###### 2. 调试分析 Linux 0.00 多任务切换

2.1. 实验目的

通过调试一个简单的多任务内核实例，使大家可以熟练的掌握调试系统内核的方法；

掌握Bochs虚拟机的调试技巧；

通过调试和记录，理解操作系统及应用程序在内存中是如何进行分配与管理的；

2.2. 实验内容

通过调试一个简单的多任务内核实例，使大家可以熟练的掌握调试系统内核的方法。 这个内核示例中包含两个特权级 3 的用户任务和一个系统调用中断过程。我们首先说明这个简单内核的基本结构和加载运行的基本原理，然后描述它是如何被加载进机器 RAM 内存中以及两个任务是如何进行切换运行的。

2.2.1. 掌握Bochs虚拟机的调试技巧

直接cv的实验1中做完的内容

·如何单步跟踪？

输入s命令即可单步执行，每次输入 s 命令，虚拟机将执行当前指令并停在下一条指令之前。

·如何设置断点进行调试？

使用 b +地址命令设置断点，再点击运行，bochs就会在断点位置停止

·如何查看通用寄存器的值？

调试界面左边蓝色部分：



·如何查看系统寄存器的值？

调试界面左侧紫色部分：



·如何查看内存指定位置的值？

使用如下命令（x）：  
 x /<count><format> <address>

# <count>：指定要查看的数据项数量。

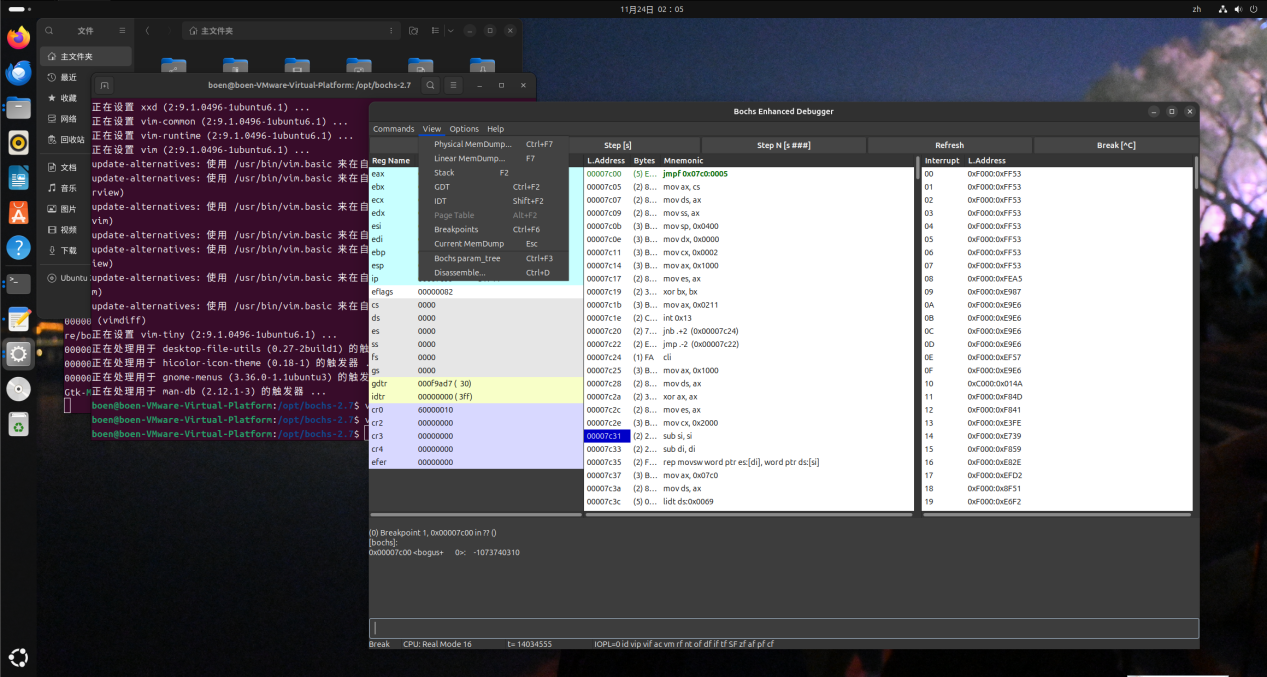
# <format>：指定数据的显示格式，如 b（字节）、w（字）、d（双字）等。

# <address>：指定要查看的内存地址。

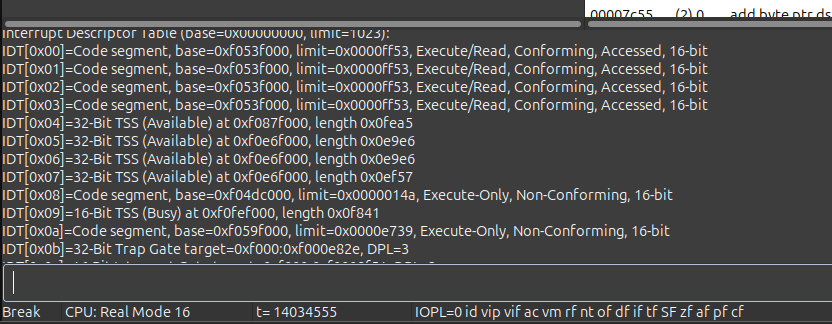
X /1wd 0x7c00:  
 18b4061fdbefcfeac008bbb6c204641

·如何查看各种表，如 gdt ，idt ，ldt 等？

点击调试界面左上角的 view ，即可选择查看 GDT ， IDT ， STACK 等，打开后其内容显示在调试器右边部分的界面，LDT ， TSS 可以从 GDT 中得到基址间接查看。

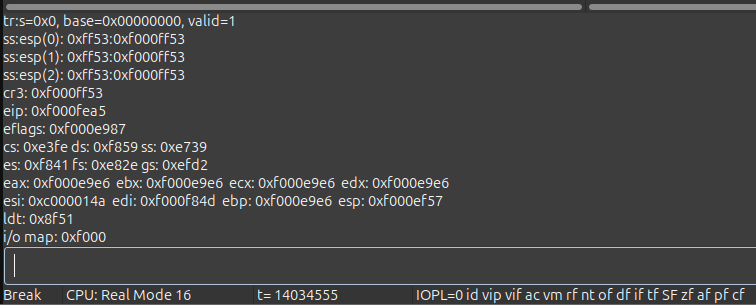


可以使用info+表名查看表中的内容（info idt（小写））：



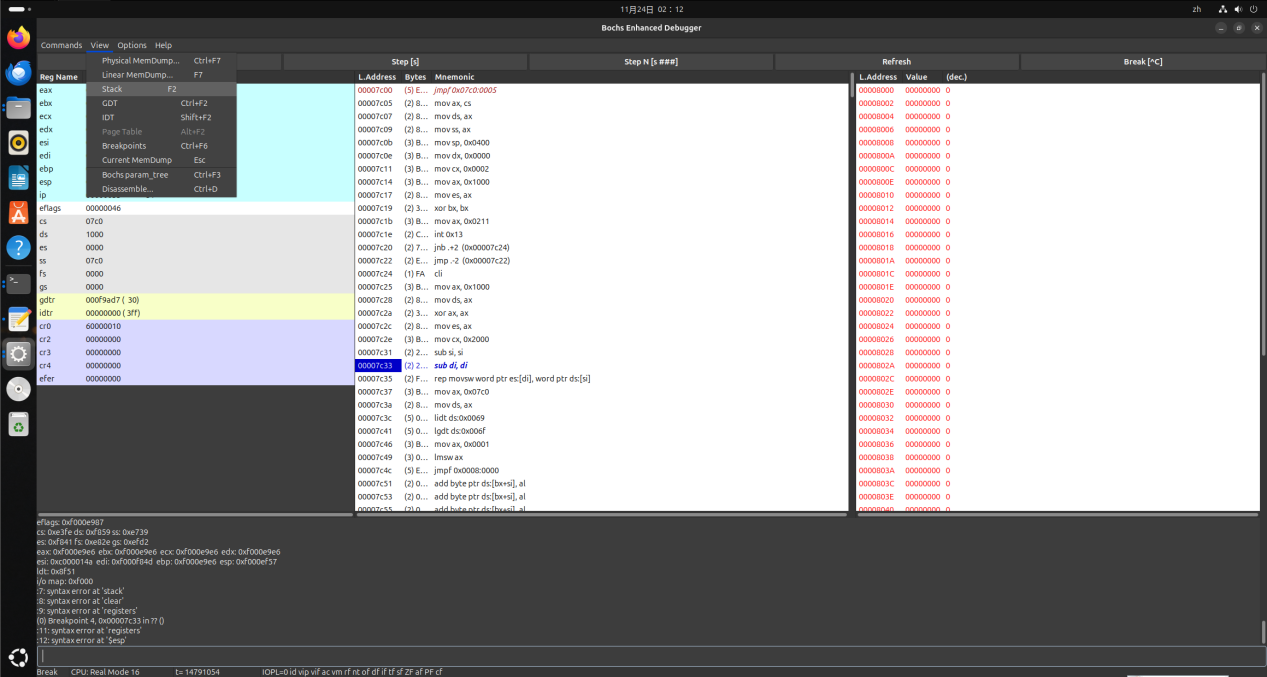
·如何查看 TSS？

同上 info tss（小写）：



·如何查看栈中的内容？

同同上上，在断点时，使用view中的stack查看：



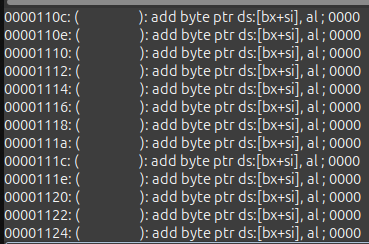
·如何在内存指定地方进行反汇编？

使用disasm命令：  
 disasm <address> <count>

# <address>：指定要反汇编的内存地址。

# <count>：指定要反汇编的指令数量。

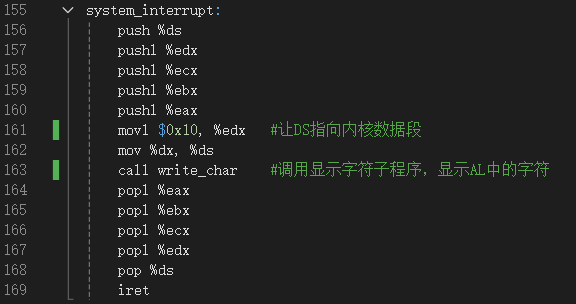
disasm 0x1234 10:



2.3. 实验报告

通过仔细的调试与跟踪程序，完成以下任务：

·当执行完 system\_interrupt 函数，执行 153 行 iret 时，记录栈的变化情况。

首先阅读理解system\_interrupt中的内容：  
 

iret用于在处理器状态转移期间从中断或异常处理程序返回到被中断的程序，还原被中断程序的执行环境，包括寄存器、堆栈以及特权级别的状态。

iret 指令执行以下操作：

从堆栈中弹出 EIP寄存器的值，以恢复中断或异常处理程序返回到的下一条指令的地址。

从堆栈中弹出 CS 寄存器的值，以恢复中断或异常处理程序返回到的代码段。

从堆栈中弹出标志寄存器 EFLAGS的值，以恢复标志寄存器的状态。

如果在中断或异常处理程序执行期间切换了堆栈，iret 会从堆栈中弹出新的 ESP寄存器的值，以恢复原始堆栈。

根据从堆栈中弹出的 CS 寄存器的值，恢复到适当的特权级别。这允许在不同特权级别（如内核态和用户态）之间切换。

进入bochs，调试找到iret的地址为0x17，在0x17c处设置断点，后运行至断点处：  
 如下图，可以观察到在执行iret（也就是0x17c）之前的栈状态：

CPU 正在执行的下一条指令的内存地址EIP：0x17c（即将执行iret）

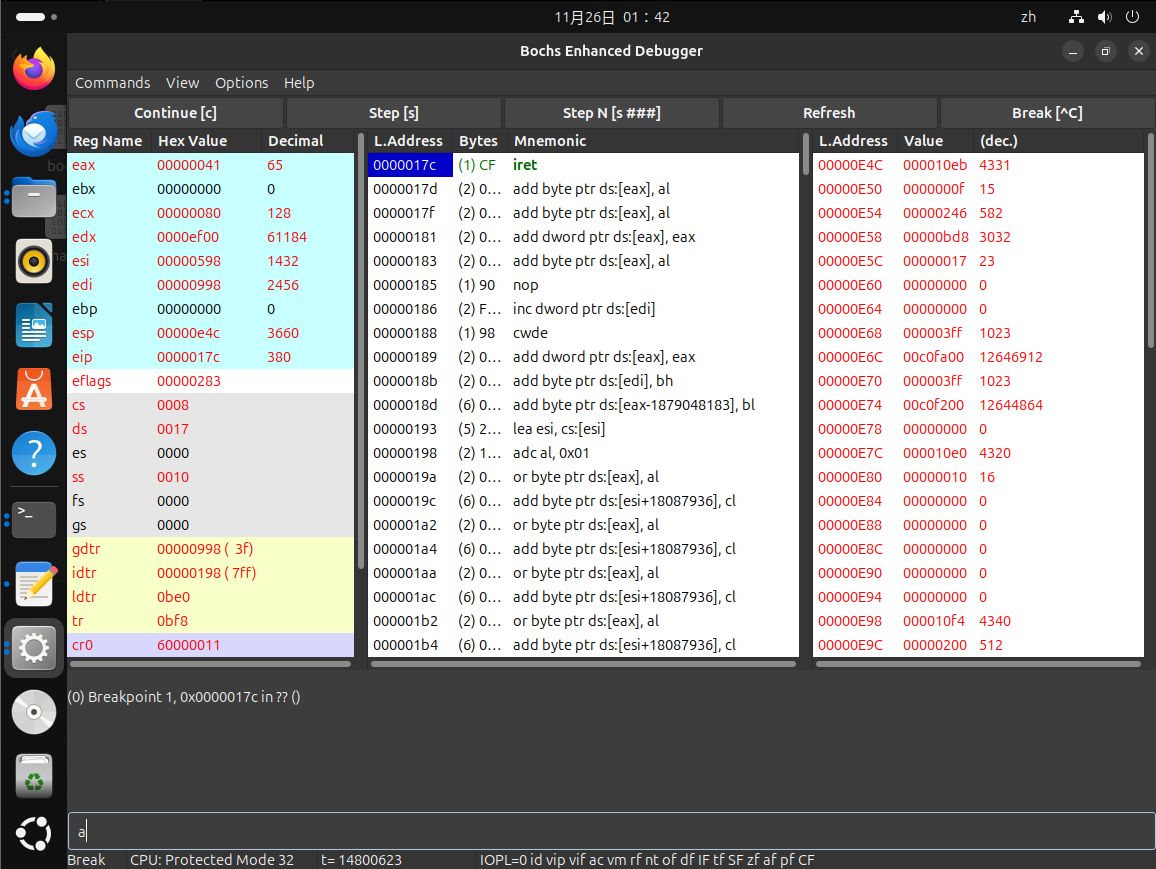
代码段寄存器CS：0x8（当前执行的代码段以及执行代码的特权级别为 0 内核态）

栈顶指针ESP：0xe4e

此时stack中的值从栈顶开始是：

0x10eb（即将弹出的EIP寄存器的值） 0x000f （即将弹出的CS寄存器的值）

0x0246（即将弹出的EFLAGS寄存器的值） 0x0bd8（即将弹出的ESP寄存器的值）



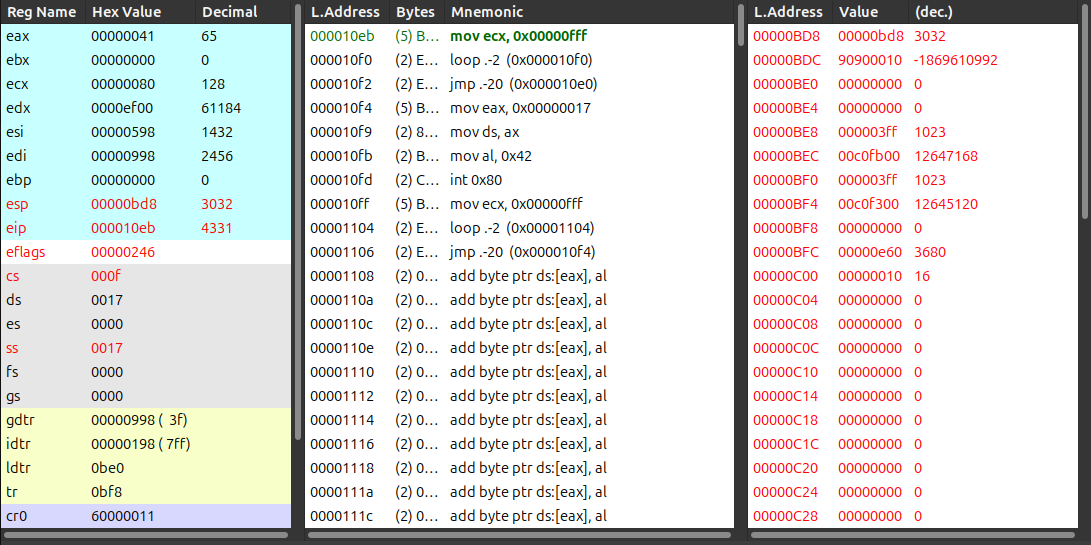
单步运行，检查执行iret命令之后cpu状态：

执行命令之后，如下图，栈的变化情况为：

CPU 正在执行的下一条指令的内存地址EIP：0x10eb

代码段寄存器CS：0xf（当前执行的代码段以及执行代码的特权级别为 3 用户态）

栈顶指针ESP：0xbd8



·当进入和退出 system\_interrupt 时，都发生了模式切换，请总结模式切换时，特权级是如何改变的？栈切换吗？如何进行切换的？

1）特权级

在进入和退出 system\_interrupt 时发生了特权级的模式切换。

进入 system\_interrupt：

用户态（特权级 3）切换到内核态（特权级 0），中断服务在内核态中执行。

处理器将特权级从 3 切换到 0，CS的值由 0xf 变为 0x8。

退出 system\_interrupt：

内核态（特权级 0）切换回用户态（特权级 3）。

iret执行之后，处理器将特权级从 0 切换到 3，CS的值由 0x8 变为 0xf。

2）栈

在进入和退出 system\_interrupt 时同时会发生栈的切换。

进入 system\_interrupt：

发生堆栈切换，以便在内核态中使用内核堆栈，来保护用户堆栈的完整性。

退出 system\_interrupt：

发生堆栈切换，以返回到原特权级的堆栈，并继续用户态的执行。

3）如何切换

这种特权级的切换和堆栈切换是操作系统内核和处理器硬件协同工作的结果。

保存当前上下文：在模式切换之前，当前执行的任务的上下文以及当前特权级需要被保存压入栈中

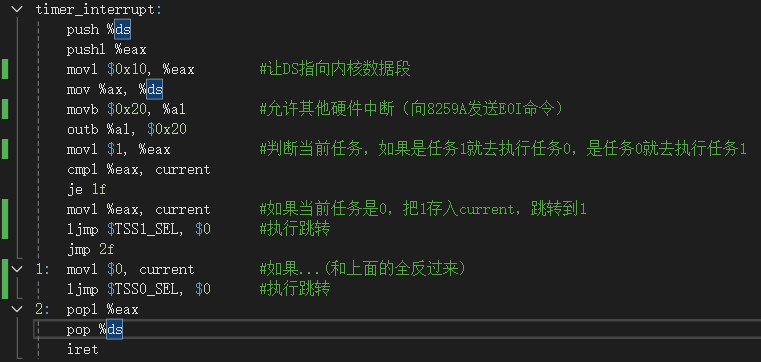
选择新特权级别：根据要切换到的特权级别，操作系统选择新的代码段描述符和堆栈描述符，切换堆栈和特权级，并将它们加载到相应的寄存器中。

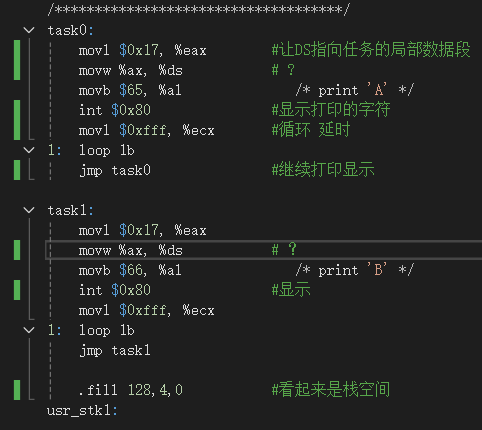
在进入内核态时，通常会使用内核堆栈，以避免破坏用户堆栈。

在退出内核态时，特权级和堆栈状态会恢复，以确保程序的正常执行。

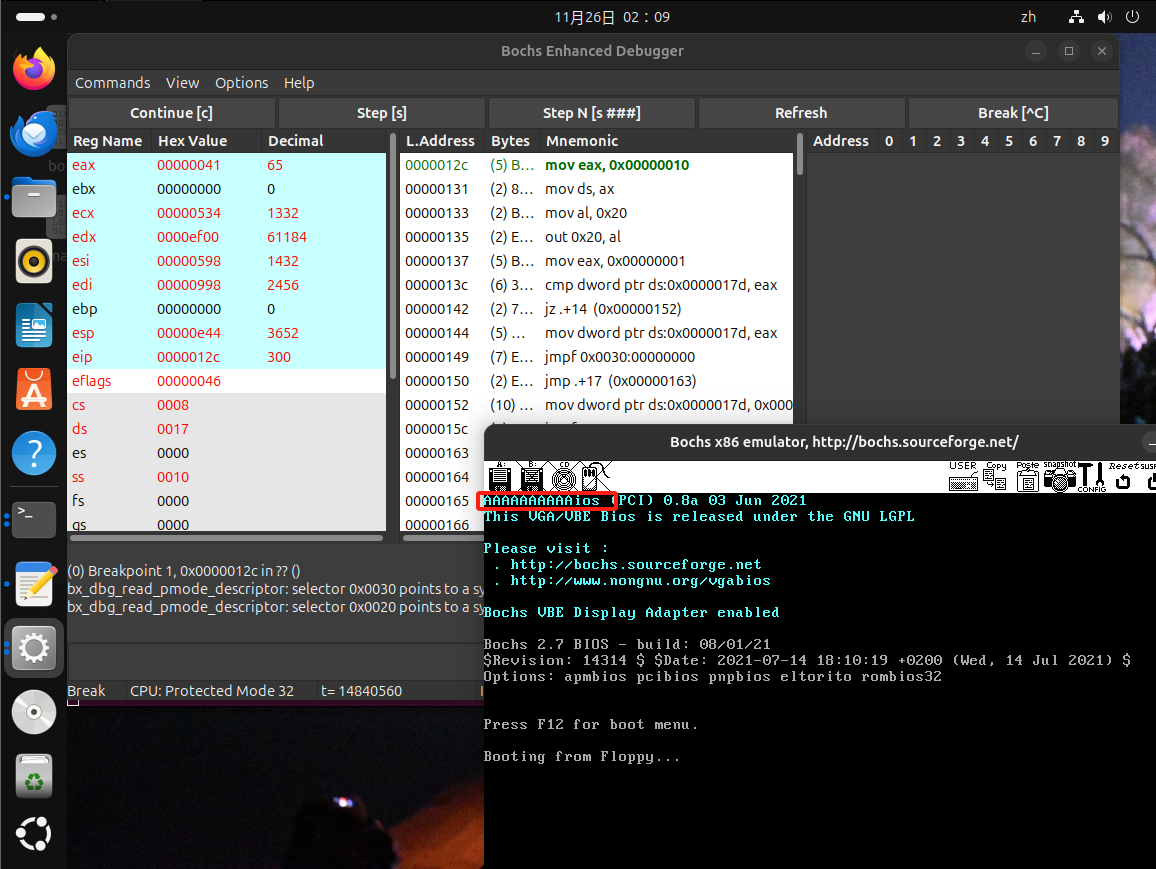
·当时钟中断发生，进入到 timer\_interrupt 程序，请详细记录从任务 0 切换到任务 1 的过程。

首先阅读理解timer\_interrupt中的内容：

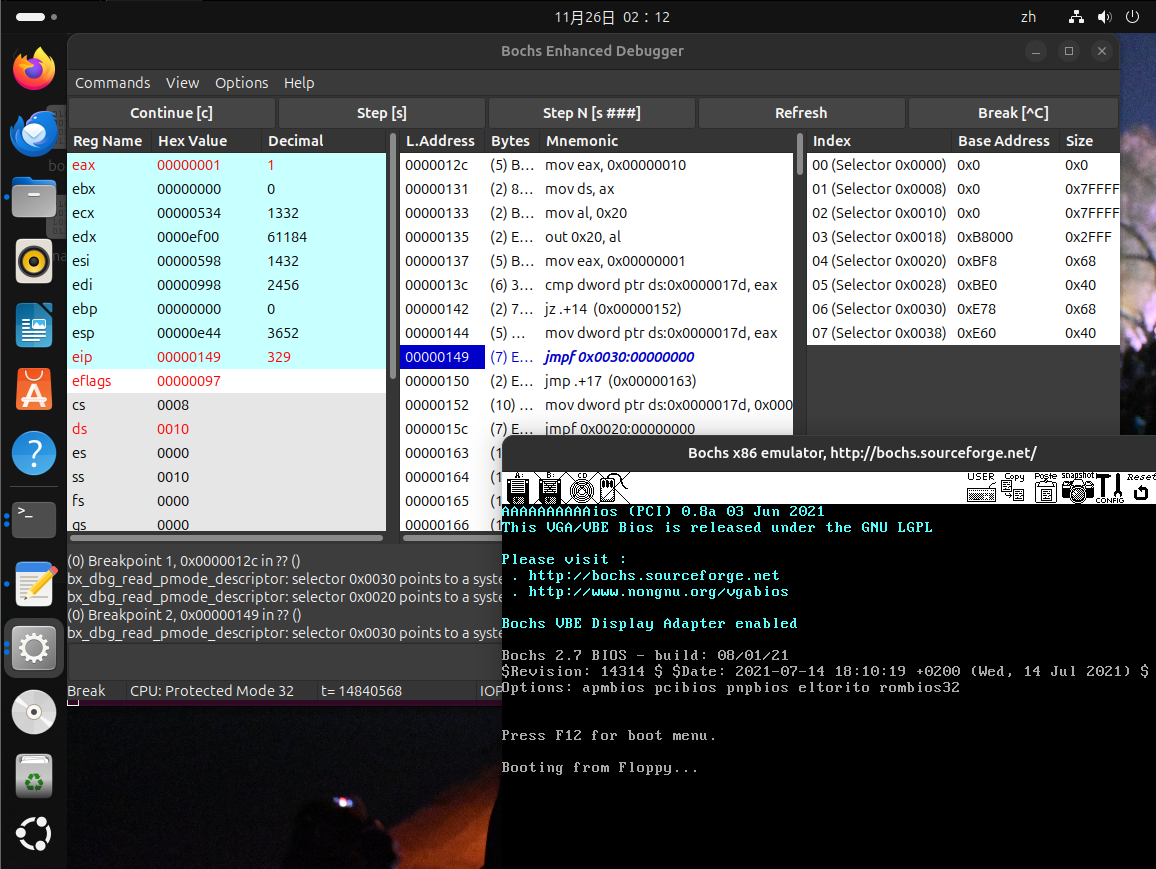


发现有task0和1的出现，观察发现task0和1分别是打印A和B（字母），如下图：  
 

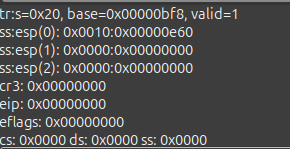
推测是先将任务0的执行结果存储到寄存器中，再运行任务1，先在首部（0x12c处）设置断点观察运行结果：

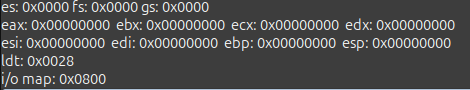


如图（框住的部分）可知程序进行了A的打印但是没有打印B，也就是只执行了task0还未执行task1，单步运行到CS:EIP 0x08:0x0149处准备执行指令jmpf 0x30:0 ，即远跳转到0x30:0，将一个 TSS 选择子（0x30）装入 CS，实际上就是为了将任务切换到这个 TSS，而0x30恰好是task1的TSS，如下图：

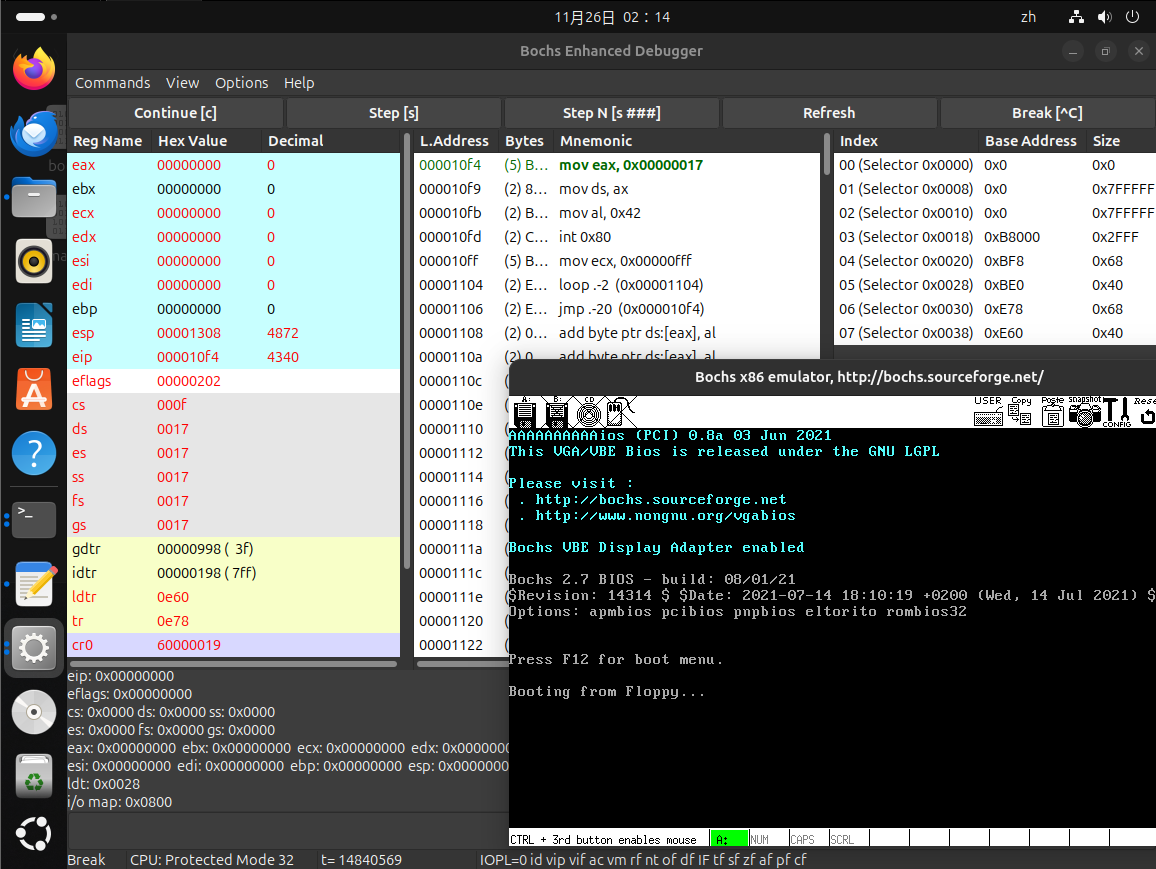


输入输入info tss，记录任务切换前task0 的TSS：

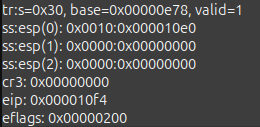


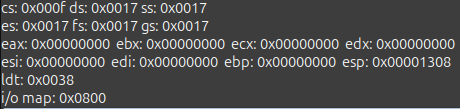


单步执行后，寄存器状态如下图CS:EIP 0xf:0x10f4，现在已经切换到任务1，由于任务1是第一次执行，直接跳转到了用户程序task1的入口处：

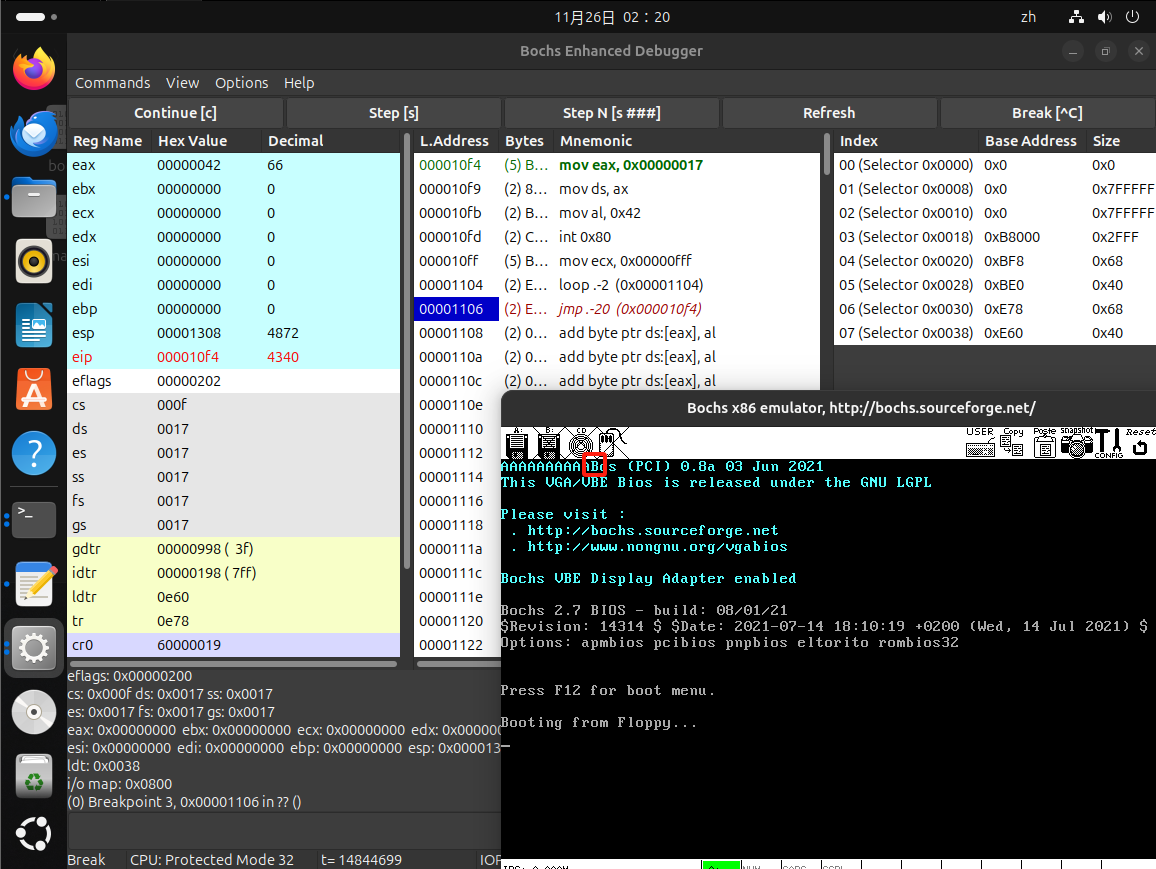


输入 info tss，查看此时的TSS，发现TSS变化为task1 的TSS，并且与寄存器中的值相同，这说明了任务切换时会根据对应任务的 TSS的各个字段修改寄存器。



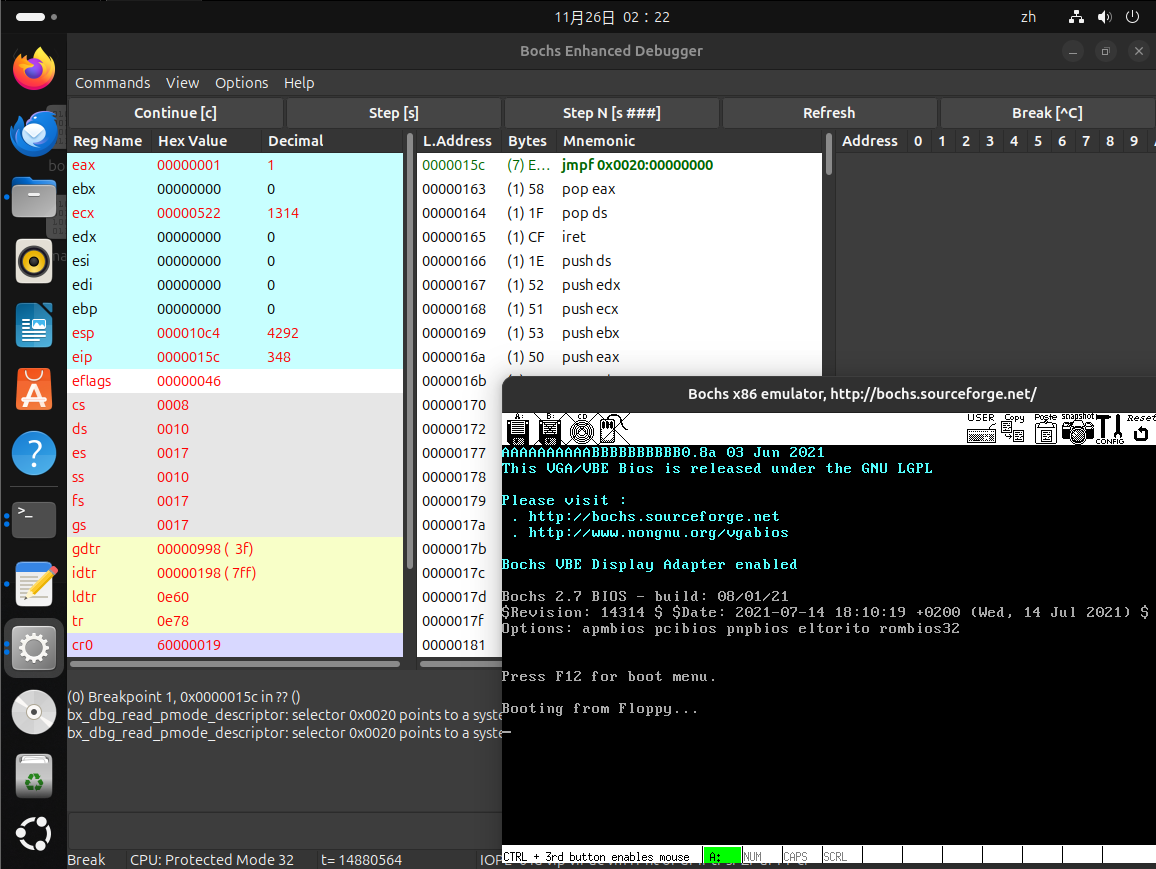


末尾设置断点，运行至断点后发现已经打印了B，再次单步运行，跳回10f4，与之前对代码的注释以及猜想相同。

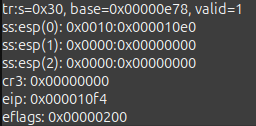
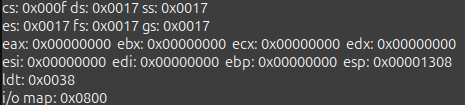


·又过了 10ms ，从任务1切换回到任务 0 ，整个流程是怎样的？ TSS 是如何变化的？各个寄存器的值是如何变化的？

在0x15c处设置断点，然后运行程序：

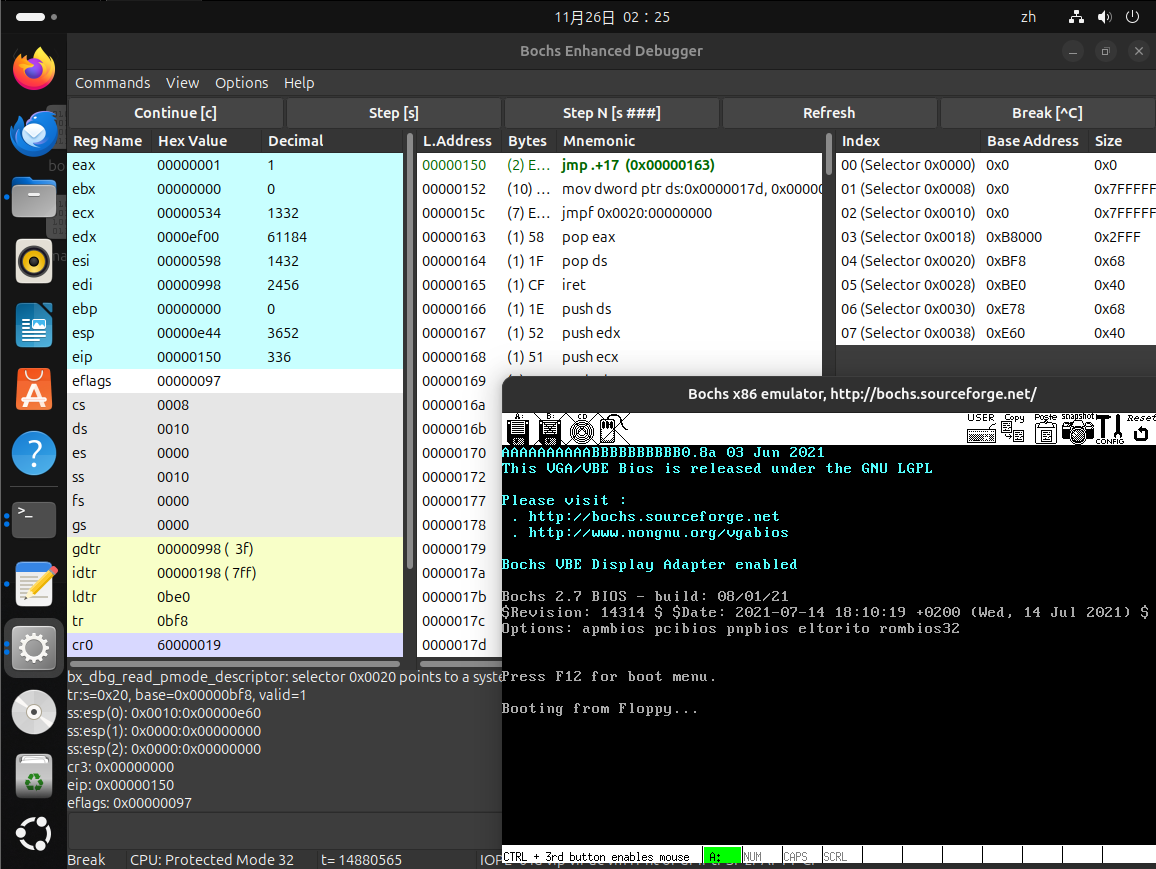


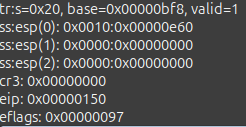
可以看到AB均被打印，可知 程序已经执行了task1，输入info tss检查TSS中内容：

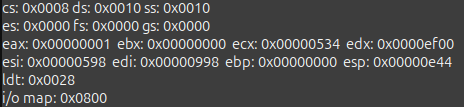
 

与寄存器内容相同，说明已经将 task1 的上下文保存在 task1 的TSS中。

然后点击单步运行，执行 jmpf 0x20:0 ，将一个 TSS 选择子（0x20）装入 CS，即task0 的TSS，由于第一次任务切换时将寄存器现场保存到了 task0 TSS里，因此将TSS切换回来后，CS:EIP 会指向第一次任务切换的下一条地址，也就是0x8:0x150，运行结果以及TSS信息如下：







·请详细总结任务切换的过程。

时钟中断触发：任务切换通常是由系统中的时钟中断timer\_interrupt 触发的。时钟中断以固定的时间间隔（每10毫秒）发生一次，它是多任务处理的触发点。

保存当前任务上下文：当时钟中断触发时，操作系统会执行时钟中断处理程序。在处理程序执行之前，当前正在执行的 taskA 的上下文会被保存。EAX、ECX、EDX、EFLAGS、ESP、CS、EIP等寄存器的状态会保存在 taskA 的 TSS 中。

选择下一个任务：在时钟中断处理程序中，操作系统会选择下一个要执行的 taskB ，任务B的上下文信息存储在 taskB 的TSS中。

加载下一个任务上下文：时钟中断处理程序通过 taskB 的 TSS 将 taskB 的上下文信息（寄存器的值）加载到处理器中，处理器现在准备执行 taskB 。

切换堆栈：如果 taskA 和 taskB 使用不同的内核堆栈，堆栈指针寄存器ESP将从 taskA 的内核堆栈切换到 taskB 的内核堆栈，以确保 taskB 可以正常执行内核代码。

特权级别切换：如果 taskA 和 taskB 属于不同的特权级别，时钟中断处理程序会执行特权级别切换操作。

taskB 开始执行： taskB 的上下文准备就绪并加载到处理器中， taskB 开始执行。

时钟中断返回：时钟中断处理程序执行完毕后，处理器返回到 taskB 的执行点， taskB 继续执行。

在这个过程中，TSS寄存器中的值将发生变化，以反映新任务的上下文。旧任务的上下文信息已经被保存，以便在未来的任务切换中恢复。其他寄存器的值也会根据任务上下文的不同而变化，以确保任务切换的正确执行。这个过程允许多个任务在同一个系统中轮流执行，实现多任务处理。

2.3.1. 评分标准

记录描述要详细完整，前4项每题 20%，总共 80%

总结任务切换过程，10%

格式规范美观，10%