访问 NULL 指针错误背后的原理

文章目录

前言

说到 NULL 指针大家都是谈之色变，第一印象就是 `NullPointerException`，`Segmentation fault` 之类的错误。NULL 指针大部分情况下会导致程序被终止。但是其实严格来说，访问空指针会产生不可预料的结果。只不过大部分情况是程序被终止。为什么呢？接下来让我们来探讨访问 NULL 指针错误背后的原理。

NULL 在编译器中的实现

首先，我们来看看 NULL 指针到底是什么？

> Null 是一个特殊指针值（或是一种对象引用）表示这个指针并不指向任何的对象。

举一些例子，C/C++ 中的 `NULL`，Python 中的 `None` 等等。大部分 `NULL` 实现是用 `0` 代表 `NULL`，例如说 C/C++ 。实际上，NULL 的值并不重要，重要的是它代表的含义。例如说，JVM 规范并没有规定 NULL 的值，不同虚拟机实现可以自己定义 NULL 的值。

总之， NULL 的值取决编译器实现。

访问 NULL 指针的过程

C 语言中，NULL 的值是 0，即 `NULL == 0` 是成立的。我们前面说访问 NULL 指针的行为会产生不可预料的后果。但是在 Linux 系统中后果是确定的：访问空指针会产生 `Segmentation fault` 的错误。因此这里的“不可预料”指的是在不同系统产生的后果不一样。

让我们假设现在使用的是 C 语言，运行在 Linux 系统上，以此来分析访问 NULL 指针的过程。

1. Linux 中，每个进程空间的 0x0 虚拟地址开始的线性区(memory region)都会被映射到一个用户态没有访问权限的页上。通过这样的映射，内核可以保证没有别的页会映射到这个区域。
2. 编译器把空指针当做 `0` 对待，开心地让你去访问空指针。
3. 缺页异常处理程序被调用，因为在 0x0 的页没有在物理内存里面。
4. 缺页异常处理程序发现你没有访问的权限。
5. 内核发送 `SIGSEGV` 信号给进程，该信号默认是让进程自杀。

可以看到：不需要特定的编译器实现或者内核的支持，只需要让一个页映射到 0x0 的虚拟地址上，就完美的实现了检测空指针的错误。

总结

为了研究这个问题，我查了很多资料。空指针的问题涉及 Linux 内存管理的知识，主要参考了 Robert Love 大神对该 问题 的回答和《深入理解Linux内核》。最大的感悟是带着问题去看内核的书，你会理解内核为什么要这么做，同时可以加深理解和记忆。

总之，空指针的实现取决于编译器的实现，访问空指针的后果取决于操作系统的实现。大部分系统类似于 Linux，会产生 `Segmentation fault` 的错误，至于内部实现就要看各个系统的代码了。

参考资料

What actually happens when dereferencing a NULL pointer?

《深入理解Linux内核》