

STL sort

Murashko Ivan

Introduction

The most tricky thing in C++ is an error catch. Most part of errors are located in the application level and you can be quite sure that the library, especially the well known STL, is error free. But what should you do if your application code is trivial i.e. seems to be error free but you got a SIGSEGV inside a system library. Most probably you just fell into a situation that is described in the article.

The examples in the article uses gcc 5.5.0 from the docker image gcc:5:

```
$ docker run gcc:5 gcc --version
gcc (GCC) 5.5.0
```

1 Example

Lets look at the following code:

```
#include <iostream>
#include <vector>
#include <algorithm>

void print(std::vector<int>& data) {
    for (int item : data) {
        std::cout << item << " ";
    }
    std::cout << std::endl;
}

int main(){
    std::vector<int> data = {1, 1, 1, 1, 1, 1, 1, 1, 1,
                            1, 1, 1, 1, 1, 1, 1, 1, 1};
    auto comp = [](int i1, int i2) { return i1 <= i2; };
    std::sort(data.begin(), data.end(), comp);
    print(data);
    return 0;
}
```

What's output does it produces? One can expect the following one:

```
$ ./src/sort
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$
```

i.e. the original array has to be displayed. In reality the following output is produced by gcc 5.5.0

```
$ ./src/sort
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$
```

As one can see the array is broken. How it can be if every function in the code snapshot seems to be correct? The answer is below.

2 STL requirements

The STL library states the following requirements for `std::sort` [1](pp. 897-898): the compare function must follow so called strict weak ordering requirement. Especially it says that

```
comp(x, x) == true;
```

The requirement is violated by the \leq operand used in our example.

Why the requirement is so important? The answer is in the algorithm that is used for sorting. The algorithm is called Quick sort [2].

Using the following command

```
docker run gcc:5 tail -n +1892 \
/usr/include/c++/4.9.2/bits/stl_algo.h | head -n 21
```

one can get the following code

```
/// This is a helper function...
template<typename _RandomAccessIterator, typename _Compare>
_RandomAccessIterator
__unguarded_partition(_RandomAccessIterator __first,
                     _RandomAccessIterator __last,
                     _RandomAccessIterator __pivot, _Compare __comp)
{
    while (true)
    {
        while (__comp(__first, __pivot))
            ++__first;
        --__last;
        while (__comp(__pivot, __last))
            --__last;
        if (!(__first < __last))
            return __first;
        std::iter_swap(__first, __last);
        ++__first;
    }
}
```

Conclusion

TBD

References

- [1] ISO. ISO/IEC JTC1 SC22 WG21 N 3690: Programming languages — C++ / ISO. — 2013. — Sep. — P. 1359. — <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf>.
- [2] Wikipedia contributors. Quicksort — Wikipedia, the free encyclopedia. — <https://en.wikipedia.org/w/index.php?title=Quicksort&oldid=917319740>. — 2019. — [Online; accessed 23-September-2019].