

操作系统第二次作业

20373068 周宇光

- 1、假设有一个简单的计算机硬件系统，CPU 是同学们自己设计的，有内存和硬盘（可以没有 MMU），请设计一个尽量简单的启动过程。（要求：列出必要的硬件支持、启动软件的基本功能和启动过程，只要将控制权交给操作系统镜像就算完成启动）

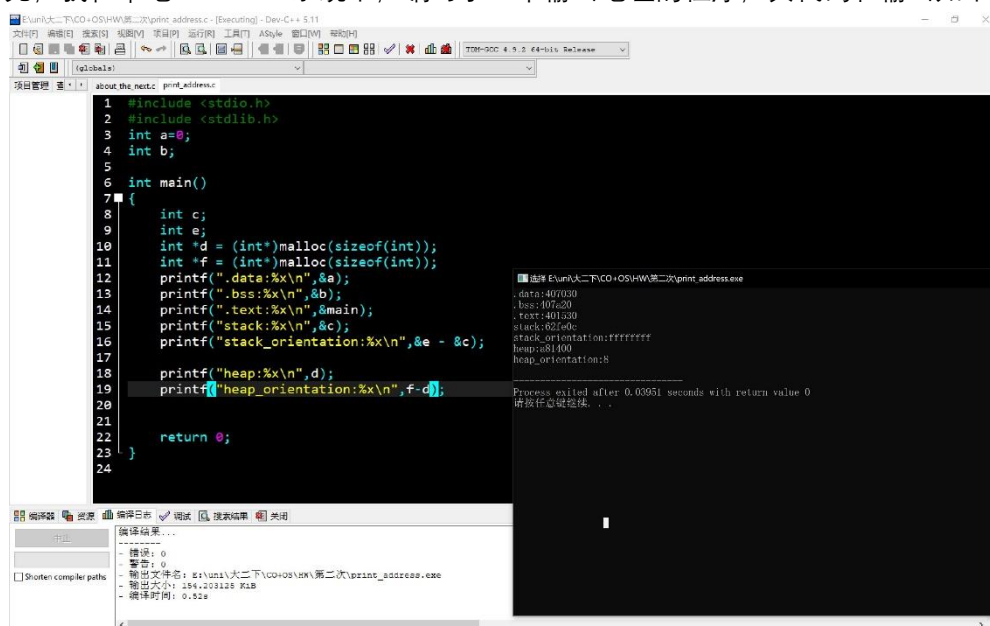
启动过程：首先，对系统加电。加电后，从硬盘的 MBR 中读取设备相关的启动信息、磁盘分区信息等。检查完 MBR 后，设计的 Booter 开始运行，由 MBR 信息初始化内存分布、各种 CP0 寄存器的值以及 CPU 参数。然后，由 loader 将操作系统内核的镜像映射到内存，并根据已有的内存分布设置好堆栈的位置。最后，由 Loader 跳转至操作系统内核的 C 程序入口。

硬件支持：包含 CP0 寄存器的 MIPS 流水线 CPU、内存、有 MBR 的硬盘

启动软件的基本功能：Booter 负责 stage1 的各种初始化工作；Loader 负责映像操作系统内核，跳转至 stage2 的操作系统

- 2、编写一段程序，分别输出属于该程序代码段、数据段、堆和栈的地址。（提示：不要求一定要输出各个段的首地址，不要编一个 ELF 解析程序，只要输出不同类型变量的地址就可以。）

首先，我在本地 windows 系统下，编写了一个输出地址的程序，其代码和输出如下：



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int a=0;
4 int b;
5
6 int main()
7 {
8     int c;
9     int e;
10    int *d = (int*)malloc(sizeof(int));
11    int *f = (int*)malloc(sizeof(int));
12    printf(".data:%x\n",&a);
13    printf(".bss:%x\n",&b);
14    printf(".text:%x\n",&main);
15    printf("stack:%x\n",&c);
16    printf("stack_orientation:%x\n",&e - &c);
17
18    printf("heap:%x\n",d);
19    printf("heap_orientation:%x\n",f-d);
20
21    return 0;
22 }
23
24
```

```
.data:407030
.bss:407e20
.text:401e30
stack:821e0c
stack_orientation:ffffffff
heap:681400
heap_orientation:8

Process exited after 0.03951 seconds with return value 0
请按任意键继续...
```

可以看到，各个代码段的分布。为了对比，我在 linux 的跳板机上也进行了这份代码的编译运行。结果如下：

```
git@20373068:~$ ./address
.data:702a014
.bss:702a018
.text:7027189
stack:c0317670
stack_orientation:1
heap:88c22a0
heap_orientation:8
git@20373068:~$ ls
0-29.jpg 20373068 address address.c objdump_trial
```

发现了一点我不大能理解的，就是为什么 linux 和 windows 的栈增长方向不一样？我怀疑可能是两个变量 c 和 e 编译时顺序的问题。

3、动态内存分配需要对内存分区进行管理，一般使用位图和空闲链表两种方法。128MB 的内存以 n 字节为单元分配，对于链表，假设内存中数据段和空闲区交替排列，长度均为 64KB。并假设链表中的每个节点需要记录 32 位的内存地址信息、16 位长度信息和 16 位下一节点域信息。这两种方法分别需要多少字节的存储空间？那种方法更好？

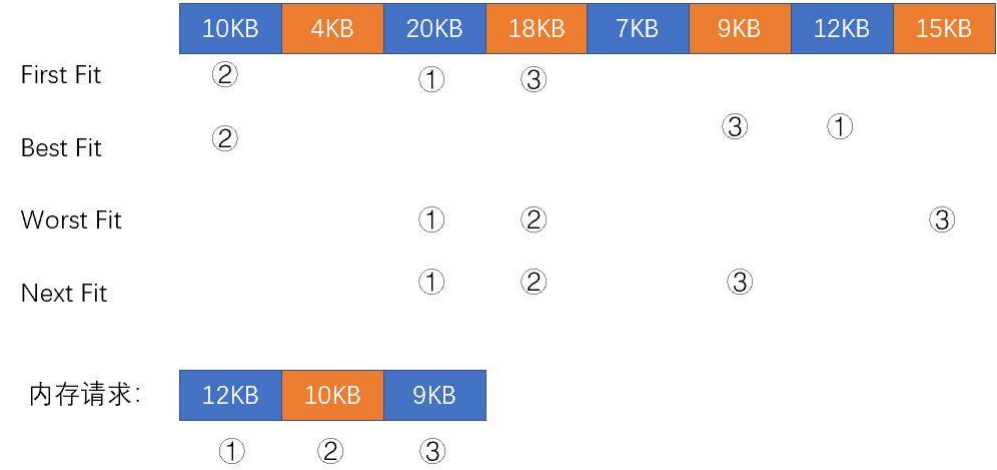
如果以位图管理，则需要一位来记录一个 B 的内存状况，对于 128MB 的内存空间，需要 16MB 的 bit-map。

如果以链表的方式来管理，则对于每一段都需要建立一个链表项，总计有 2K 段，每个单元的大小为 64 位=8B，因此需要 2KB 的内存空间。

显然，链表管理法的效果更好，因为内存空间以 64KB 为单位，足够连续。如果位图方法一位记录更多信息，比如一位记录一个物理页（4KB）大小的使用情况，那么位图法的效率会高很多。

4、在一个内存系统中，按内存地址排列的空闲区大小是: 10KB、4KB、20KB、18KB、7KB、9KB、12KB 和 15KB。对于连续的内存请求: 12KB、10KB、9KB。使用 FirstFit、BestFit、WorstFit 和 NextFit 将找出哪些空闲区？

匹配情况如下图所示



5、解释逻辑地址、物理地址、地址映射，并举例说明。

逻辑地址是 CPU 使用的地址, 在 32 位 MIPS 处理器中, 逻辑地址的范围是 0x00000000-0x7fffffff;

物理地址是某个具体的物理设备中, 存储单元的地址。

地址映射是指逻辑地址和物理地址之间的映射关系。比如在 kseg0 区, 逻辑地址 0x80000000-0x9fffffff 被映射到了物理地址 0x00000000-0x1fffffff 中。