

Отладка программ на языке C

Сергей Лега

legabox@gmail.com

23.11.2012

Типичные ошибки

- Вовремя не аллоцированная память
- Вовремя не освобожденная память
- Чтение данных неверного размера
- Выход за границы выделенной памяти
- Использование неинициализированных данных

Отладчики

- Неопределенное поведение программы
- Низкая производительность программ
- Segfault


Анализаторы кода

- Метод «пристального взгляда»
- valgrind/doctor memory
- gdb

Пример «плохого» кода

```
int n;
```

```
if (n > 0) {  
    <code>  
}
```



Значение переменной n не
определено в момент
исполнения кода

Вывод valgrind'a

```
==18042== Conditional jump or move depends on  
uninitialized value(s)  
==18042== at 0x100000E3E: main (test.c:24)
```

Компиляция и запуск

- Компиляция

```
gcc -g3 <source filename> -o <output filename>
```

- Запуск:

```
valgrind <programm name> <programm's arguments>
```

GDB

- Позволяет «прогнать» исполнение программы вручную

Запуск программы

`gdb <program name>`

- Если есть аргументы командной строки:
`set args <arguments delimited by the space>`

Breakpoint & Watchpoint

- Watchpoint - отладчик остановит программу, когда выражение, на которое ссылается Watchpoint, считывает или изменяется программой.
- Breakpoint — отладчик останавливает программу при входе в код, на который ссылается Breakpoint.

Установка Breakpoint'ов & Watchpoint'ов

- Breakpoint:

`break <имя функции>`

`break <номер строки>`

- Watchpoint:

`watch <expr>` - значение `expr` изменяется

`rwatch <expr>` - значение `expr` считывается

`awatch <expr>` - значение `expr` считывается
или изменяется

Отключение Breakpoint'ов & Watchpoint'ов

- Просмотр информации о текущих breakpoint'х и watchpoint'х:
info breakpoints
или
info watchpoints
- Отключение breakpoint'в и watchpoint'в:
disable <номер breakpoint'а или watchpoint'а>

Удаление Breakpoint'ов

- clear — удалить текущий breakpoint.
- clear <имя функции> - удалить breakpoint на входе в функцию.
- delete — удалить все breakpoint'ы и watchpoint'ы

Запуск программы

После того, как установлены Watchpoint'ы, Breakpoint'ы и аргументы, программу можно запустить:

run

Подробнее о Watchpoint'х

- Область видимости переменной — та часть кода программы, в которой возможно использование данной переменной.
- Watchpoint не может быть установлен, если область видимости переменной, которую хотят исследовать, не пересекается с исследуемым в данный момент кодом.

Подробнее о Watchpoint'х

Для того, чтобы попасть в область видимости нужной переменной, нужно установить breakpoint на функции (ях), в которой (ых) определяется или используется исследуемая переменная

Пример

test.c:

```
int main () {  
    int x = 30;  
    int y = 10;  
    x = y;  
  
    return 0;  
}
```

Компиляция и запуск

Компиляция:

```
gcc -O0 -g3 -o test test.c
```

Запуск:

```
gdb ./test
```

Установка breakpoint'а:

```
break main
```

ЛИСТИНГ ВЫВОДА gdb

```
(gdb) break main
Breakpoint 1 at 0x100000f23: file test.c, line 4.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000100000f23 in main at
gdb_example_2.c:4
(gdb) run
Starting program: <path to binary>/test
Reading symbols for shared libraries +. done
Breakpoint 1, main (argc=1, argv=0x7fff5fbff928) at
test.c:4
4 int x = 30;
```

Теперь установим Watchpoint на переменную x:

Листинг вывода gdb

(gdb) watch x

Hardware watchpoint 2: x

(gdb) info watchpoints

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x0000000100000f23	in main at test.c:4

breakpoint already hit 1 time

2	hw watchpoint	keep	y		x
---	---------------	------	---	--	---

x

Тип breakpoint'a/watchpoint'a

Включен ли breakpoint?

Адрес функции в сегменте кода

Что отслеживает breakpoint?

Продолжение и выполнение по шагам

- `next` — выполнение следующей строки исходного текста, отладка не «уйдет» во вложенные функции.
- `step` - выполнение следующей строки исходного текста, отладка «уходит» во вложенные функции.
- `continue` — продолжение выполнения программы до следующего breakpoint'a.

ЛИСТИНГ ВЫВОДА gdb

```
(gdb) continue
Continuing.
Hardware watchpoint 2: x
Old value = 0
New value = 10
main (argc=1,
argv=0x7fff5fbff928) at test.c:9
9 return 0;
```

Просмотр переменных и исходного кода

- `print <expr>` - показать значение переменной
- `list` — показать исходный код рядом с текущей позицией в файле.
- `backtrace` – просиомтреть стек вызовов функций
- `quit` – выход из gdb 😊

Редактирование исходного текста из консоли GDB

- Редактор по умолчанию для GDB — ex, чтобы использовать свой редактор надо добавить в ~/.bash_rc строку:
`export EDITOR=<путь редактору>`
- Режим редактирования включается командой `edit`

Спасибо за внимание!

Материалы лекций можно найти:

- <http://ilab.mipt.ru>
- <http://wiki.ilab.mipt.ru>
- http://groups.google.com/group/ilab_edu