段错误的原因及调试方法

1. 定义

段错误是指访问了不可访问的内存，例如访问了不存在的内存地址、访问了系统保护的内存地址、访问了只读的内存地址等等情况。

段错误一旦发生，cpu就会产生相应的保护，于是segmentation fault就出现

2. 引发段错误的原因

在编程中以下几类做法容易导致段错误,基本上是错误地使用指针引起的。

2.1 访问不存在的内存地址

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
   int *ptr = NULL;    //ptr指向的地址为NULL
   *ptr = 0;           //修改ptr指向的变量
}
```

2.2 访问系统保护的内存地址

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int *ptr = 0x00;  //将ptr指向0地址
    ptr = 100;
}
```

2.3 访问只读的内存地址

```
void main() {
  char *a="I am a teacher.";
  char *b="You are a student.";
  *b=*a;
  printf("复制后的字符串：\n%s\n%s\n",a,b);
}
```

2.4 越界使用

```
#include <stdio.h>
int main(){
    char test[1];
    printf("%c", test[10]);
    return 0;
}
```

2.5 栈溢出

```
#include <string.h>
#include <stdio.h>
 int main()
 {
     int a [10 * 1024 * 1024];
```

```
    a[0] = 1;
    return 0;
}
```

3. 段错误的调试方法

3.1 gdb调试

示例程序：segfault1.c

```
#include <stdio.h>
int main(){
   char *p;
   p = NULL;
   *p = 'x';
   printf("%c", *p);
   return 0;
}
```

3.1.1 调试步骤

1. 为了能够使用gdb调试程序，在编译阶段加上-g参数：

   ```
   $ gcc -g -o segfault1 segfault1.c
   $ ./segfault1
   Segmentation fault (core dumped)
   ```

2. 使用gdb命令调试程序：

   ```
   $ gdb ./segfault1
   ```

3. 进入gdb后，运行程序：

   ```
   (gdb) r
   Program received signal SIGSEGV, Segmentation fault.
   0x000055555555465e in main () at segfault1.c:5
   5          *p = 'x';
   ```

4. 完成调试后，输入quit命令退出gdb：

3.1.2 适用场景

一般用于测试阶段，gdb会有副作用：使程序运行减慢，运行不够稳定，等等。

3.1.3 使用core文件

段错误会触发SIGSEGV信号，SIGSEGV默认的handler会打印段错误出错信息，并产生core文件，由此我们可以借助于程序异常退出时生成的core文件中的调试信息，使用gdb工具来调试程序中的段错误。

利用core文件，出现段错误后，可直接进入gdb查看错误位置，不需要在gdb中再运行一次。

1. 在一些Linux版本下，默认是不产生core文件的，首先可以查看一下系统core文件的大小限制：

   ```
   $ ulimit -c
   0
   ```

2. 可以看到默认设置情况下，本机Linux环境下发生段错误时不会自动生成core文件，下面设置下core文件的大小限制（单位为KB）：

   ```
   $ ulimit -c 1024
   $ ulimit -c         //再次查看
   1024
   ```

3. 运行程序，发生段错误生成core文件：

```
$./segfault1
Segmentation fault (core dumped)
```

4. 加载core文件，使用gdb工具进行调试：

```
$ gdb ./segfault1 ./core
... ...
Core was generated by `./segfault1'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000560399c6165e in main () at segfault1.c:5
5          *p = 'x';
```

5. 完成调试后，输入quit命令退出gdb：

3.2 使用objdump进行反汇编查看

示例程序：segfault2.c

```
#include<stdio.h>
char *copyString(char *p1,char *p2) {
  *p2 = *p1;
  return p2;
}
void main() {
  char *a="I am a teacher.";
  char *b="You are a student.";
  b = copyString(a,b);
  printf("复制后的字符串：\n%s\n%s\n",a,b);
}
```

3.2.1 错误地址定位

1. **dmesg定位错误地址**

   使用dmesg命令，找到最近发生的段错误输出信息：

```
$ dmesg
... ...
[38171.710288] segfault2[12402]: segfault at 5622e3767758 ip 00005622e3767661 sp 00007ffc4fa0f540 error 7 in segfault2[5622e376700
```

   其中，对我们接下来的调试过程有用的是发生段错误的地址：指令指针地址：000055e262872661。

2. **catchsegv定位错误地址**

   使用dmesg命令，找到最近发生的段错误输出信息：

```
$ catchsegv ./segfault2
... ...
Backtrace:
./segfault2(+0x661)[0x563014153661]
./segfault2(+0x69a)[0x56301415369a]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7)[0x7f084471cbf7]
./segfault2(+0x56a)[0x56301415356a]
```

3.2.2 反汇编查看

1. 使用objdump生成二进制的相关信息，重定向到文件中：

```
$ objdump -S ./segfault2 > segfault2Dump
```

   其中，生成的segfault1Dump文件中包含了二进制文件的segfault1的汇编代码。

2. 在segfault1Dump文件中查找发生段错误的地址：

```
$ vi ./segfault2Dump
000000000000064a <copyString>:
#include<stdio.h>
char *copyString(char *p1,char *p2) {
 64a:   55                      push   %rbp
 64b:   48 89 e5                mov    %rsp,%rbp
 64e:   48 89 7d f8             mov    %rdi,-0x8(%rbp)
 652:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
  *p2 = *p1;
 656:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 65a:   0f b6 10                movzbl (%rax),%edx
 65d:   48 8b 45 f0             mov    -0x10(%rbp),%rax
 661:   88 10                   mov    %dl,(%rax)
  return p2;
 663:   48 8b 45 f0             mov    -0x10(%rbp),%rax
}
 667:   5d                      pop    %rbp
 668:   c3                      retq
```

3.2.3 适用场景

不需要-g参数编译（建议还是加上），不需要借助于core文件，但需要有一定的汇编语言基础。

4. 一些注意事项

1. 出现段错误时，首先应该想到段错误的定义，从它出发考虑引发错误的原因。
2. 在使用指针时，定义了指针后记得初始化指针，在使用的时候记得判断是否为NULL。
3. 在使用数组时，注意数组是否被初始化，数组下标是否越界，数组元素是否存在等。