Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

内存分段和常见段错误

袁晓筱

2018年7月22日

目录

内存分段

相关知识

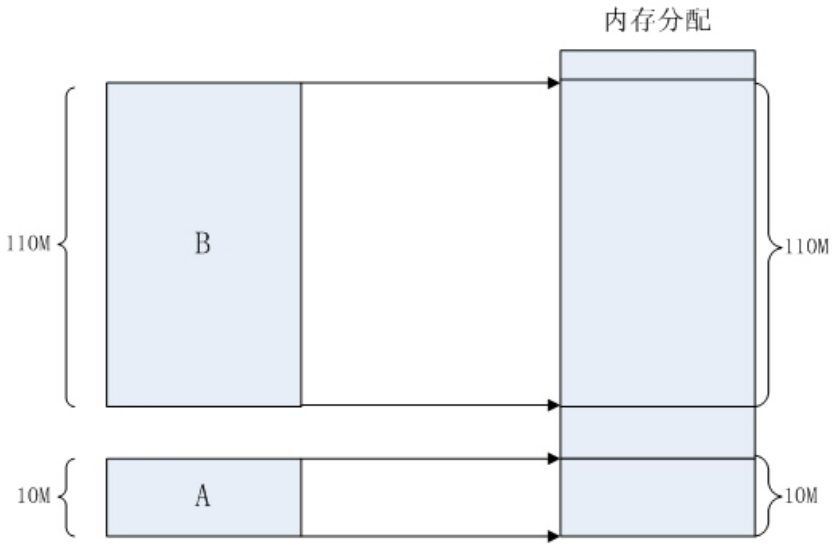
- 程序和内存的关系
  - 程序在运行时，由操作系统将可执行文件载入到计算机的内存中，成为一个进程。进程被创建时，系统就会为其分配内存空间
- 寻址
  - 程序中部分指令的执行依赖于对其他资源的查找，也就是寻址

```c
# include "stdio.h"

int main(void){
    int a = 10;
    int b = a;

    printf ("a address: %x\n", &a);
    printf ("b address: %x\n", &b);

    return 0;
}
```

```
a address: 601040
b address: d4a8fd4
```

为什么要分段

- 早期的内存分配机制
  - 将要运行的程序全部装入内存，直接在内存上运行,访问实际的物理地址



  - 问题：
    - 进程地址空间不隔离，没有权限管理
    - 内存使用效率低
    - 程序运行地址不确定
- 使用虚拟存储器可以解决上述问题
- 分段是虚拟存储器实现内存管理的一种方式

虚拟存储器

- 它是计算机系统内存管理的一种技术，它为每个进程提供了一个大的、一致的、私有的地址空间
- 创建进程时，32位的操作系统会为该进程分配一个4GB大小的虚拟地址空间
- 虚拟存储器内部的逻辑地址都从0开始
- 程序访问一个虚拟地址，操作系统将这个虚拟地址映射到适当的物理内存上，操作系统将保证不同的程序最终访问的内存地址位于不同的区域，彼此没有重叠，就可以达到内存地址空间隔离的效果
- 虚拟地址与物理地址之间的映射方式可以分为两种：
  - 分段
  - 分页

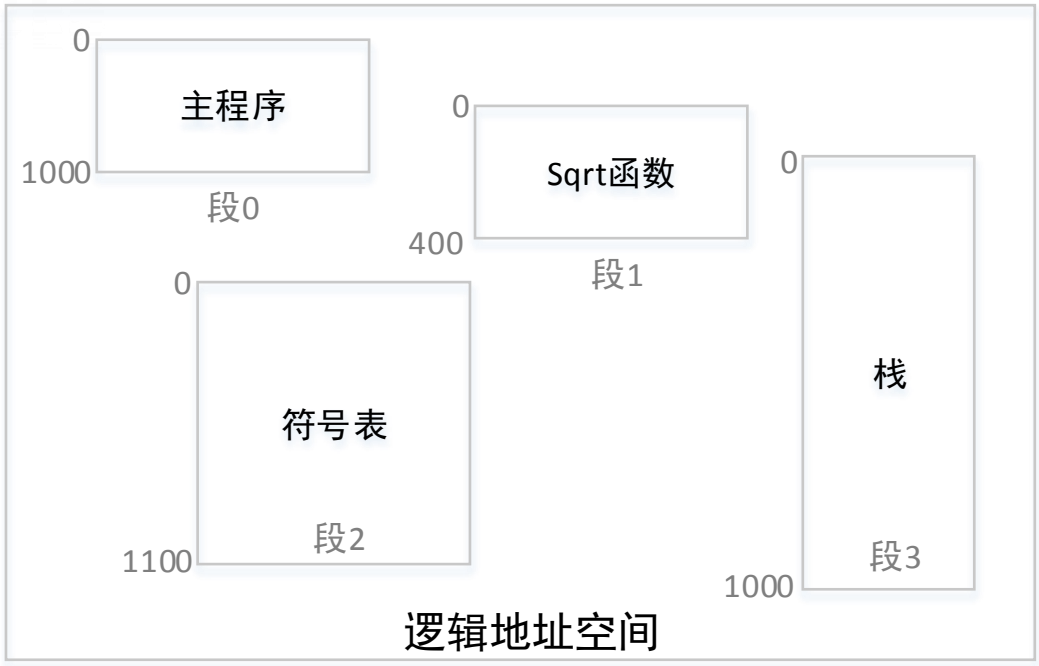
分段

- 程序按模块划分，主存按段分配。段与段在内存中可以不相邻接，段内地址连续
  - 例如一段main函数，里面包含Sqrt函数调用，这段代码可以分为：主函数代码段、Sqrt函数代码段、栈、符号表等。

分段

- 逻辑地址空间中，每个段都从0开始编址
- 可执行目标程序由各个模块构成，链接程序将各模块的逻辑地址构成一个逻辑地址空间。逻辑地址空间由一组段组成
- 每个段都有名字和长度，因此通过两个量来指定地址：段名称和偏移。段名称通过段编号来指定。因此逻辑地址由有序对构成：<段编号s, 段内偏移d>
- 每一道程序（或一个进程等）由一张段表控制，每个程序段在段表中占一行
- 虚拟地址是由段和偏移量组成的

分段



逻辑地址空间

	基址	界限	有效位
0	1000	1400	
1	400	6300	
2	1100	3200	
3	1000	4700	

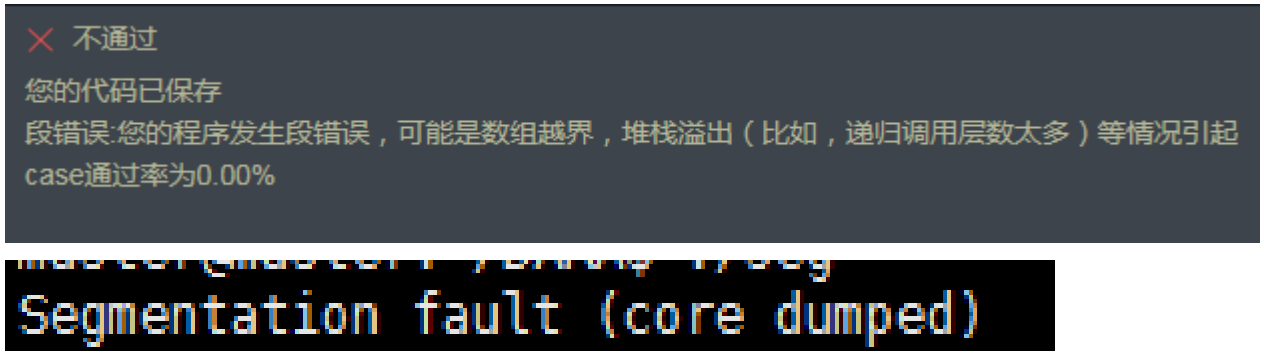
段表

物理内存

常见段错误

段错误(Segmentation fault)

- Segmentation fault译为存储器段错误、存储器区块错误或访问权限冲突
- 分段机制将进程之间和进程内部的不同数据段之间隔离起来。当程序内存的数据的访问超出了系统所给这个程序数据段的范围，就会引发段错误，系统就会给进程发送一个信号SIGSEGV，程序将终止退出
- 当程序试图访问不允许访问的存储器位置或尝试以不允许的方式访问存储器位置时，会发生段错误



常见段错误

- 与指针相关的错误

```
1  # include "stdio.h"
2
3  int main(void){
4      int val;
5      scanf("%d", val);
6
7      return 0;
8  }
```

  - 间接引用坏指针。scanf将val的内容解释为地址，并试图写入这个地址

```
3  int *stackref(){
4      int val;
5
6      return &val;
7  }
```

  - 函数返回一个指针，指向栈里的局部变量，然后弹出栈帧。p不再指向一个合法变量

常见段错误

```
4  //删除有*size项的二叉堆的第一项，然后对剩下的*size-1重新建堆
5  int *binheapDelete(int **binheap, int *size){
6
7      int *packet = binheap[0];
8      binheap[0] = binheap[*size - 1];
9
10     *size--;
11
12     //建堆函数
13     heapify(binheap, *size, 0);
14     return packet;
15 }
```

- 一元运算符--和*的优先级相同，从右向左结合。因此第10行减少的是指针自己的值

常见段错误

- 堆有关的错误

```
3  int *heapref(int n, int m){
4      int i;
5      int *x, *y;
6
7      x = (int *)malloc(n * sizeof(int));
8
9      /*...*/
10
11     free(x);
12
13     y = (int *)malloc(m * sizeof(int));
14     for(i = 0; i < m; i++){
15         y[i] = x[i]++;
16     }
17
18     return y;
19 }
```

```
3  int main(void){
4      char *str = (char *)malloc(10);
5
6      /*...*/
7
8      free(str);
9      str = NULL;
10
11     strcpy(str,"abcdef");
12
13     return 0;
14 }
```

  - 引用空闲堆块中的数据
  - 释放置空后，不能再使用该指针

常见段错误

- 栈有关的错误

```
3  int main(void){
4      main();
5
6      return 0;
7  }
```

  - 栈溢出
  - 常见的还有在线编程，使用递归无法通过

常见段错误

- 数组有关的错误

```
4  int **makeArray(int n, int m){
5      int i;
6      int **A = (int**)malloc(n * sizeof(int *));
7
8      for(i = 0; i <= n; i++){
9          A[i] = (int *)malloc(m * sizeof(int));
10     }
11     return A;
12 }
```

  - 第6行创建了一个n个元素的指针数组，第8、9行试图初始化这个数组的n+1个元素
  - 常见的给字符串分配空间，缺少’\0’的位置

段错误调试

段错误的调试方法

- 使用printf输出信息
  - 常用
- 使用objdump
  - 需要有一定的汇编语言基础
  - 编译过程中会优化代码，汇编指令和C语言指令难以对应
- 使用gdb调试
  - 源码可获取
  - 源码不可获取

段错误的调试方法

- 使用gcc和gdb（需要源码可以获取）
  - 为了能够使用gdb调试程序，在编译阶段加上-g参数
  - 使用gdb命令开始调试程序

```
1  # include "stdio.h"
2  # include "string.h"
3
4  int main(void){
5      char *ptr = "test";
6      strcpy(ptr, "TEST");
7
8      return 0;
9  }
```

```
master@master:~/DARA$ gcc -g segfalt.c -o seg
master@master:~/DARA$ gdb seg
GNU gdb (Ubuntu 7.7-0ubuntu3) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from seg...done.
(gdb)
```

段错误的调试方法

- 进入gdb后，r参数运行程序

```
(gdb) r
Starting program: /home/master/DARA/seg

Program received signal SIGSEGV, Segmentation fault.
0x00000000004004fd in main () at segfalt.c:6
6               strcpy(ptr, "TEST");
(gdb)
```

- 程序收到SIGSEGV信号，触发段错误，并提示地址

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004004fd in main () at segfalt.c:6
6               strcpy(ptr, "TEST");
(gdb) bt
#0  0x00000000004004fd in main () at segfalt.c:6
(gdb)
```

段错误的调试方法

- 使用core文件和gdb
  - 在一些Linux版本下，默认不产生core文件，因此首先设置core文件大小限制

```
master@master:~/DARA$ ulimit -c
0
master@master:~/DARA$ ulimit -c 1024
master@master:~/DARA$ ulimit -c
1024
```

  - 运行程序，生成core文件

```
master@master:~/DARA$ ./seg
Segmentation fault (core dumped)
```

– 加载core文件，使用gdb工具调试

```
master@master:~/DARA$ gdb seg core
GNU gdb (Ubuntu 7.7-0ubuntu3) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from seg...done.
[New LWP 66568]
Core was generated by `./seg'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00000000004004fd in main () at segfalt.c:6
6               strcpy(ptr, "TEST");
(gdb)
```

道德经

大成若缺，其用不弊。
大盈若冲，其用不穷。
大直若屈。大巧若拙。
大辩若讷。静胜躁，寒
胜热。清静为天下正。

谢谢！