Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 15

30 октября 2019 г.

Поиск ошибок работы с памятью

Частые ошибки работы с динамической памятью тяжело отлаживать (даже в небольших программах).

- Ошибки доступа за границы буфера
- Ошибки использования неинициализированного или уже освобожденного указателя
- Недостаточный размер буфера

Разработан ряд инструментов анализа, которые облегчают жизнь программисту.

- valgrind: динамический двоичный транслятор `http://valgrind.org`
- sanitizers: компиляторная инструментация от Google `https://github.com/google/sanitizers/wiki`

Disclaimer: Linux-only tools[1].

[1] На Windows работает Dr.Memory: `https://drmemory.org/`

Поиск ошибок работы с памятью

valgrind: динамический двоичный транслятор (плюс набор инструментов, ваш — memcheck).

```
#include <stdlib.h>
void f(void) {
  int* x = malloc(10 * sizeof(int));
  x[10] = 0;        // problem 1: heap block overrun
}
int main(void) {
  f();
  return 0;
}
$ gcc -Og -g -o me && valgrind ./me
 <...>
==27164== Invalid write of size 4
==27164==    at 0x400554: f (me.c:4)
==27164==    by 0x400568: main (me.c:7)
==27164==  Address 0x51da068 is 0 bytes after a block of size 40 alloc'd
==27164==    at 0x4C2C12F: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd
==27164==    by 0x400553: f (me.c:3)
==27164==    by 0x400568: main (me.c:7)
```

Поиск ошибок работы с памятью

sanitizers: встроенная в gcc/clang инструментация (нас интересует address sanitizer).

```
#include <stdlib.h>
void f(void) {
  int* x = malloc(10 * sizeof(int));
  x[10] = 0;          // problem 1: heap block overrun
}
int main(void) {
  f();
  return 0;
}
```

```
$ gcc -Og -g -fsanitize-address -o mesa && ./mesa
==27179==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60400000dff8
WRITE of size 4 at 0x60400000dff8 thread T0
    #0 0x4007c0 in f /home/bonzo/tmp/me.c:4
    #1 0x4007d5 in main /home/bonzo/tmp/me.c:7
    #2 0x7fba219d870f in __libc_start_main (/lib64/libc.so.6+0x2070f)
    #3 0x4006b8 in _start (/home/bonzo/tmp/mesa+0x4006b8)
0x60400000dff8 is located 0 bytes to the right of 40-byte region [0x60400000dfd0
allocated by thread T0 here:
    #0 0x7fba21df074a in malloc (/usr/lib64/libasan.so.2+0x9674a)
    #1 0x400793 in f /home/bonzo/tmp/me.c:3
SUMMARY: AddressSanitizer: heap-buffer-overflow /home/bonzo/tmp/me.c:4 f
```

Вычисления с плавающей точкой

- Предпосылки: дробные двоичные числа
- Стандарт арифметики с плавающей точкой IEEE 754: записи чисел, примеры
- Округление, сложение, умножение
- Плавающие типы языка Си
- Флаги компилятора gcc

Дробные двоичные числа

Что такое 1011.101_2?

Дробные двоичные числа

Что такое 1011.101_2?

$$1\times 2^3+0\times 2^2+1\times 2^1+1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}=11\frac{5}{8}=11.625.$$

Дробные двоичные числа

Что такое 1011.101_2?

$$1\times 2^3+0\times 2^2+1\times 2^1+1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}=11\frac{5}{8}=11.625.$$

$0.1111\ldots_2 = 1.0-\varepsilon(\varepsilon\to 0)$, т.к. $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots+\frac{1}{2^n}\to 1$ при $n\to\infty$.

Точно можно представить только числа вида $\frac{x}{2^k}$.

Осстальные рациональные числа представляются периодическими двоичными дробями: $\frac{1}{5}=0.(0011)_2$.

Иррациональные числа представляются апериодическими двоичными дробями и могут быть представлены только приближенно.

Представление чисел с плавающей точкой (IEEE 754)

Числа с плавающей точкой представляются в *нормализованной* форме:

$$(-1)^s M 2^e, \text{ где}$$

- s — код знака числа (он же знак мантиссы);
- M — мантисса ($1 \leqslant M < 2$);
- e — (двоичный) порядок.

Первая цифра мантиссы в нормализованном представлении всегда 1. В стандарте принято решение не записывать в представление числа эту единицу (тем самым мантисса как бы увеличивается на разряд).

В представление числа записывается не M, а $frac = M - 1$.

Представление чисел с плавающей точкой (IEEE 754)

Чтобы не записывать отрицательных чисел в поле порядка, вводится *смещение* $bias = 2^{k-1} - 1$, где k — количество бит в поле для записи порядка, и вместо порядка e записывается *код* порядка exp, связанный с e соотношением $e = exp - bias$.

Нормализованное число $(-1)^s M 2^e$ упаковывается в машинное слово с полями s, $frac$ и exp.

s	exp (код порядка)	$frac$ (код мантиссы)

Ширина поля s всегда равна 1. Ширина полей exp и $frac$ зависит от точности числа.

Типы плавающей арифметики (точность)

Одинарная точность (32 бита): exp — 8 бит, $frac$ — 23 бита.
$bias = 127$, $-126 \leqslant e \leqslant 127$, $1 \leqslant exp \leqslant 254$.

Двойная точность (64 бита): exp — 11 бит, $frac$ — 52 бита.
$bias = 1023$, $-1022 \leqslant e \leqslant 1023$, $1 \leqslant exp \leqslant 2046$.

Повышенная точность (80 бит): exp — 15 бит, $frac$ — 64 бита.

Пример

```
float f = 15213.0;
```

$15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$.

$M = 1.\underline{1101101101101}_2$,
$frac = \underline{1101101101101}0000000000_2$.

$e = 13, bias = 127, exp = 140 = 10001100_2$.

Результат:

0	10001100	11011011011010000000000
s	exp	$frac$

Представление нуля

Для типа `float` код порядка *exp* изменяется от 00000001 до 11111110 (значению 00000001 соответствует порядок $e = -126$, значению 11111110 — порядок $e = 127$).

Код $exp = 00000000$, $frac = 000\ldots0$ представляет нулевое значение; в зависимости от значения знакового разряда s это либо $+0$, либо -0.

А какое значение представляют коды
$exp = 00000000$, $frac \neq 000\ldots0$ и
$exp = 11111111$?

Большие числа

Пусть $exp = 111\ldots1$.

Если при этом $frac = 000\ldots0$, то коду будет соответствовать значение ∞ (со знаком s).

Если же $frac \neq 000\ldots0$, то код не будет представлять *никакое* число («значение», представляемое таким кодом, так и называется: «не число» — NaN — Not a number).

Денормализованные числа

Это числа, представляемые кодами $exp = 000\ldots0$, $frac \neq 000\ldots0$.

exp вносит в значение такого числа постоянный вклад 2^{-k-2}, $frac$ меняется от $000\ldots1$ до $111\ldots1$ и рассматривается уже не как мантисса, а как значение, умножаемое на exp.

Пример: 8-разрядные числа

s	exp	$frac$
1 бит	4 бита	3 бита

	s	exp	$frac$	E	значение	
Денормализованные числа	0	0000	000	-6	0	Близкие к 0
	0	0000	001	-6	$1/8 \times 1/64 = 1/512$	
	0	0000	010	-6	$2/8 \times 1/64 = 2/512$	
				...		
	0	0000	110	-6	$6/8 \times 1/64 = 6/512$	
	0	0000	111	-6	$7/8 \times 1/64 = 7/512$	Наибольшее денормализованное
Нормализованные числа	0	0001	000	-6	$8/8 \times 1/64 = 8/512$	Наименьшее нормализованное
	0	0001	001	-6	$9/8 \times 1/64 = 9/512$	
				...		
	0	0110	110	-1	$14/8 \times 1/2 = 14/16$	
	0	0110	111	-1	$15/8 \times 1/2 = 15/16$	Ближайшее к 1 снизу
	0	0111	000	0	$8/8 \times 1 = 1$	
	0	0111	001	0	$9/8 \times 1 = 9/8$	Ближайшее к 1 сверху
				...		
	0	1110	110	7	$14/8 \times 128 = 224$	
	0	1110	111	7	$15/8 \times 128 = 240$	Наибольшее нормализованное
	0	1111	000		$+\infty$	

Важные частные случаи

Что	*exp*	*frac*	Численное значение `float`	Численное значение `double`
Ноль	00...00	00...00	`0.0`	
Наименьшее положительное денормализованное	00...00	00...01	$2^{-23} \times 2^{-126}$	$2^{-52} \times 2^{-1022}$
Наибольшее положительное денормализованное	00...00	11...11	$(1-\varepsilon) \times 2^{-126}$	$(1-\varepsilon) \times 2^{-1022}$
Единица	01...11	00...00	`1.0`	
Наибольшее положительное нормализованное	11...10	11...11	$(2-\varepsilon) \times 2^{127}$	$(2-\varepsilon) \times 2^{1023}$

Операции над числами с плавающей точкой

$$x +_{FP} y = \text{Round}(x + y)$$

$$x \times_{FP} y = \text{Round}(x \times y),$$

где *Round* означает округление.

Выполнение операции:

- сначала вычисляется точный результат (получается более длинная мантисса, чем запоминаемая, иногда в два раза);
- потом фиксируется исключение (например, переполнение);
- потом результат округляется, чтобы поместиться в поле *frac*.

Умножение чисел с плавающей точкой

$$(-1)^{s^1} M_1 2^{e^1} \times (-1)^{s^2} M_2 2^{e^2}$$

Точный результат: $(-1)^s M 2^e$, где

- $s = s_1 \wedge s_2$,
- $M = M_1 \times M_2$,
- $e = e_1 + e_2$.

Преобразование:

- если $M \geqslant 2$, сдвиг M вправо с одновременным увеличением e;
- если e не помещается в поле *exp*, фиксируется переполнение;
- округление M, чтобы оно поместилось в поле *frac*.

Основные затраты на перемножение мантисс.

Сложение чисел с плавающей точкой

$$(-1)^{s^1} M_1 2^{e^1} + (-1)^{s^2} M_2 2^{e^2}, \text{ где } e_1 > e_2.$$

Точный результат: $(-1)^s M 2^e$.

- Порядок суммы — e_1.
- К мантиссе M_1 прибавляется $e_1 - e_2$ старших разрядов мантиссы M_2.

Преобразование:

- если $M \geqslant 2$, сдвиг M вправо с одновременным увеличением e;
- если $M < 1$, сдвиг M влево на k позиций с одновременным вычитанием k из e;
- если e не помещается в поле *exp*, фиксируется переполнение;
- округление M, чтобы оно поместилось в поле *frac*.

Пример. Сложение чисел «типа» `float`

Мантисса — 6 десятичных цифр, порядок — 2 десятичных цифры.
$0.231876 * 10^{02} + 0.645391 * 10^{-03} + 0.231834 * 10^{-01} + 0.245383 * 10^{-02} + 0.945722 * 10^{-03}$.

Сложение по порядку: $0.2321\mathbf{4}7 * 10^{02}$.

$23.1876 + 0.000645391 = 23.188245391 = 23.1882 = 0.231882 * 10^{02}$;
$23.1882 + 0.0231834 = 23.2113834 = 23.2114 = 0.232114 * 10^{02}$;
$23.2114 + 0.00245383 = 23.21385383 = 23.2138 * 10^{02}$;
$23.2138 + 0.000945722 = 23.214745722 = 23.2147 = 0.232147 * 10^{02}$.

Сложение по размеру: $0.2321\mathbf{5}7 * 10^{02}$.

$0.000645391 + 0.000945722 = 0.001591113 = 0.00159111 = 0.159111 * 10^{-02}$;
$0.00159111 + 0.00245383 = 0.00494493 = 0.494493 * 10^{-02}$;
$0.00494493 + 0.0231834 = 0.02812833 = 0.0281283 = 0.281283 * 10^{-01}$;
$0.0281283 + 23.1876 = 23.2157283 = 23.2157 = 0.232157 * 10^{02}$.

Выводы по операциям

При вычислении суммы чисел с одинаковыми знаками необходимо упорядочить слагаемые по возрастанию и складывать, начиная с наименьших слагаемых.

При вычислении суммы чисел с разными знаками необходимо сначала сложить все положительные числа, потом — все отрицательные числа и в конце выполнить одно вычитание.

Вычитание (сложение чисел с противоположными знаками) часто приводит к потере точности, которая у чисел с плавающей точкой определяется количеством значащих цифр в мантиссе (при вычитании двух близких чисел мантисса «исчезает», что ведет к резкой потере точности).
Итак, чем меньше вычитаний, тем точнее результат.

Вещественные типы языка Си

`float, double, long double`.

Традиционные арифметические операции.

Внимание: в функциях с переменным числом параметров `float` автоматически преобразуется в **`double`** (в части переменных параметров).

Режимы `gcc` для работы с плавающей точкой*

`https://gcc.gnu.org/wiki/FloatingPointMath`

Детальное резюме того, что бывает в `gcc`, и таблица преобразований, влияющих на результат вычислений.

- `-ffast-math`: считать максимально быстро, но, возможно, нарушать стандарт IEEE-754
  - Полезно для тестирования, но не распространения финальной версии программы
- `-fno-math-errno`: не устанавливать переменную `errno` как результат ошибочного выполнения математических функций
  - Можно обойтись и без этого, но зависит от библиотеки Си.
  - Компилятор может заменять вызовы функций инструкциями процессора (например, `sqrt`).

David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. 23, 1 (March 1991), 5-48.

`https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html`

Режимы gcc для работы с плавающей точкой*

https://gcc.gnu.org/wiki/FloatingPointMath

Детальное резюме того, что бывает в gcc, и таблица преобразований, влияющих на результат вычислений.

> `-fno-trapping-math`: считать, что вычисления с плавающей точкой не могут вызывать исключений процессора (traps)
>
> Т.е. вы гарантируете отсутствие в своем коде ситуаций, вызывающих деления на ноль, переполнения, некорректные операции.
>
> Компилятор может более свободно комбинировать, переставлять, удалять операции с плавающей точкой.

David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. 23, 1 (March 1991), 5-48.

https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html