

Раскручиваем стек

Пономарев Иван, Synesis





**Что такое стек
вызовов?**

Стек вызовов

LIFO структура данных, хранящая адреса возврата в подпрограммы

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)
- Выделяется:
 - Ядром ОС (для главного потока)
 - Внутри процесса при создании потока (pthread)

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)
- Выделяется:
 - Ядром ОС (для главного потока)
 - Внутри процесса при создании потока (pthread)
- Вершина стека - регистр **Stack Pointer** (SP, ESP, RSP)

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)
- Выделяется:
 - Ядром ОС (для главного потока)
 - Внутри процесса при создании потока (pthread)
- Вершина стека - регистр **Stack Pointer** (SP, ESP, RSP)
- Размер элемента - машинное слово

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)
- Выделяется:
 - Ядром ОС (для главного потока)
 - Внутри процесса при создании потока (pthread)
- Вершина стека - регистр **Stack Pointer** (SP, ESP, RSP)
- Размер элемента - машинное слово
- Наиболее популярен **Full Descending** стек
 - Stack Pointer указывает на элемент, находящийся на вершине стека
 - Стек растёт **вниз** (в сторону уменьшения адресов)

Как работает стек вызовов

- Стек - область памяти, уникальная для каждого потока (thread)
- Выделяется:
 - Ядром ОС (для главного потока)
 - Внутри процесса при создании потока (pthread)
- Вершина стека - регистр **Stack Pointer** (SP, ESP, RSP)
- Размер элемента - машинное слово
- Наиболее популярен **Full Descending** стек
 - Stack Pointer указывает на элемент, находящийся на вершине стека
 - Стек растёт **вниз** (в сторону уменьшения адресов)
- Операции над стеком:
 - Push = сдвиг SP + запись в память
 - Pop = сдвиг SP в обратную сторону
 - Выделение памяти = сдвиг SP

Простейший стек вызовов

```
void func3() {  
    // Мы здесь  
}  
  
void func2() {  
    func3();  
}  
  
void func1() {  
    func2();  
}  
  
void func0() {  
    func1();  
}
```

Stack Pointer →

Адрес возврата в func2()

Адрес возврата в func1()

Адрес возврата в func0()

...

Стековый кадр

- Адреса возврата
- Сохранённые регистры
- Локальные переменные
- Аргументы функции

Stack Pointer →

Локальные переменные func3()

Сохранённые регистры func3()

Адрес возврата в func2()

Аргументы для func3()

func3()

Локальные переменные func2()

Сохранённые регистры func2()

Адрес возврата в func1()

Аргументы для func3()

func2()

...

Соглашение вызова (calling convention)

- Как передаются аргументы?
 - Регистры?
 - Стек?
 - В каком порядке?
- Кто очищает стек от аргументов?
 - Вызывающая функция? (caller)
 - Вызываемая функция? (callee)
- Как возвращается результат?
 - Регистры?
 - Стек?
- Какие регистры необходимо сохранить в стек?

Раскрутка стека вызовов (call stack unwinding)

- Проход по содержимому стека
- Перебор стековых кадров в порядке, обратном вызову функций
- Логика над полученными данными

Раскрутка стека вызовов (call stack unwinding)

- Проход по содержимому стека
- Перебор стековых кадров в порядке, обратном вызову функций
- Логика над полученными данными

Применение:

- Обработка C++ исключений
- Получение backtrace (stack trace)
 - Отладчики
 - Системы краш репортов
 - Профилировщики

Сложности раскрутки стека

- Разный формат стековых кадров для разных функций:
 - Различное количество аргументов, передаваемых по-разному
 - Различное число локальных переменных
 - Разные регистры сохраняются в стек

Сложности раскрутки стека

- Разный формат стековых кадров для разных функций:
 - Различное количество аргументов, передаваемых по-разному
 - Различное число локальных переменных
 - Разные регистры сохраняются в стек
- Зависимость от архитектуры и ABI
 - Разные соглашения вызова для разных функций

Сложности раскрутки стека

- Разный формат стековых кадров для разных функций:
 - Различное количество аргументов, передаваемых по-разному
 - Различное число локальных переменных
 - Разные регистры сохраняются в стек
- Зависимость от архитектуры и ABI
 - Разные соглашения вызова для разных функций
- Оптимизации компилятора

Способы применения раскрутки стека

- Для текущего запущенного процесса
 - Получение backtrace (boost stacktrace)
 - Создание crash report (signal handler)

Способы применения раскрутки стека

- Для текущего запущенного процесса
 - Получение backtrace (boost stacktrace)
 - Создание crash report (signal handler)
- Для другого запущенного процесса (ptrace, process_vm_readv)
 - Отладчики (gdb, llDb)
 - Профилировщики (perf)

Способы применения раскрутки стека

- Для текущего запущенного процесса
 - Получение backtrace (boost stacktrace)
 - Создание crash report (signal handler)
- Для другого запущенного процесса (ptrace, process_vm_readv)
 - Отладчики (gdb, llDb)
 - Профилировщики (perf)
- Для завершенного процесса (postmortem)
 - Отладчик + core dump
 - Google breakpad / crashpad

Раскрутка стека в деталях

Разбор алгоритма

Регистры

- Stack pointer (SP, ESP, RSP)
 - Вершина стека.
 - Может меняться во время исполнения функции

Регистры

- Stack pointer (SP, ESP, RSP)
 - Вершина стека.
 - Может меняться во время исполнения функции
- Frame pointer, Base pointer (FP, EBP, RBP):
 - Используется для адресации локальных переменных
 - Указывает на начало стекового кадра текущей функции
 - Не меняется во время исполнения функции
 - Может не использоваться (-fomit-frame-pointer)

Регистры

- Stack pointer (SP, ESP, RSP)
 - Вершина стека.
 - Может меняться во время исполнения функции
- Frame pointer, Base pointer (FP, EBP, RBP):
 - Используется для адресации локальных переменных
 - Указывает на начало стекового кадра текущей функции
 - Не меняется во время исполнения функции
 - Может не использоваться (-fomit-frame-pointer)
- Instruction pointer, program counter (PC, IP, EIP, RIP)
 - Указывает на исполняемую инструкцию.

Регистры

- Stack pointer (SP, ESP, RSP)
 - Вершина стека.
 - Может меняться во время исполнения функции
- Frame pointer, Base pointer (FP, EBP, RBP):
 - Используется для адресации локальных переменных
 - Указывает на начало стекового кадра текущей функции
 - Не меняется во время исполнения функции
 - Может не использоваться (-fomit-frame-pointer)
- Instruction pointer, program counter (PC, IP, EIP, RIP)
 - Указывает на исполняемую инструкцию.
- Другие регистры в зависимости от архитектуры
 - Link Register (LR) - адрес возврата (ARM 32-bit)

Виртуальная раскрутка стека (Virtual Unwinding)

- Виртуальные регистры (VRs):
 - Копия (snapshot) значений регистров в памяти
 - Все манипуляции происходят с копией
 - Значения извлекаются из стека

Виртуальная раскрутка стека (Virtual Unwinding)

- Виртуальные регистры (VRs):
 - Копия (snapshot) значений регистров в памяти
 - Все манипуляции происходят с копией
 - Значения извлекаются из стека
- Доступ к памяти стека:
 - Стековые регистры (SP, FP) содержат адреса элементов стека
 - Чтение - разыменование указателей

Виртуальная раскрутка стека (Virtual Unwinding)

- Виртуальные регистры (VRs):
 - Копия (snapshot) значений регистров в памяти
 - Все манипуляции происходят с копией
 - Значения извлекаются из стека
- Доступ к памяти стека:
 - Стековые регистры (SP, FP) содержат адреса элементов стека
 - Чтение - разыменование указателей
- Стек “заморожен”, пока выполняется раскрутка
 - Не можем выйти из функции, пока продолжается раскрутка (текущий поток)
 - Выполнение потока приостановлено (другой поток / процесс)

Общий алгоритм раскрутки стека

1. Получаем исходные данные - значения регистров (ip, sp, fp)
2. Выполняем логику над значениями регистров
3. **Считываем предыдущий стековый кадр и восстанавливаем значения регистров для предыдущего стекового кадра**
4. Зацикливаем пункт 2

Алгоритм №0

ABI с использованием Frame Pointer

ABI c Frame Pointer

```
Save/Load + Add new... CppInsights C++ x86-64 gcc 8.3 -fno-omit-frame-pointer A 11010 LXO lib.f .text // s+ Intel Demangle - + -
```

```
1 #include <string>
2 #include <iostream>
3
4 void func1(const std::string &name, int day, int month, int year) {
5     std::cout << name << day << month << year << std::endl;
6 }
7
8 int main()
9 {
10     const std::string name("corehard");
11     func1(name, 25, 5, 2019);
12 }
```

Load or save text

```
func1(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> name, int day, int month, int year)
{
    push rbp
    mov rbp, rsp
    sub rsp, 32
    mov QWORD PTR [rbp-8], rdi
    mov DWORD PTR [rbp-12], esi
    mov DWORD PTR [rbp-16], edx
    mov DWORD PTR [rbp-20], ecx
    mov rax, QWORD PTR [rbp-8]
    mov rsi, rax
    mov edi, OFFSET FLAT:_ZSt4cout
    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)
    mov rdx, rax
    mov eax, DWORD PTR [rbp-12]
    mov esi, eax
    mov rdi, rdx
    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)
    mov rdx, rax
    mov eax, DWORD PTR [rbp-16]
    mov esi, eax
    mov rdi, rdx
    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)
    mov rdx, rax
    mov eax, DWORD PTR [rbp-20]
```

ABI с Frame Pointer

- Пролог функции сохраняет предыдущее значение frame pointer в стеке после адреса возврата
- Раскрутка стека - проход по односвязному списку
- Флаг `-fno-omit-frame-pointer`
- Используется на платформах Apple

Stack Pointer →
Frame Pointer →

Локальные переменные func3()

Frame Pointer func2()

Адрес возврата в func2()

Аргументы func3()

Локальные переменные func2()

Frame Pointer func1()

Адрес возврата в func1()

Аргументы func2()

Локальные переменные func0()

Frame Pointer func0()

Адрес возврата

ABI с Frame Pointer

Алгоритм раскрутки стека

1. Получаем исходные данные - значения регистров (ip, sp, fp)
2. Выполняем логику над значениями регистров
3. Считываем адрес возврата, Instruction Pointer = *(FP - 2)
4. Считываем предыдущее значение Frame Pointer = *(FP - 1)
5. Зацикливаем пункт 2

ABI с Frame Pointer

Преимущества:

- Простота реализации
- Скорость раскрутки стека

Недостатки:

- Дополнительные накладные расходы на вызов функций
- Ограничение возможности оптимизации кода

ABI без Frame Pointer

```
C++ source #1 CppInsights C++  
1 #include <string>  
2 #include <iostream>  
3  
4 void func1(const std::string &name, int day, int month, int year) {  
    std::cout << name << day << month << year << std::endl;  
}  
5  
6  
7  
8 int main()  
9 {  
10     const std::string name("corehard");  
11     func1(name, 25, 5, 2019);  
12 }
```

```
x86-64 gcc 8.3 -fomit-frame-pointer  
A 11010 Lx0: libt: .text // s+ intel Demangle - + *  
1 func1(std::basic_string<char, std::char_traits<char>, std::allocator<char> > &name, int day, int month, int year)  
2     sub rsp, 40  
3     mov QWORD PTR [rsp+24], rdi  
4     mov DWORD PTR [rsp+20], esi  
5     mov DWORD PTR [rsp+16], edx  
6     mov DWORD PTR [rsp+12], ecx  
7     mov rax, QWORD PTR [rsp+24]  
8     mov rsi, rax  
9     mov edi, OFFSET FLAT:_ZSt4cout  
10    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)  
11    mov rdx, rax  
12    mov eax, DWORD PTR [rsp+20]  
13    mov esi, eax  
14    mov rdi, rdx  
15    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)  
16    mov rdx, rax  
17    mov eax, DWORD PTR [rsp+16]  
18    mov esi, eax  
19    mov rdi, rdx  
20    call std::basic_ostream<char, std::char_traits<char> >::operator<< (int)  
21    mov rdx, rax  
22    mov eax, DWORD PTR [rsp+12]  
23    mov esi, eax  
24    mov rdi, rdx
```

Алгоритм №1

Таблицы раскрутки стека

Таблицы раскрутки стека

- Секции с дополнительной информацией
- Содержат инструкции раскрутки стека

Форматы:

- DWARF
- ARM EXIDX (32-bit)
- Windows x64 .pdata
 - RUNTIME_FUNCTION
 - UNWIND_INFO

Таблицы раскрутки стека

Недостатки:

- Необходимость в дополнительных данных
- Скорость раскрутки стека ниже

Преимущества:

- Свобода выбора ABI и соглашения вызова
- Нет ограничений по оптимизации кода

DWARF

- Стандарт <http://www.dwarfstd.org/doc/DWARF5.pdf>
- Раздел 6.4 “Call Frame Information”
- ELF секция .eh_frame или .debug_frame
- Таблица бинарного поиска - ELF секция .eh_frame_hdr

Canonical Frame Address (CFA)

An area of memory that is allocated on a stack called a “call frame.” The call frame is identified by an address on the stack. We refer to this address as the Canonical Frame Address or CFA. Typically, the CFA is defined to be the value of the stack pointer at the call site in the previous frame (which may be different from its value on entry to the current frame).

<http://www.dwarfstd.org/doc/DWARF5.pdf> § 6.4

Canonical Frame Address (CFA)

- Значение Stack Pointer на момент исполнения инструкции **call**
- Canonical Frame Address \approx Frame Pointer

DWARF, § 6.4.1

Abstractly, this mechanism describes a very large table that has the following structure:

LOC	CFA	R0	R1	...	RN
L0					
L1					
LN					

The first column indicates an address for every location that contains code in a program. (In shared object files, this is an object-relative offset.) The remaining columns contain virtual unwinding rules that are associated with the indicated location.



DWARF

- Компактный бинарный формат

DWARF

- Компактный бинарный формат
- 2 типа записей

DWARF

- Компактный бинарный формат
- 2 типа записей
 - Frame Description Entry (FDE)
 - Одна запись на функцию
 - Содержит инструкции добавления строк в таблицу и заполнения ячеек
 - Может содержать указатель на LSDA (запись в секции .gcc_except_table) для обработки C++ исключений

DWARF

- Компактный бинарный формат
- 2 типа записей
 - Frame Description Entry (FDE)
 - Одна запись на функцию
 - Содержит инструкции добавления строк в таблицу и заполнения ячеек
 - Может содержать указатель на LSDA (запись в секции .gcc_except_table) для обработки C++ исключений
 - Common Information Entry (CIE)
 - Общая для нескольких FDE
 - Содержит общие параметры для всех связанных FDE
 - Может содержать указатель на personality routine для обработки C++ исключений

DWARF CIE

```
$ readelf --debug-dump=frames elf_file
...
00000030 0000000000000014 00000000 CIE
  Version: 1
  Augmentation: "zR"
  Code alignment factor: 1
  Data alignment factor: -8
  Return address column: 16
  Augmentation data: 1b

  DW_CFA_def_cfa: r7 (rsp) ofs 8
  DW_CFA_offset: r16 (rip) at cfa-8
  DW_CFA_nop
  DW_CFA_nop
  ...
  
```

DWARF FDE

```
readelf --debug-dump=frames elf_file
...
000000b0 0000000000000024 00000084 FDE cie=00000030
pc=00000000000015d0..000000000000187e
    DW_CFA_advance_loc: 1 to 00000000000015d1
    DW_CFA_def_cfa_offset: 16
    DW_CFA_offset: r6 (rbp) at cfa-16
    DW_CFA_advance_loc: 3 to 00000000000015d4
    DW_CFA_def_cfa_register: r6 (rbp)
    DW_CFA_advance_loc: 10 to 00000000000015de
    DW_CFA_offset: r12 (r12) at cfa-24
    DW_CFA_offset: r3 (rbx) at cfa-32
    DW_CFA_advance_loc2: 671 to 000000000000187d
    DW_CFA_def_cfa: r7 (rsp) ofs 8
    DW_CFA_nop
...
```

DWARF FDE Interpreted

```
readelf --debug-dump=frames-interp elf_file
...
000000b0 0000000000000024 00000084 FDE cie=00000030
pc=00000000000015d0..000000000000187e

LOC          CFA      rbx    rbp    r12    ra
00000000000015d0  rsp+8    u      u      u      c-8
00000000000015d1  rsp+16   u      c-16   u      c-8
00000000000015d4  rbp+16   u      c-16   u      c-8
00000000000015de  rbp+16   c-32   c-16   c-24   c-8
000000000000187d  rsp+8    c-32   c-16   c-24   c-8
...
```

DWARF CIE Interpreted

```
$ readelf --debug-dump=frames-interp elf_file  
...  
00000030 0000000000000014 00000000 CIE "zR" cf=1 df=-8 ra=16  
LOC          CFA      ra  
0000000000000000 rsp+8    c-8  
...
```

DWARF

```
$ objdump -d -S elf_file
```

```
00000000000015d0 <_Z5func2fRKNSt7__cxx11basic_stringIcSt11char_traitsIcESaIcEEE>:
```

15d0:	55	push %rbp
15d1:	48 89 e5	mov %rsp,%rbp
15d4:	41 54	push %r12
15d6:	53	push %rbx
15d7:	48 81 ec 70 10 00 00	sub \$0x1070,%rsp
15de:	f3 0f 11 85 8c ef ff	movss %xmm0,-0x1074(%rbp)
15e5:	ff	
15e6:	48 89 bd 80 ef ff ff	mov %rdi,-0x1080(%rbp)
15ed:	48 8d 85 d0 f7 ff ff	lea -0x830(%rbp),%rax
15f4:	be 00 01 00 00	mov \$0x100,%esi
15f9:	48 89 c7	mov %rax,%rdi
15fc:	e8 bf fc ff ff	callq 12c0 <backtrace@plt>
1601:	89 45 e4	mov %eax,-0x1c(%rbp)
1604:	c7 45 ec 00 00 00 00	movl \$0x0,-0x14(%rbp)

DWARF

Алгоритм раскрутки стека

1. Получаем исходные данные - значения регистров (ip, sp, fp)
2. Выполняем логику над значениями регистров
3. Ищем запись (FDE) по значению регистра Instruction pointer (Program counter) в .eh_frame (.debug_frame)
4. Восстанавливаем значения регистров для предыдущего стекового кадра по инструкциям из FDE
5. Зацикливаем пункт 2



Флаги компилятора

Флаги компилятора (оптимизация)

Влияют на создание стекового кадра при вызове:

- -foptimize-sibling-calls / -fno-optimize-sibling-calls
- -finline / -fno-inline

Влияют на возможность раскрутки стека:

- -fexceptions, -funwind-tables / -fno-exceptions, -fno-unwind-tables
- -fomit-frame-pointer / -fno-omit-frame-pointer



Готовые библиотеки

libc

https://www.gnu.org/software/libc/manual/html_node/Backtraces.html

- Функция **backtrace (void** array, int size)**
- Содержится в большинстве реализаций libc
- Возвращает адреса возврата

libc backtrace

```
#include <execinfo.h>

void *backtrace_buffer[256];
const int backtrace_size = backtrace(backtrace_buffer, 256);
for (int i = 0; i < backtrace_size; ++i) {
    void * const program_counter = backtrace_buffer[i];
    cout << program_counter;
    Dl_info dl_info;
    if (dladdr(program_counter, &dl_info) && dl_info.dli_sname) {
        cout << " " << dl_info.dli_sname << " + " << (intptr_t)program_counter -
(intptr_t)dl_info.dli_saddr;
    }
    cout << endl;
}
```

C++ ABI

<https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

http://refspecs.linuxbase.org/LSB_4.1.0/LSB-Core-generic/LSB-Core-generic/baselib-unwind-backtrace.html

- Библиотека компилятора, используемая для обработки C++ исключений
- Функция `_Unwind_Backtrace`

C++ ABI

```
#include <unwind.h>

_Unwind_Reason_Code unwind_function(_Unwind_Context *context, void *arg) {
    _Unwind_Ptr program_counter = _Unwind_GetIP(context);
    cout << reinterpret_cast<void *>(program_counter);
    Dl_info dl_info;
    if (dladdr(reinterpret_cast<void *>(program_counter), &dl_info) &&
        dl_info.dli_sname) {
        cout << " " << dl_info.dli_sname << " + " << program_counter -
            reinterpret_cast<_Unwind_Ptr>(dl_info.dli_saddr);
    }
    cout << endl;
    return _URC_NO_REASON;
}
...
_Unwind_Backtrace(&unwind_function, NULL);
```

libunwind

<https://www.nongnu.org/libunwind>

<https://github.com/libunwind/libunwind>

- Платформы: Linux, FreeBSD, HP-UX
- Язык: C
- Множество архитектур

libunwind

```
#include <libunwind.h>

unw_context_t unwind_context;
unw_getcontext(&unwind_context); // Сохраняем состояние процессора (виртуальные регистры)
unw_cursor_t unwind_cursor; // Инициализируем курсор
if (!unw_init_local(&unwind_cursor, &unwind_context)) {
    for (;;) {
        // Получаем значение счётчика команд (program counter, instruction pointer)
        unw_word_t program_counter;
        if (unw_get_reg(&unwind_cursor, UNW_REG_IP, &program_counter)) break;
        cout << reinterpret_cast<void *>(program_counter);
        // Получаем имя функции и смещение инструкции внутри функции
        unw_word_t function_offset;
        char function_name[128];
        if (!unw_get_proc_name(&unwind_cursor, function_name, sizeof(function_name), &function_offset)) {
            cout << " " << function_name << " + " << function_offset;
        }
        cout << endl;
        if (unw_step(&unwind_cursor) <= 0) break;
    }
}
```

Windows

- CaptureStackBackTrace <https://msdn.microsoft.com/ru-RU/library/windows/desktop/bb204633>
- StackWalk <https://docs.microsoft.com/en-us/windows/desktop/api/dbghelp/nf-dbghelp-stackwalk>
- RtlVirtualUnwind (64-bit) <https://docs.microsoft.com/en-us/windows/desktop/api/winnt/nf-winnt-rtlvirtualunwind>

Спасибо за внимание

Ссылки:

- <http://www.dwarfstd.org/doc/DWARF5.pdf> раздел 6.4
- <https://itanium-cxx-abi.github.io/cxx-abi/abi.html>

Контакты:

- E-mail: ivantrue@gmail.com
- Skype: ivan_arh
- Telegram: @ivanarh
- Github: <https://github.com/ivanarh>