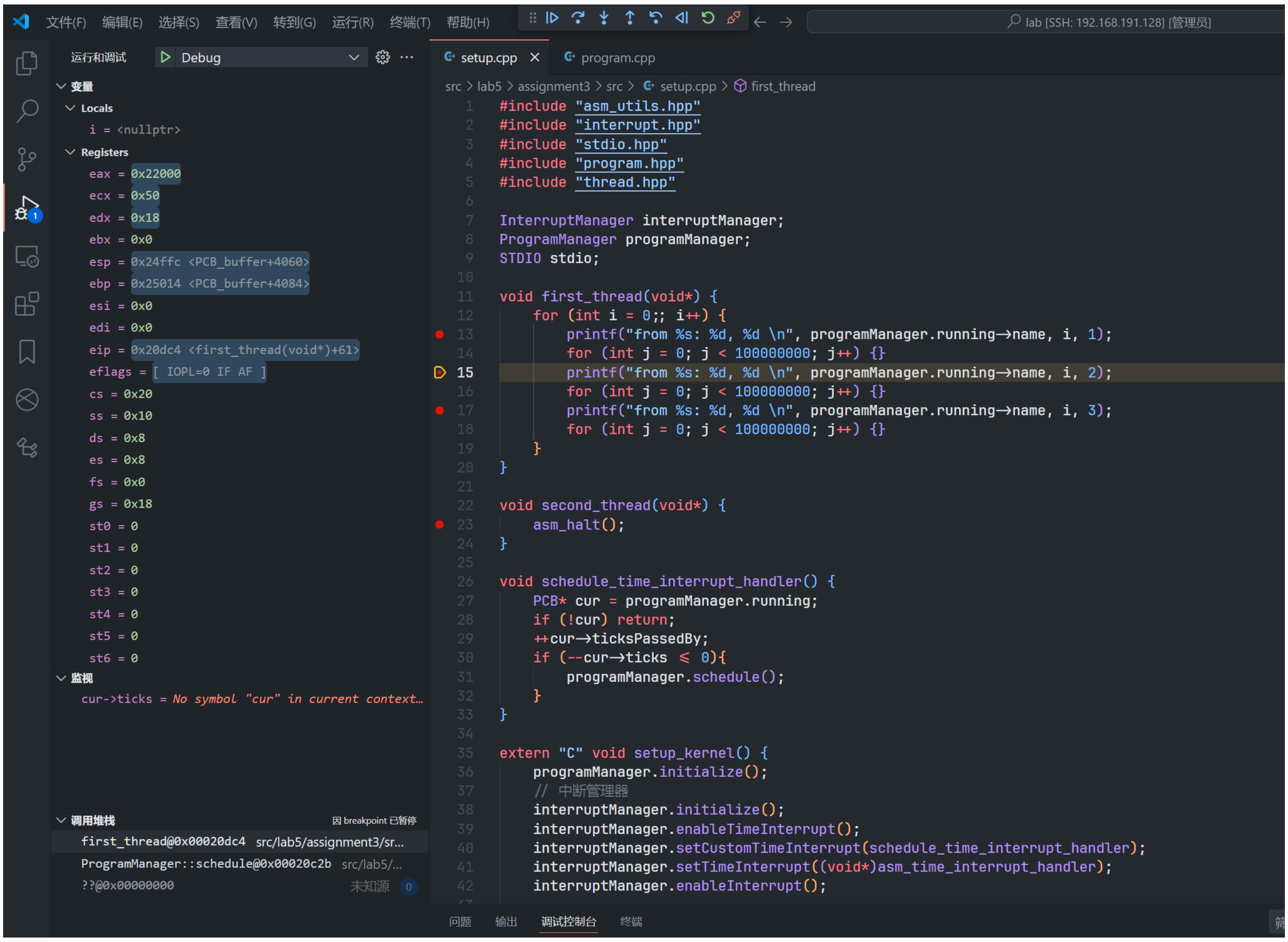
继续执行，在若干次断点后在此调用 program.schedule()：


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
program.cpp
src > lab5 > assignment3 > src > program.cpp > ProgramManager::schedule
因 breakpoint 已暂停

```
变量
  Locals
    > this = <args>
      status = <nullptr>
      item = <nullptr>
    > next = Object@*0x24020
    > cur = Object@*0x25020
  Registers
      eax = 0x3410c
      ecx = 0x1
      edx = 0x1
      ebx = 0x0
      esp = 0x25f7c <PCB_buffer+8028>
      ebp = 0x25f94 <PCB_buffer+8052>
      esi = 0x0
      edi = 0x0
      eip = 0x20c03 <ProgramManager::schedule()+315>
      eflags = [ IOPL=0 ]
      cs = 0x20
      ss = 0x10
      ds = 0x8
      es = 0x8
      fs = 0x0
      gs = 0x18
      st0 = 0
      st1 = 0
      st2 = 0
监视
    cur->ticks = 0x2
调用堆栈
    ProgramManager::schedule@0x00020c03  src/lab5/...
    schedule_time_interrupt_handler@0x00020e88  s...
    c_time_interrupt_handler@0x00020816  src/lab5/...
    asm_time_interrupt_handler@0x0002112c  未知源
```

```cpp
    allPrograms.push(thread->pid);   // 将线程加入所有线程列表
    readyPrograms.push(thread->pid); // 将线程加入就绪线程列表

    // 恢复中断
    interruptManager.setInterruptStatus(status);

    return thread->pid;
}

void ProgramManager::schedule() {
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    if (readyPrograms.size() == 0) {
        interruptManager.setInterruptStatus(status);
        return;
    }

    // printf("schedule\n");
    // printf("pid %d name \"%s\"\n", running->pid, running->name);

    if (running->status == ProgramStatus::RUNNING) {
        running->status = ProgramStatus::READY;
        running->ticks = running->priority;
        readyPrograms.push(running->pid);
    }
    else if (running->status == ProgramStatus::DEAD) {
        releasePCB(running);
    }

    int item = readyPrograms.front();
    PCB* next = getPCB(item);
    PCB* cur = running;
    next->status = ProgramStatus::RUNNING;
    running = next;
    readyPrograms.pop();

    asm_switch_thread(cur, next);

    interruptManager.setInterruptStatus(status);
}

void program_exit() {
```

此时取出的 item 为 0，即 pid1 的值。


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
lab [SSH: 192.168.191.128] [管理员]
运行和调试
Debug
setup.cpp
program.cpp
src > lab5 > assignment3 > src > setup.cpp > first_thread
问题
输出
调试控制台
终端

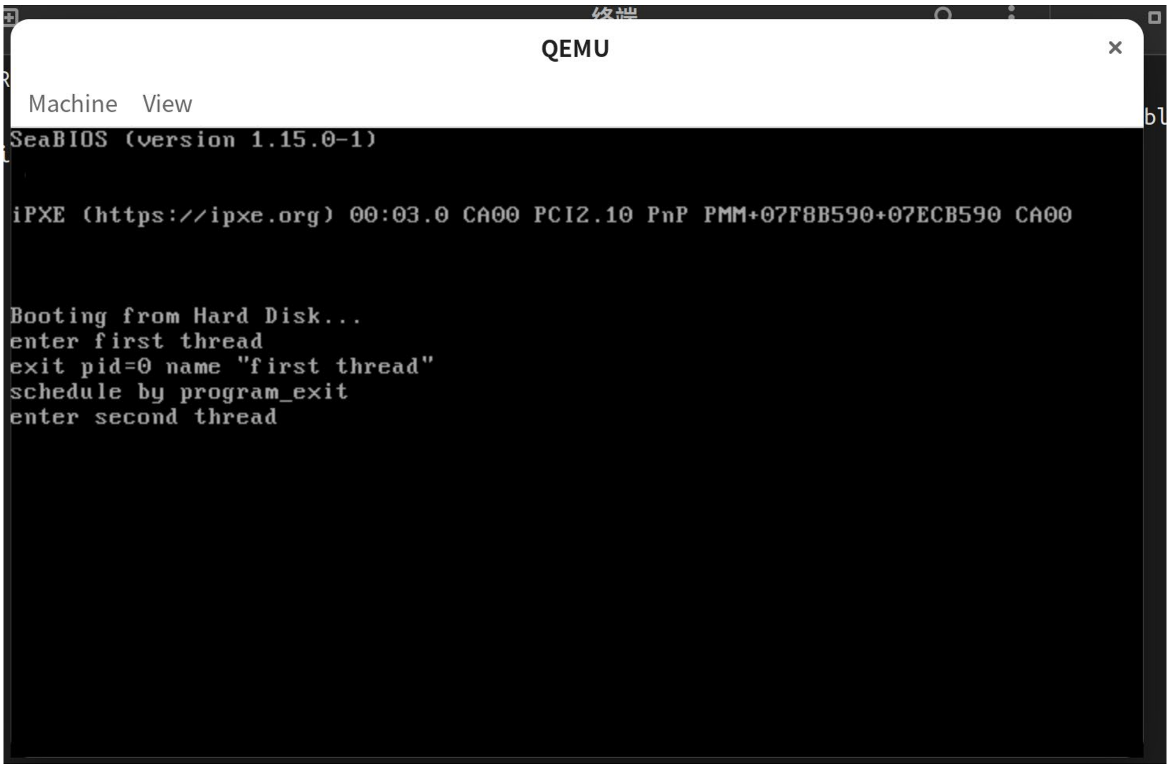
```
变量
  Locals
    i = <nullptr>
  Registers
    eax = 0x22000
    ecx = 0x50
    edx = 0x18
    ebx = 0x0
    esp = 0x24ffc <PCB_buffer+4060>
    ebp = 0x25014 <PCB_buffer+4084>
    esi = 0x0
    edi = 0x0
    eip = 0x20dc4 <first_thread(void*)+61>
    eflags = [ IOPL=0 IF AF ]
    cs = 0x20
    ss = 0x10
    ds = 0x8
    es = 0x8
    fs = 0x0
    gs = 0x18
    st0 = 0
    st1 = 0
    st2 = 0
    st3 = 0
    st4 = 0
    st5 = 0
    st6 = 0
监视
  cur->ticks = No symbol "cur" in current context…
调用堆栈                                       因 breakpoint 已暂停
  first_thread@0x00020dc4  src/lab5/assignment3/sr...
  ProgramManager::schedule@0x00020c2b  src/lab5/...
  ??@0x00000000                          未知源 0
```

```cpp
#include "asm_utils.hpp"
#include "interrupt.hpp"
#include "stdio.hpp"
#include "program.hpp"
#include "thread.hpp"

InterruptManager interruptManager;
ProgramManager programManager;
STDIO stdio;

void first_thread(void*) {
    for (int i = 0;; i++) {
        printf("from %s: %d, %d \n", programManager.running->name, i, 1);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 2);
        for (int j = 0; j < 100000000; j++) {}
        printf("from %s: %d, %d \n", programManager.running->name, i, 3);
        for (int j = 0; j < 100000000; j++) {}
    }
}

void second_thread(void*) {
    asm_halt();
}

void schedule_time_interrupt_handler() {
    PCB* cur = programManager.running;
    if (!cur) return;
    ++cur->ticksPassedBy;
    if (--cur->ticks <= 0){
        programManager.schedule();
    }
}

extern "C" void setup_kernel() {
    programManager.initialize();
    // 中断管理器
    interruptManager.initialize();
    interruptManager.enableTimeInterrupt();
    interruptManager.setCustomTimeInterrupt(schedule_time_interrupt_handler);
    interruptManager.setTimeInterrupt((void*)asm_time_interrupt_handler);
    interruptManager.enableInterrupt();
```

成功回到 first_thread 的下一条指令。

Assignment 4 调度算法的实现

FCFS:



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00



Booting from Hard Disk...
enter first thread
exit pid=0 name "first thread"
schedule by program_exit
enter second thread
```

线程执行的过程没有被打断，直到线程结束在 program_exit 被释放并调度下一个线程。

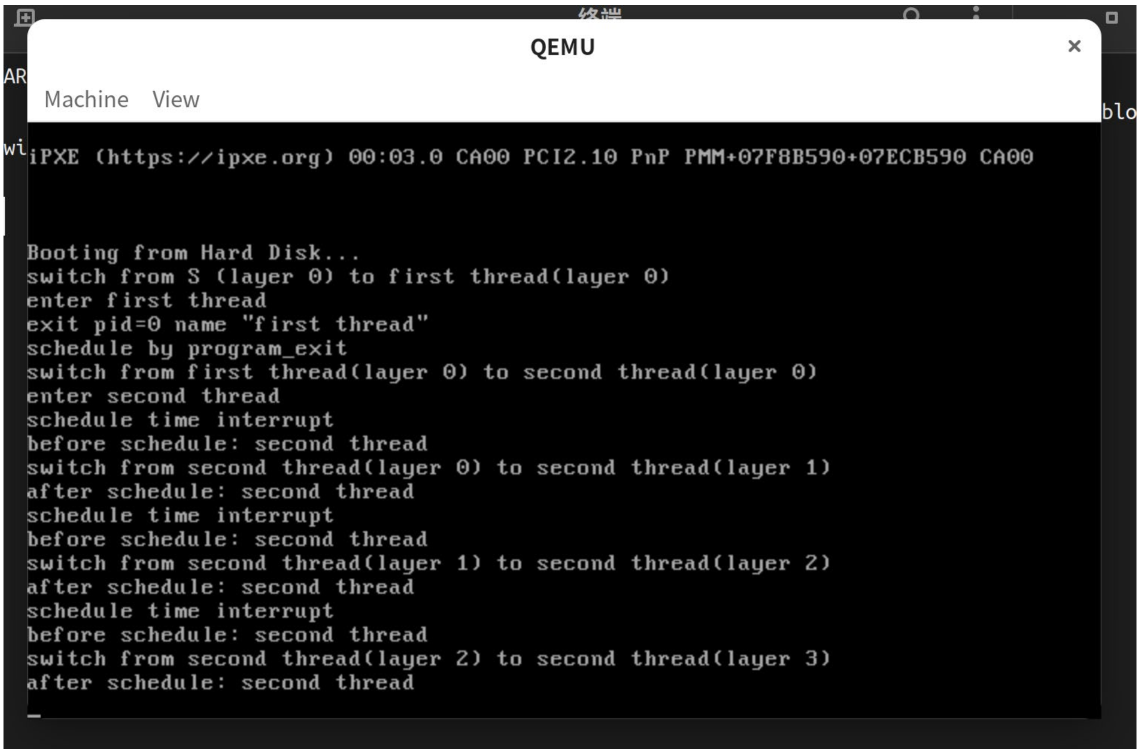
SJF：将 thread2 放到上面，并将 pid1 优先级（实际上是时间）改为 10：

```
int pid2 = programManager.executeThread(second_thread, nullptr, "second thread", 1);
int pid1 = programManager.executeThread(first_thread, nullptr, "first thread", 10);
```

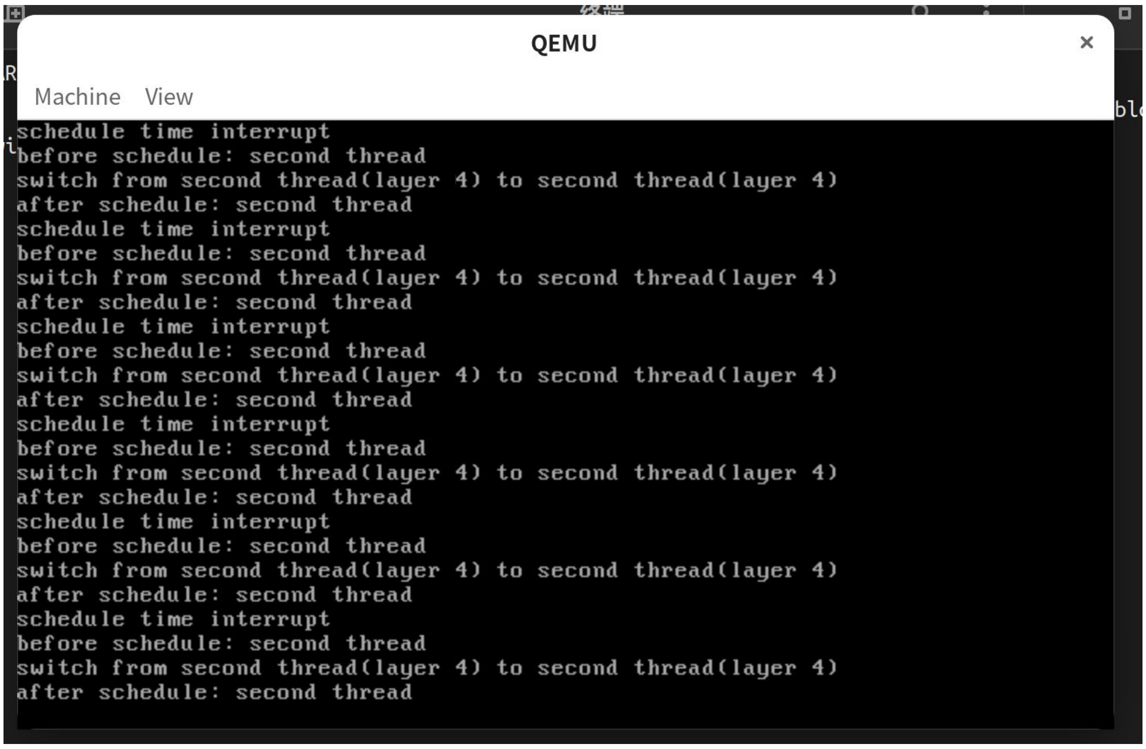
结果和 FCFS 一样就不放了，测试结果表明 SJF 能每次选取时间最短的任务调度。

PR 和上面一样的也不放了。

MLFQ：



```
iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00



Booting from Hard Disk...
switch from S (layer 0) to first thread(layer 0)
enter first thread
exit pid=0 name "first thread"
schedule by program_exit
switch from first thread(layer 0) to second thread(layer 0)
enter second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 0) to second thread(layer 1)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 1) to second thread(layer 2)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 2) to second thread(layer 3)
after schedule: second thread
_
```



```
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
schedule time interrupt
before schedule: second thread
switch from second thread(layer 4) to second thread(layer 4)
after schedule: second thread
```

每次从尽可能从高优先级队列选取任务并调度，如果一次调度时间片结束还没结束，就将其放到低优先级的队列。

5. 总结

被迫写 C style 代码。没有 C++标准库的情况下连虚函数和 RTTI 都没法用，不知道是不是 ld 的参数没有配置好，不是很想折腾。

Assignment 3 中 asm_switch_thread 的汇编层面上的调试还是太难了，即使知道栈会怎么变化也没什么好的办法能漂亮的展示出栈的切换前后函数返回地址的变化，索性放弃。