

本科生实验报告

实验课程:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_操作系统\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验名称:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_内核线程\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

专业名称:\_\_\_\_\_\_\_\_\_\_\_计算机科学与技术\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学生姓名:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_凌国明\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学生学号:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_21307077\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验地点:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_教室\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验成绩:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

报告时间:\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023.05.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **实验要求**

* 学习C语言的可变参数机制的实现方法，实现一个简单的printf函数
* 学习线程控制块的数据结构——PCB，实现RR和其他的线程调度算法

1. 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。
2. 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。
3. 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。

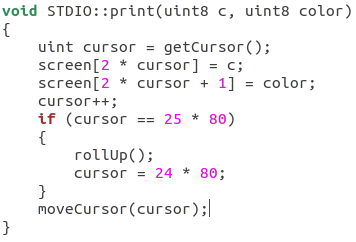
* 一个新创建的线程是如何被调度然后开始执行的。
* 一个正在执行的线程是如何被中断然后被换下处理器的，以及换上处理机后又是如何从被中断点开始执行的。

1. 将线程调度算法修改为下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。

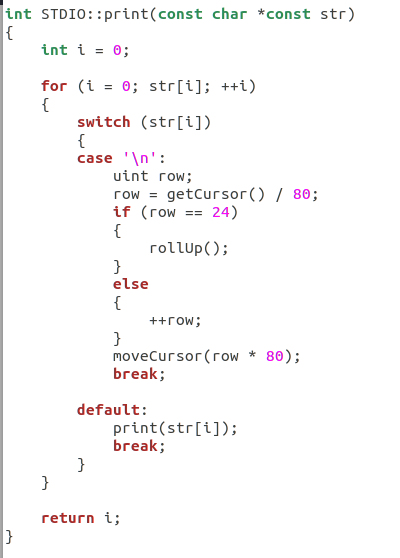
* 先来先服务。
* 最短作业（进程）优先。
* 响应比最高者优先算法。
* 优先级调度算法。
* 多级反馈队列调度算法

1. **实验过程**
2. **实现printf**

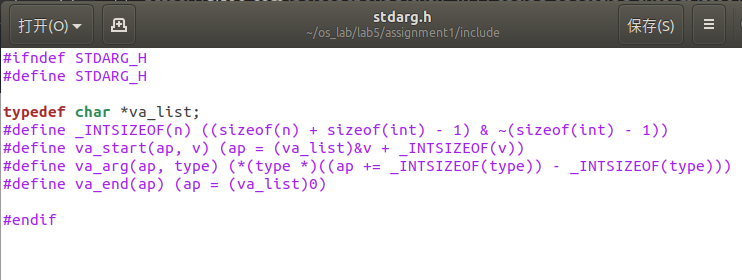
* 第一步，实现print一个字符的函数，在当前光标处打印一个字符，并将光标加一，注意光标在最后一个位置时要进行滚屏



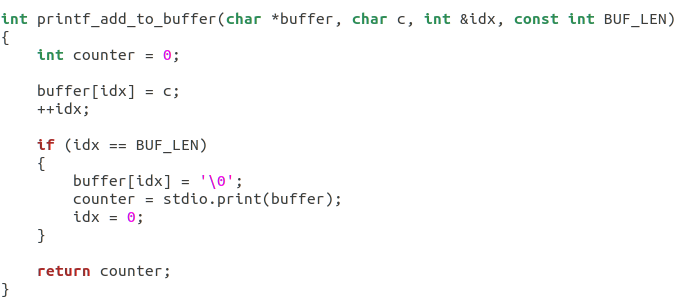
* 第二步，实现print一个字符串的函数，对字符串每个字符进行遍历，当遇到'\n'时换行，其他情况直接输出该字符



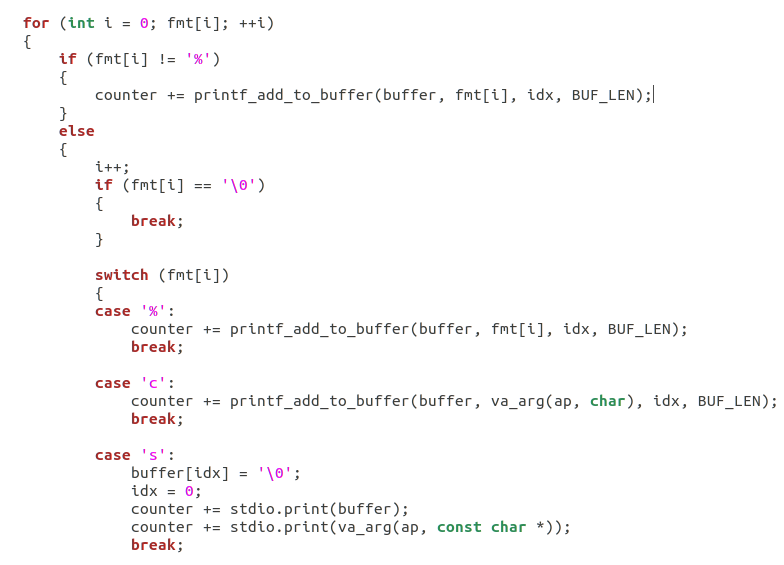
* 第三步，在include中编写stdarg文件



* 第四步，编写一个add\_to\_buffer函数，将要打印的字符放到缓冲区，等到缓冲区满或者遇到特殊条件时才输出



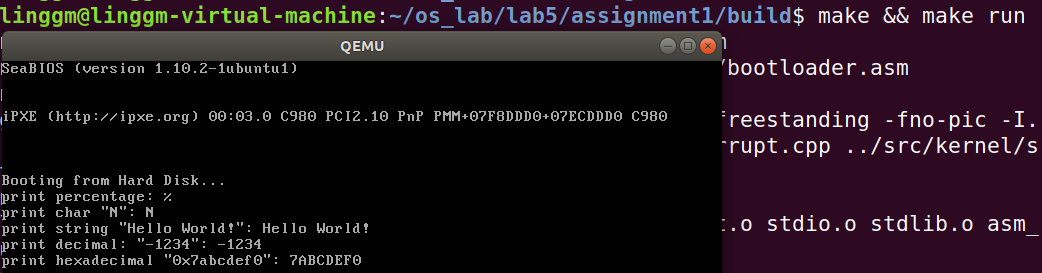
* 第五步，实现printf

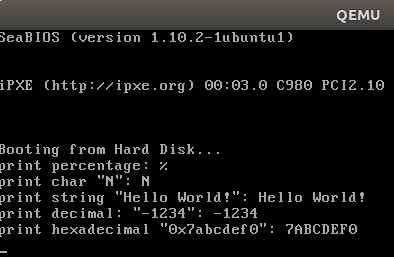


如果遇到普通字符，则直接输出；如果遇到%，则看%的下一个字符是什么。

如果 % 的下一个字符是'\0'，则退出；如果是'c'，则以字符的格式返回一个可变参数列表中的参数，并将其加入缓冲区；如果是's'，则输出并清空缓冲区，并以char\*的格式返回一个可变参数列表中的参数，并直接打印；如果是'd'，则以int的格式返回一个可变参数列表中的参数，转换为字符串后加入缓冲区；如果是'x'，则以int的格式返回一个可变参数列表中的参数，进行进制转换，然后转换为字符串，加入缓冲区；

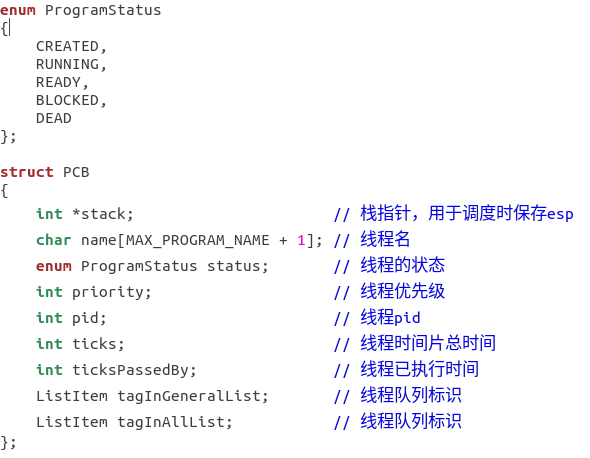
* 最后，在setup\_kernel.cpp中加入教程的测试语句，并修改makefile，make && make run运行qemu，得到结果如下



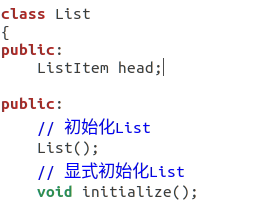


1. **实现PCB和线程创建调度**

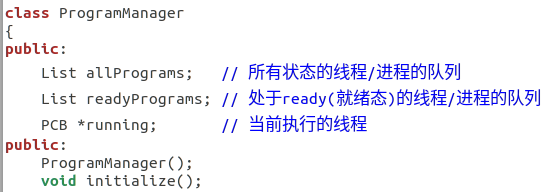
* 第一步，声明PCB结构体，成员有栈指针，线程名，线程状态，优先级，pid，已执行时间等

****

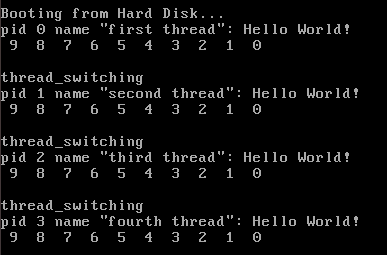
* 第二步，声明list类，是带头结点的双向链表

****

* 第三步，编写“程序管理器”类，成员有所有线程的队列、就绪线程的队列、当前执行的线程，实现创建线程、分配PCB、回收PCB、进行线程调度的功能（“程序管理器”类的具体实现见“关键代码说明”部分）

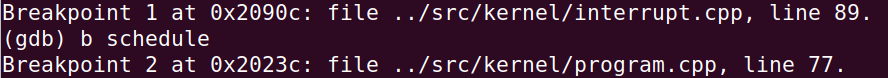
****

* 最后，加入教程的测试语句，make && make run运行qemu，得到结果如下

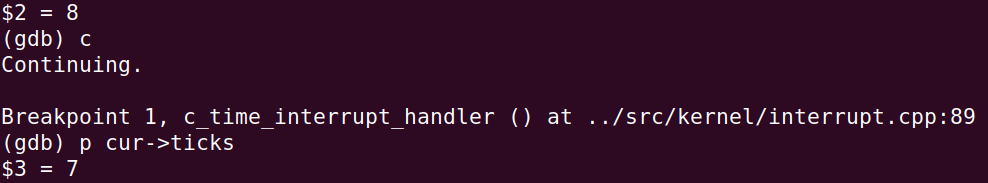


1. **对任务二的内容进行debug，查看线程调度时的信息**

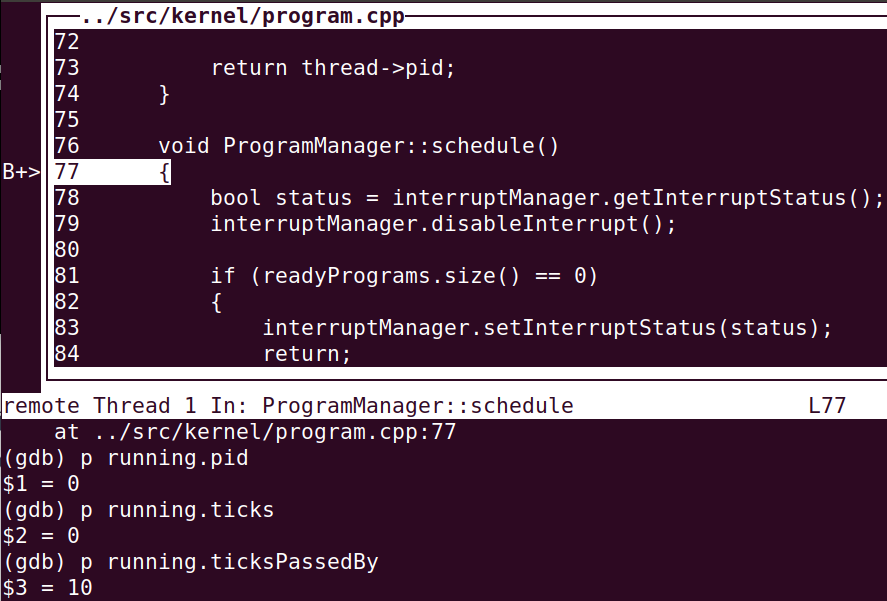
* 第一步，在中断处理函数和调度函数入口设置断点



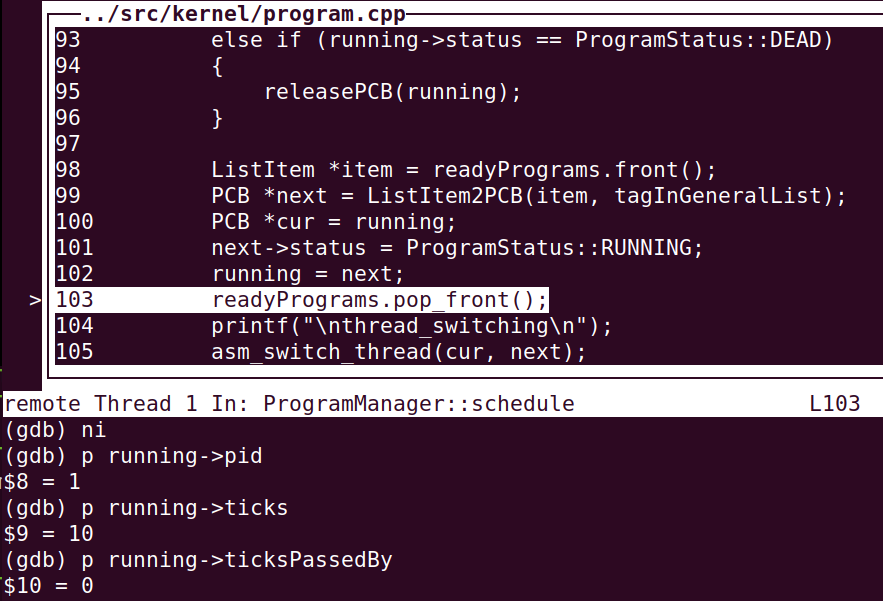
* 时钟中断发生时，查看ticks的信息，发现每次时钟中断ticks-=1，正确



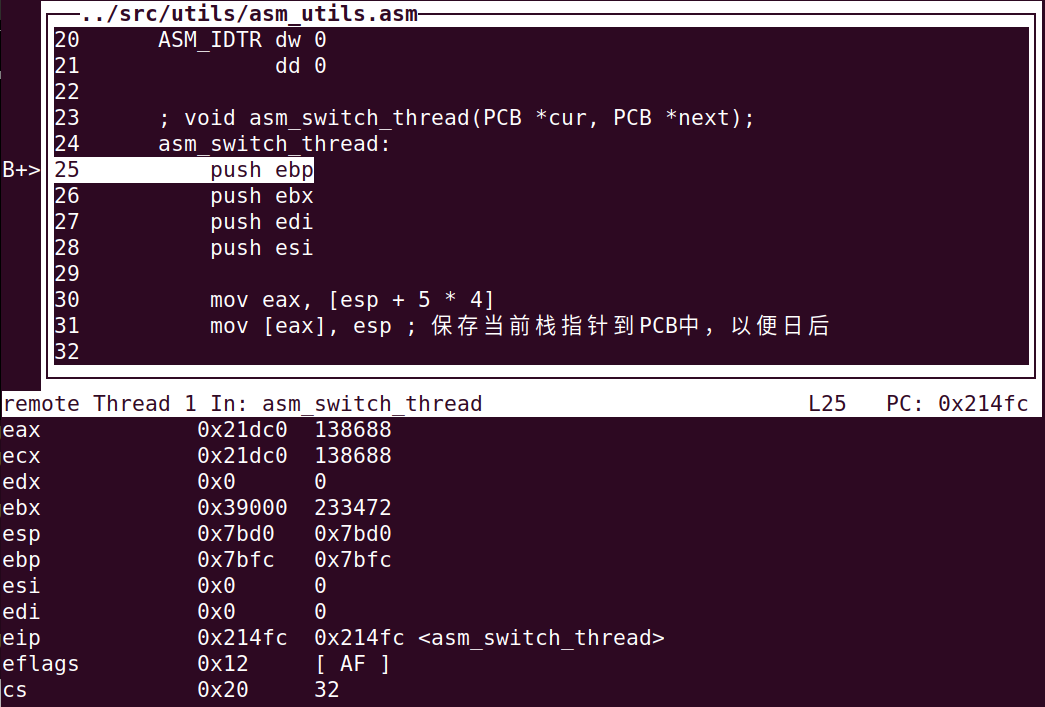
* ticks为0时，发生调度，查看首次调度发生时，pid为0，ticks为0



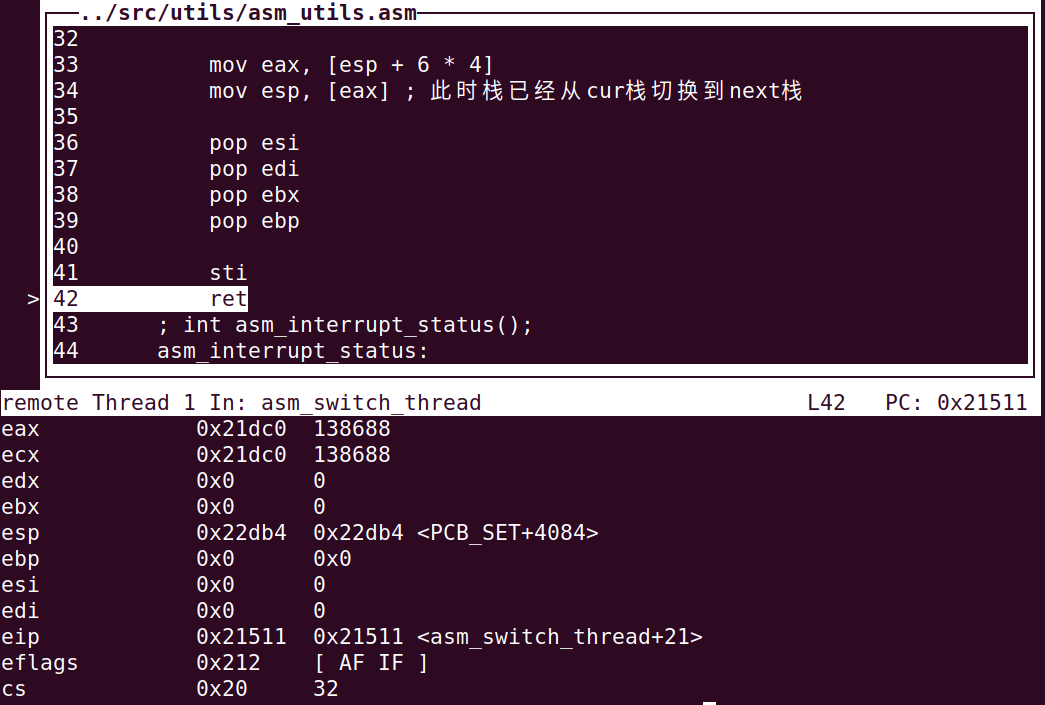
* 切换running的线程为下一个线程后，running的pid变为1，完成调度



* asm\_thread\_switch前查看寄存器信息



* asm\_thread\_switch后查看寄存器信息

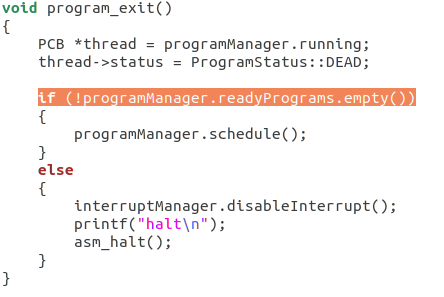


1. **修改任务二中的代码，实现先来先服务的调度算法**

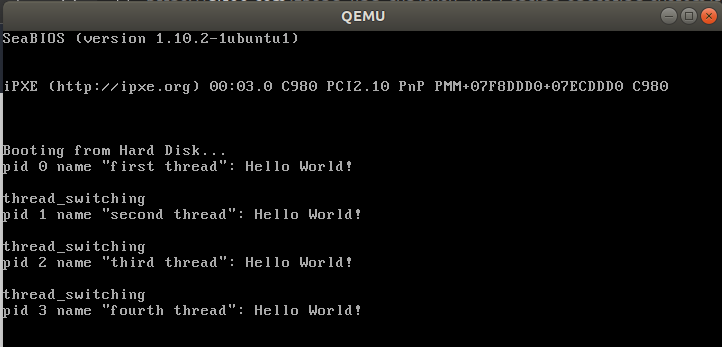
* 第一步，修改first\_thread，second\_thread和third\_thread，函数的最后不挂起，这样线程才能执行完毕，从而调度下一个线程



* 第二步，修改中断处理函数，时钟中断发生时，不做任何事情
* 第三步，修改program\_exit函数，在某一个线程终止时，如果就绪队列非空（有线程处于ready状态），则进行一次调度；否则挂起



* **结果**

****

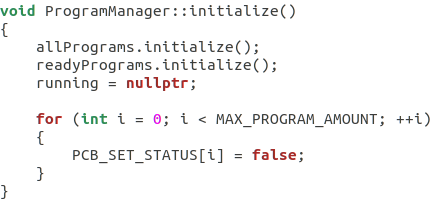
1. **关键代码**

任务一的代码在“实验过程”中作了展示和说明，不赘述

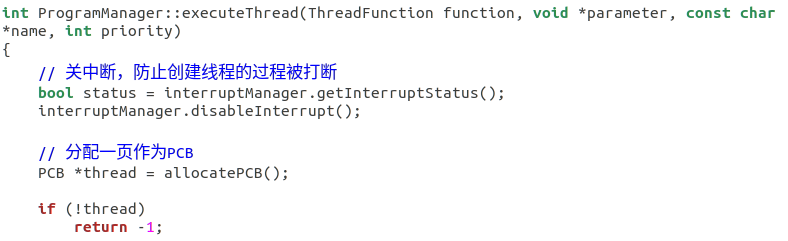
任务四的代码与任务二的代码大同小异，不同点在“实验过程”中

下面主要介绍任务二的代码：

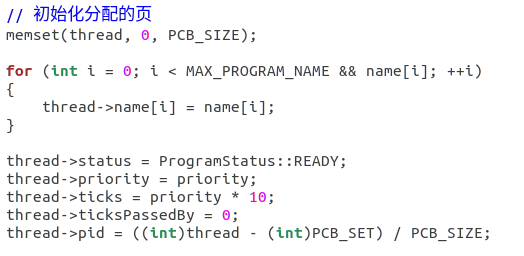
1. **程序管理类初始化**，先初始化两个队列，然后将running置空指针，将PCB的状态置为“未分配”



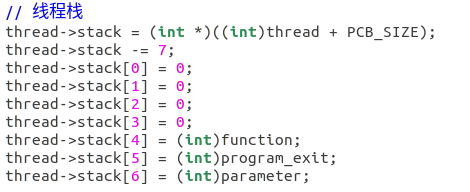
1. **创建线程，**先关中断，然后分配PCB



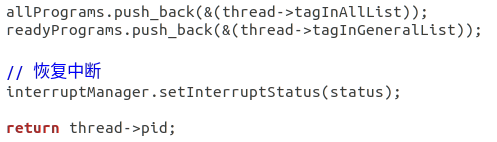
* 再初始化分配的页，全置0，然后初始化名字、状态、优先级等



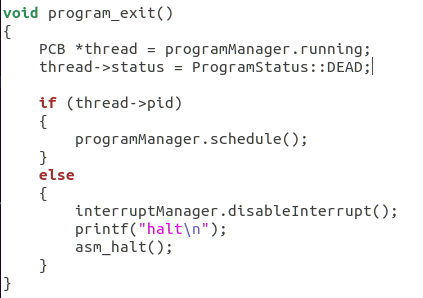
* 然后对线程栈进行处理



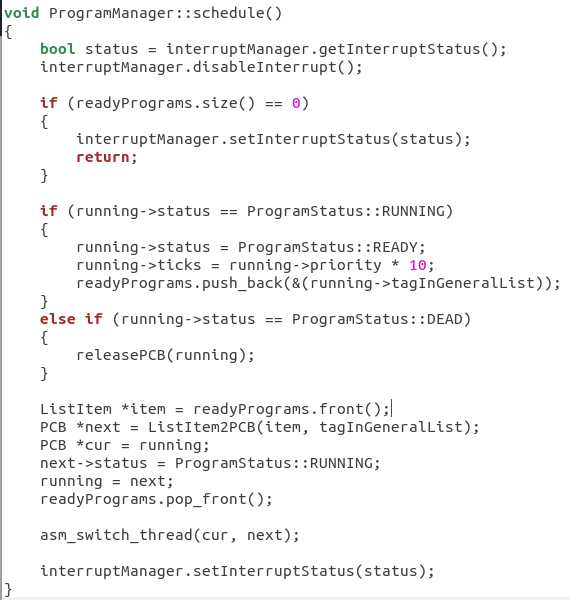
* 最后在两个队列中push新的线程，恢复中断，返回



1. **线程exit函数：**将当前running的线程的状态置dead，如果不是主线程，则调度；如果是主线程，则halt挂起

****

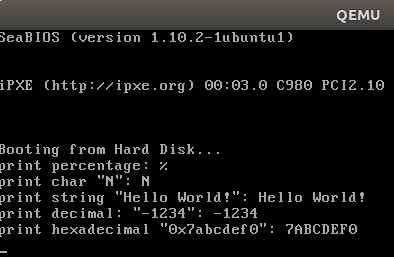
1. **线程调度函数：**如果就绪队列里没有线程（没有就绪线程）则原封不动返回；如果有就绪的线程，而且当前线程处于running状态，则置当前线程为ready，入就绪队列，将就绪队列头部的线程置running态；如果有就绪的线程，而且当前线程处于dead状态，则释放当前线程的PCB，将就绪队列头部的线程置running态，运行。

****

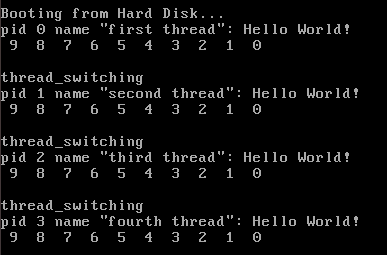
1. **interrupt\_handler函数**：每次时钟中断，将ticks--，ticks为0时调度，这实现了RR调度算法

****

1. **实验结果**
2. **Assignment1**

****

1. **Assignment2**

****

1. **Assignment3（debug信息详见过程部分）**
2. **Assignment4**

****