RTFM: Linux, DevOps и системное администрирование

DevOps-инжиниринг и системное администрирование. Случаи из практики.

C: отладка с gdb – примеры

Автор: setevoy | 01/30/2019 0 Comments

Rate this (3 Votes)

gdb (*GNU Project Debugger*) используется при отладке/дебаге кода.

Ниже приводятся примеры работы с кодом на C.

Используем такой код:

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <stdlib.h>
4.
5.  char * buf;
6.  int sum_to_n(int num) {
7.
8.      int i,sum=0;
9.      for(i=1;i<=num;i++)
10.         sum+=i;
11.
12.     return sum;
13. }
14.
15. void printSum() {
16.
17.     char line[10];
18.
19.     printf("Enter a number: ");
20.     fgets(line, 10, stdin);
21.
22.     if(line != NULL)
```

```
23.             strtok(line, "\n");
24.
25.         sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
26.
27.         printf("%s\n",buf);
28.     }
29.
30.     int main(void) {
31.
32.         printSum();
33.         return 0;
34.     }
```

Собираем его с опцией -g, что бы включить отладочную информацию:

Terminal

```
$ gcc debug.c -g -o debug
```

Re-play Copy to Clipboard Pause Full View

Запускаем, и получаем ошибку:

Terminal

```
$ ./debug
Enter a number: 1
Segmentation fault
```

Re-play Copy to Clipboard Pause Full View

В результате получаем **Segmentation fault**, которая сигнализирует об ошибочном обращении к памяти.

Теперь используем gdb, что бы найти причину проблемы.

Содержание

Запуск gdb

Запускаем gdb, и первым аргументом передаём исполняемый файл для запуска:

Terminal

```
$ gdb debug
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
...
Reading symbols from debug...done.
(gdb)
```

Re-play Copy to Clipboard Pause Full View

```
run
```

Теперь можно начать выполнение программы, используя команду run (или r):

Terminal

```
(gdb) r
Starting program: /home/admin/Scripts/debug
Enter a number: 1
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7aaebcd in __GI__IO_default_xsputn (f=0x7fffffffe340, data=
<optimized out>, n=4) at genops.c:450
450     genops.c: No such file or directory.
```

Re-play Copy to Clipboard Pause Full View

```
backtrace
```

backtrace указывает gdb на необходимость вывести список всех вызываемых функций из стека программы:

Terminal

```
(gdb) backtrace
$ 0  0x00007ffff7aaebcd in __GI__IO_default_xsputn (f=0x7fffffffe340, data=<optimized out
>, n=4) at genops.c:450
$ 1  0x00007ffff7a80e36 in _IO_vfprintf_internal (s=s@entry=0x7fffffffe340, format=format@e
ntry=0x5555555549b7 "sum=%d", ap=ap@entry=0x7fffffffe468) at vfprintf.c:1320
$ 2  0x00007ffff7aa3afb in __IO_vsprintf (string=0x555555554720 <_start> "1\355I\211\321^H
\211\342H\203\344\360PTL\215\005Z\002", format=0x5555555549b7 "sum=%d", args=args@entry=0x7
fffffffe468) at iovsprintf.c:42
$ 3  0x00007ffff7a89357 in __sprintf (s=<optimized out>, format=<optimized out>) at sprint
f.c:32
$ 4  0x00005555555548ef in printSum () at debug.c:26
$ 5  0x000055555555490c in main () at debug.c:33
```

Re-play Copy to Clipboard Pause Full View

Тут видно, что debug.c в строке 33 вызывает printSum(), printSum () в строке 22 вызывает sprintf(), которая далее начинает выполнение более низкоуровневых функций, и в результате падает.

Всё, что выполняется после sprintf() нам неподконтрольно, и проблема возникает в данных, которые мы передаём в sprintf(), так что давайте изучим их внимательнее:

```
1.  ...
```

```
2.  26      sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
3.  ...
```

Далее - используем breakpoint-ы, что бы изучить значения переменных.

Break Point

Для того, что бы приостановить выполнение программы на каком-то этапе - мы можем задать один или цепочку брейкпоинтов, используя break (b).

Например, что бы сделать паузу перед вызовом sprintf() на строке 26 - указываем break 26, и запускаем программу заново - run (r):

Terminal

```
(gdb) b 26
Breakpoint 1 at 0x8c2: file debug.c, line 26.
(gdb) r
Starting program: /home/admin/Scripts/debug
Enter a number: 1
Breakpoint 1, printSum () at debug.c:26
26          sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
```

Re-play Copy to Clipboard Pause Full View

print

С помощью print (p) можно получить текущие значения переменных.

Тут в буфер sprintf() мы передаём переменую line - проверим её значение:

Terminal

```
(gdb) print line
$ 1 = "1\000\000IUUUU\000"
```

Re-play Copy to Clipboard Pause Full View

Видим нашу единицу, за ней null terminator - '\0', далее - мусор. С этим всё хорошо.

Проверим содержимое buf:

Terminal

```
(gdb) print buf
$ 2 = 0x0
```

Re-play Copy to Clipboard Pause Full View

Теперь проблема становится очевидной: мы пытаемся скопировать данные в буфер, на который указывает buf, но под него не выделена память, что и приводит к Segmentation fault.

К счастью - тут buf является глобальной переменной и была проинициализирована со значением 0 (*null pointer*). Если бы это было не так, и она находилась бы внутри функции - мы получили бы какое-то произвольное значение вида:

Terminal

```
(gdb) p buf
$ 2 = 0x555555554720 <_start> "1\355I\211\321^H\211\342H\203\344\360PTL\215\005Z\002"
```

Re-play Copy to Clipboard Pause Full View

Попробуем исправить наш код - добавим вызов malloc() для выделения памяти под buf:

```
1.  ...
2.  void printSum() {
3.
4.      char line[10];
5.
6.      printf("Enter a number: ");
7.      fgets(line, 10, stdin);
8.
9.      if(line != NULL)
10.         strtok(line, "\n");
11.
12.     char * buf = malloc(1024);
13.     sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
14.
15.     printf("%s\n",buf);
16. }
17. ...
```

list

После обновления исходного кода - номера строк изменились.

Что бы увидеть исходный код из самого gdb - используем list (l):

Terminal

```
(gdb) l
18          printf("Enter a number: ");
19          fgets(line, 10, stdin);
20
21          if(line != NULL)
22              strtok(line, "\n");
```

```
23
24          char * buf = malloc(1024);
25          sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
26
27          printf("%s\n",buf);
```

Re-play Copy to Clipboard Pause Full View

`condition` – break point с условиями

gdb позволяет задать условные точки паузы, т.е. пауза будет выполнена если условие верно.

Например – сделать паузу, если значение переменной `line` будет равно “1” можно так:

1. задаём точку паузы – `(gdb) b 25`
2. задаём условие использовать breakpoint под номером 1, используя функцию `$_streq` – `(gdb) condition 1 $_streq(line, "1")`

Проверяем:

Terminal

```
(gdb) b 25
Breakpoint 1 at 0x920: file debug.c, line 25.
(gdb) condition 1 $_streq(line, "1")
(gdb) r
Starting program: /home/admin/Scripts/debug
Enter a number: 1
Breakpoint 1, printSum () at debug.c:25
25          sprintf(buf,"sum=%d", sum_to_n(atoi(line)));
```

Re-play Copy to Clipboard Pause Full View

Enter a number: 1 – тут в `line` мы передаём единицу в виде string, условие срабатывает – получаем остановку выполнения.

При другом значении – программа выполнится полностью:

Terminal

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/admin/Scripts/debug
Enter a number: 2
sum=3
[Inferior 1 (process 3404) exited normally]
```

Re-play Copy to Clipboard Pause Full View

См. 10.12 Convenience Functions и 5.1.6 Break Conditions.

Аналогично используется $_regex.

Зададим условие, но иначе – прямо при указании брейкпоинта, без использования condition, и используем if для самого break:

Terminal

```
(gdb) break 27 if $_regex(buf, "^sum=1")
Breakpoint 1 at 0x920: file debug.c, line 27.
```

Re-play Copy to Clipboard Pause Full View

Запускаем, и указываем 1:

Terminal

```
(gdb) r
Starting program: /home/admin/Scripts/debug
Enter a number: 1
Breakpoint 1, printSum () at debug.c:27
27          printf("%s\n",buf);
```

Re-play Copy to Clipboard Pause Full View

Если в *sum=* будет другое значение – условие не сработает:

Terminal

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/admin/Scripts/debug
Enter a number: 2
sum=3
[Inferior 1 (process 3663) exited normally]
```

Re-play Copy to Clipboard Pause Full View

Возвращаясь к проблеме с buf – проверим её значение сейчас, после того, как мы добавили malloc():

Terminal

```
(gdb) p buf
```

```
$ 2 = 0x555555756830 "sum=1"
```

Re-play Copy to Clipboard Pause Full View

next, step, until - шаги выполнения

- next (n): выполняет текущую инструкцию, и переходит к началу следующей
- step (s): аналогична next, но с отличиями: когда вы находитесь в начале функции, и используете next - функция будет выполнена полностью, и вернёт значение. Если использовать step - выполнение перейдёт к первой строке внутри функции
- until (u): аналогична next, но в случае, если вы находитесь в начале цикла - выполнение будет продолжаться до конца выполнения этого цикла

Пример.

Запускаем программу, задаём точку на строке 8 - перед началом цикла:

```
1.    1 #include <stdio.h>
2.    2 #include <string.h>
3.    3 #include <stdlib.h>
4.    4
5.    5 int sum_to_n(int num) {
6.    6
7.    7     int i,sum=0;
8.    8     for(i=1;i<=num;i++)
9.    9         sum+=i;
10.  10
11.  11     return sum;
12.  12 }
13.  ...
```

Terminal

```
(gdb) b 8
Breakpoint 1 at 0x8ae: file debug.c, line 8.
```

Re-play Copy to Clipboard Pause Full View

Начинаем выполнение:

Terminal

```
(gdb) r
Starting program: /home/admin/Scripts/debug
Enter a number: 1
Breakpoint 1, sum_to_n (num=1) at debug.c:8
8           for(i=1;i<=num;i++)
```

Re-play Copy to Clipboard Pause Full View

Остановились в начале цикла.

Пробуем next:

```
                                    Terminal

(gdb) n
9               sum+=i;

                                                Re-play  Copy to Clipboard  Pause  Full View
```

Ещё раз next – снова возвращаемся к началу цикла:

```
                                    Terminal

(gdb)
8           for(i=1;i<=num;i++)

                                                Re-play  Copy to Clipboard  Pause  Full View
```

Но если вызовем until – весь цикл будет пройден полностью:

```
                                    Terminal

(gdb) u
11          return sum;
(gdb) p sum
$ 1 = 1

                                                Re-play  Copy to Clipboard  Pause  Full View
```

Другие команды

- list (l): уже упоминалась выше, отобразит исходный код
- delete (d): удаляет брейкпоиинты. Если вызвать без агрументов – удалит все точки, если указать номер (d 1) – то удалит заданную по номеру
- clear function_name: удалить все брейкпоинты в указанной функции
- x: отобразить содержимое адреса

Пример использования x.

Получаем адрес переменной sum (используя &):

```
                                    Terminal

(gdb) p &sum
$ 2 = (int *) 0x7fffffffe528
```

Re-play Copy to Clipboard Pause Full View

Получаем значение памяти по адресу *0x7fffffffe528*:

Terminal

```
(gdb) x 0x7fffffffe528
0x7fffffffe528: 0x00000001
```

Re-play Copy to Clipboard Pause Full View

Вот и наша единичка – *0x00000001*:

Terminal

```
(gdb) p sum
$ 3 = 1
```

Re-play Copy to Clipboard Pause Full View

Ссылки по теме

- GDB Tutorial
- How to Debug Using GDB
- 3.9 Options for Debugging Your Program
- 5.2 Continuing and Stepping
- Linux: C – адресное пространство процесса

Similar posts

- 12/13/2014 Windows: установка Cygwin – GCC + make + gbd (3)
- 07/19/2014 C++: отладка с GDB (0)
- 01/29/2019 C: libmysqlclient – примеры работы с MySQL API (0)
- 03/10/2018 What is: Linux namespaces, примеры PID и Network namespaces (0)
- 09/17/2017 Linux: C – адресное пространство процесса (0)

Раздел: C/C++ GNU/Linux utils HOWTO's Scripting/coding Проблемы и решения Метки: C++, debug, gdb

ALSO ON **RTFM: LINUX, DEVOPS И СИСТЕМНОЕ АДМИНИСТРИРОВАНИЕ**

Kubernetes: мониторинг ...

2 года назад • 4 comments

Настройка мониторинга Kubernetes кластера с Prometheus Operator

Jenkins: running workers in ...

год назад • 1 comment

Jenkins Kubernetes Plugin configuration to run Jenkins workers in Kubernetes …

Kubernetes: Evicted поды и Quality of ...

2 года назад • 1 comment

Что такое Quality of Service для подов в Kubernetes, роль requests и limits, и …

0 Comments **RTFM: Linux, DevOps и системное администрирование**

Политика конфиденциальности Disqus **Войти**

Favorite **Твитнуть** **Поделиться** **Новое**

Начать обсуждение...

ВОЙТИ С ПОМОЩЬЮ ИЛИ ЧЕРЕЗ DISQUS

Имя

Прокомментируйте первым.

Подписаться **Добавь Disqus на свой сайтДобавить DisqusДобавить** **Do Not Sell My Data**

Iconic One Theme | Powered by Wordpress