Undefined Behaviour in C

Dennis Sobczak

Field of work Scientific Computing
Field Computer Science
Faculty for Mathematics, Computer Science and Natural Sciences
University of Hamburg

2013-11-28

UHH Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Presentation outline

1 What is 'Undefined Behaviour'?

2 How does the compiler benefit?

3 Dangers

4 What one should be aware of?

5 Summary

6 References

Introduction

- Also: One of the 'dark sides' of C
- Operations against the C Standard
- Undefined behaviour is a behaviour unexpected
- It is what is evoked by breaking the rules
- Many different possibilities and many different classes
- Can cause bugs in a program, crash your system or do other unexpected things

Norms and Standards

- C has spread very fast
- Constant modifications and expansions
- Many versions of C
- In conclusion: Not supported completely by every C compiler
- ANSI, ISO Standards
- Rules and restrictions to keep faultless and reliable code

Why is undefined behaviour possible?

- C is an extremely efficient low-level programming language
- Not as 'safe' as other programming languages
- Causing undefined behaviour enables certain optimizations

Compiler

- Register allocation
- Scheduling
- Peephole optimizations
- Loop transformations
- Eliminating unnecessary abstractions
- . . .

GCC and LLVM

- Both compilers have different optimization strategies
- These can be disabled or enabled by setting flags

LLVM

- "Low-Level-Virtual-Machine"
- By LLVM Developer Group
- Initial release: 2003
- Written in C++
- Frontend: Clang

What can be optimized?

- Code
- Compilation time
- Performance of the system and applications
- Storage use

Division by zero

```
int main(){
 int i = 1;
 int j = 0;
 int result = i / j;
 return result;
}
```

Access an array beyond its bounds

```
int arr[42] = {0};
int *ptr = arr;
ptr += 41;
*ptr;
ptr += 1;
*ptr;
ptr += 1;
```

- Solution: check range

Casting types

```
int *num;
char *charPtr;
charPtr = (char*) num;
*charPtr = (char)0;
*charPtr = (char)2;
```

- Advantage:
  - Type-Based Alias Analysis

Using uninitialized variables

- Advantage:
  - No zero-initializations
- Disadvantage:
  - Overhead for stack arrays

Signed integer overflow

```
int number = INT_MAX;
result = INT_MAX + 1;
return result;
```

- Advantage:
    - No wraparound

Oversized shift amounts

```
unit32_t shift = 1;
shift = shift >> 32;
printf(" %" PRIu32 "\n", shift);
```

- Shifting values by an amount greater or equal to the number of bits in the number
- Depending on the platform you use:
  - format your hard drive
  - shift by zero
- Solution:
  - set variables to zero (lsl)
  - can be checked, if types bitwidth is known

Dereferencing a NULL Pointer

```
int* ptr = NULL;
int& ref = *ptr;
int* ptr2 = &ref;
```

- Benefits: Scalar optimizations exposed by macro expansion and inlining
- Danger: Application can crash

Dereferencing a Dangling Pointer

```
int* foo()
{
   int y;
   return &y;
}

int main()
{
   int* pY = foo();
}
```

- Dangers: Can overwrite a memory region

Interacting compiler optimizations

- Compiler can optimize your code without permission, i.e. remove dead code or null checkings
- "Dead Code Elimination" and "Redundant Null Check Elimination" → caused bug in Linuxkernel

Linuxkernel: "Checking for the NULL pointer"

```
void contains_null_check(int *P)
{
    int dead = *P;
    if(P == 0)
    return;
    *P = 4;
}
```

Security

- It is not secure to use undefined behaviour in security-critical code
- Undefined behaviour can make a system vulnerable to
  - exploitations by others
  - or end with system crashes

Example

```
void process_something(int size)
{
   if (size > size + 1 )
   abort();
   ...

   char *string = malloc(size + 1);
   read(fd, string, size);
   string[size] = 0;
   do_something(string);
   free(string);
}
```

Changing compiler without adapting the code

- If code is used on a different compiler, the code should be adapted to the compiler
- Otherwise the compiled code might cause undefined behaviour

What one should be aware of?

- The optimization strategies of the compiler can cross plans
- The compiler is allowed to eliminate code, i.e. if redundancy is detected
- No specially-tailored warning messages possible

Clang options to avoid undefined behaviour

- -fcatch-undefined-behavior -ftrapv
    - Detects undefined behavior in code
    - But: limited
- -fwrapv
    - Wraping signed integer overflow

Code Analyzers

- Clang Static Analyzer
  - Static analyzer
  - No information at runtime
- Valgrind
  - Dynamic analyzer
  - Information at runtime

Recommendations

- Inform yourself about the type of compiler
- Turn on compiler warnings
- Document preconditions and postconditions → assertions
- Debug and test

Summary

- Undefined behaviour is unexpected behaviour
  - caused by violating rules of the C Standard
- Classes of undefined behaviour
- Dangers
  - Compiler optimizations
  - Affecting security
- How to prevent it?
  - tools
  - workarounds

References

http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html

http://blog.regehr.org/archives/213

http://en.wikipedia.org/wiki/Undefined_behavior

http://stackoverflow.com/questions/2727834/c-standard-dereferencing-null-pointer-to-get-a-reference