"ПРОГРАММИРОВАНИЕ"

Отладка

Преподаватель:

Перышкова Евгения Николаевна

Введение

W http://ru.wikipedia.org/wiki/Отладка_программы

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

W http://ru.wikipedia.org/wiki/Отладчик

Отладчик — компьютерная программа, предназначенная для поиска ошибок в других программах, ядрах операционных систем, SQL-запросах и других видах кода.

Основные функции отладчика

Пошаговое выполнение программы:

- Построчное выполнение кода
- Установка точек останова (условных или безусловных)

Операции с переменными:

- Просмотр значений
- Установка новых значений
- Отслеживание изменения значения переменной

Перемещение по стеку вызовов

Пример

```
#include <stdio.h>

#define N 10

int main()
{
    int arr[N];
    unsigned i;

    for (i = N - 1; i >= 0; --i) {
        arr[i] = i;
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Пример

```
$ gcc -Wall -o example main.c
$ ./example
Bus error (core dumped)
```

Компиляция проходит
без ошибок и предупреждений

Программа завершается
аварийно.
(Строка "Bus error (core dumped"
может отсутствовать!)

Пример

```
$ gcc -Wall -g -O0 -o example main.c
$ gdb ./example
(gdb) run
Program received signal SIGBUS, Bus error.
0x0000000000400509 in main () at main.c:11
11                arr[i] = i;
(gdb) print i
$1 = 4294967295
(gdb) break main
(gdb) run
(gdb) Breakpoint 1, main () at main.c:10
10 for (i = N - 1; i >= 0; --i) {
(gdb) display i
(gdb) next
. . .
(gdb) next
10          for (i = N - 1; i >= 0; --i) {
1: i = 0
(gdb) next
1: i = 4294967295
```

Опции компилятора

- **-g –** добавляет в выходной файл отладочную информацию в поддерживаемом операционной системой формате (stabs, COFF, XCOFF, DWARF2)
  -g0 – без отладочной информации
  -g1 – минимальный уровень информации, достаточный для просмотра backtrace.
  -g2 – уровень отладки по умолчанию (то же, что и -g).
  -g3 – включение дополнительной информации, например, определения макросов

- **-O** – определяет уровень оптимизации.
  Возможные значения:
  -O0 – без оптимизации
  -O1, **-O2**, **-O3 –** уровни оптимизации
  -Os – оптимизация по объему кода
  -Ofast – все оптимизации **-O3**, плюс дополнительные

Трассировка

Трассировка – процесс *пошагового* исполнения программы.

В режиме трассировки программист видит последовательность выполнения операторов, а также может запросить текущие значения переменных на данном шаге выполнения программы. Это упрощает процесс обнаружения ошибок.

Трассировка может быть начата и окончена в любом месте программы.

Выполнение программы может останавливаться:

1) на каждой команде;
2) на точках останова.

Трассировка может выполняться *с заходом* в процедуры и без него.

Трассировка (без оптимизации)

`gcc -O0 -S main.c`

```
int a = 10;
```

```
movl    $10, -12(%rbp)
```

```
int b = 20;
```

```
movl    $20, -8(%rbp)
```

```
int c = a + b;
```

```
movl    -8(%rbp), %eax
movl    -12(%rbp), %edx
addl    %edx, %eax
movl    %eax, -4(%rbp)
```

```
printf("c = %d\n", c);
. . .
```

```
movl    $.LC0, %eax
movl    -4(%rbp), %edx
movl    %edx, %esi
movq    %rax, %rdi
movl    $0, %eax
call    printf
```

Трассировка (первый уровень оптимизации)

`gcc -O1 -S main.c`

```
int a = 10;
```

```
int b = 20;
```

```
int c = a + b;
```

```
movl    $30, %edx
```

```
printf("c = %d\n", c);
. . .
```

```
movl    $.LC0, %esi
movl    $1, %edi
movl    $0, %eax
call    __printf_chk
```

Трассировка (третий уровень оптимизации)

`gcc -O3 -S main.c`

```
int a = 10;
int b = 20;
printf("c = %d\n", c);
int c = a + b;
printf("c = %d\n", c);
. . .
```

```
movl    $1, %edi
movl    $0, %eax
movl    $30, %edx
movl    $.LC0, %esi
call    __printf_chk
```

GDB

Типы точек останова:

- breakpoint – остановка выполнения в конкретной точке программы (номер строки, функция)
- watchpoint – остановка выполнения в случае изменения значения заданной области памяти
- catchpoint – остановка выполнения при возникновении заданного события

В документации GDB для всех трех типов точек может использоваться слово breakpoint.

Управление точками останова

```
break <имя_функции>
(gdb) break main

break <номер_строки>
(gdb) break 35

break <имя_файла>:<номер_строки>
(gdb) break source/file.c:35

break <имя_файла>:<имя_функции>
(gdb) break source/file.c:parse_arguments
```

Управление точками останова (2)

watch выражение
- **остановка при изменении значения выражения**

rwatch выражение
- **остановка при чтении программой значения выражения**

awatch выражение
- **остановка при чтении или сохранении программой значения выражения**

Использование точек останова

```
1 #include <stdio.h>
2 int f(int x)
3 {
4     int y = x * 10;
5     return y;
6 }
7  int main(){
8     int k = 5, l;
9     l = f(k);
10    printf("l = %d\n",l);
11    return 0;
12 }
```

```
(gdb) break 4
Breakpoint 1 at 0x400533: example.c, line 4.
(gdb) run
Starting program: example
Breakpoint 1, f (x=5) at example.c:4
4   int y = x * 10;
```

оператор на момент остановки не выполнен!

Список точек останова

```
(gdb) break main
Breakpoint 2 at 0x8048824: file efh.c, line 16.


(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08048846 in Init_Game at efh.c:26
2 breakpoint keep y 0x08048824 in main at efh.c:16
breakpoint already hit 1 time
3 hw watchpoint keep y efh.level
4 catch fork keep y

(gdb) delete 1 3 4
```

Команды пошагового исполнения

После останова программы в заданной точке, обычно, требуется произвести ее пошаговое исполнение. Для этого применяются команды **next** (**n**) и **step** (**s**).

- Команда **next** осуществляет переход к следующему оператору текущей функции или (если в данной функции больше нет операторов) к следующему оператору вызывающей функции. Заход в вызываемые функции данная программа не предусматривает.
- Команда **step** работает аналогично **next**, однако, если исполняемый оператор является вызовом функции, то будет осуществлен заход внутрь нее.

Команды пошагового исполнения (2)

```
#include <stdio.h>
int f(int x){
     int y = x * 10;
     return y;
}
int main(){
     int k = 5, l;
     l = f(k);
     printf("l = %d\n",l);
     return 0;
}
```

```
(gdb) break main
(gdb) run
9    int k = 5, l;
(gdb) next
10   l = f(k);
(gdb) n
11   printf("l = %d\n",l);
(gdb) n
l = 50
12   return 0;
```

```
(gdb) b main
(gdb) r
9   int k = 5, l;
(gdb) step
10 l = f(k);
(gdb) s
f (x=5) at step_in_over.cpp:4
4   int y = x * 10;
(gdb) s
5   return y;
(gdb) s
6   }
(gdb) s
main () at step_in_over.cpp:11
11   printf("l = %d\n",l);
(gdb) s
l = 50
12   return 0;
(gdb) s
13    }
```

Просмотр значений переменных

```
1 struct Point{ int x; int y; };
2 int main()
3 {
4     struct Point pts[] = {{0, 0}, {0, 1}, {2, 0}};
5     return 0;
6 }
```

```
(gdb) break 5
Breakpoint 1 at 0x4004e2: file point.c, line 5.
(gdb) run
Breakpoint 1, main () at point.c:5
10          return 0;
(gdb) print pts
$1 = {{x = 0, y = 0}, {x = 0, y = 1}, {x = 2, y = 0}}
(gdb) p *pts
$2 = {x = 0, y = 0}
(gdb) print pts[1]
$3 = {x = 0, y = 1}
```

Передача аргументов командной строки

Передача параметров в программу на языках C/C++ осуществляется через аргументы функции main(). Каждый параметр – строка.

Первый аргумент – целочисленный, обычно он называется argc (arguments counter). Через него в функцию main передается количество параметров командной строки.

Второй аргумент – массив указателей на строки (тип char **), обычно его называют argv (arguments vector). Он содержит указатели на строки, которые были указаны при вызове программы

```
#include <stdio.h>

int main(int argc, char** argv)
{
    int i;
    for (i = 0; i < argc; ++i) {
        printf("%d: %s", argc, argv[i]);
    }
    return 0;
}
```

Передача аргументов командной строки

```
$ gdb --args ./example arg1 arg2 arg3
(gdb) show args
Argument list to give program being debugged
 when it is started is "arg1 arg2 arg3".
(gdb) break main
Breakpoint 1 at 0x400503: file args.c, line 6
(gdb) run

Breakpoint 1, main (argc=5, argv=0x7fffffffe6c8) at args.c:6
6             for (i = 0; i < argc; ++i) {
(gdb) print argc
$1 = 4
(gdb) print argv[1]
$2 = arg1
```

Передача аргументов командной строки

```
$ gdb ./example
(gdb) break main
Breakpoint 1 at 0x400503: file args.c, line 6
(gdb) run arg1 arg2 arg3

Breakpoint 1, main (argc=5, argv=0x7fffffffe6c8) at args.c:6
6             for (i = 0; i < argc; ++i) {
(gdb) print argc
$1 = 4
(gdb) print argv[1]
$2 = arg1
(gdb) show args
Argument list to give program being debugged
 when it is started is “arg1 arg2 arg3".
```

GDB: получение справки

```
help ['all' |
      имя класса команд |
      команда]
```

Получение списка классов команд:

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
running -- Running the program
status -- Status inquiries
. . .
```

Пример 1

```
#include <stdio.h>

void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main() {
    int a = 0;
    int b = 1;

    printf("a = %d, b = %d\n", a, b);
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);

    return 0;
}
```

Пример 1

```
(gdb) b swap
Breakpoint 1 at 0x400537: file lecture.c, line 5.
(gdb) b main
Breakpoint 2 at 0x400553: file lecture.c, line 12.
(gdb) r
Breakpoint 2, main () at main.c:12
12     int a = 0;
(gdb) n
13 int b = 1;
(gdb) n
14     swap(a, b);
```

Пример 1

```
14      swap(a, b);                    // точка вызова функции swap
(gdb) p &a
$3 = (int *) 0x7fffffffdd18
(gdb) p &b
$4 = (int *) 0x7fffffffdd1c
(gdb) s                                // шаг в функцию swap
swap (a=0, b=1) at main.c:5
5           int tmp = a;
(gdb) p &a
$5 = (int *) 0x7fffffffdcec
(gdb) p &b
$6 = (int *) 0x7fffffffdce8
```

Задание 1

```
#include <stdio.h>
#include <stdlib.h>

void init(int* arr, int n) {
    arr = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i)
        arr[i] = i;
}

int main() {
    int* arr = NULL;
    int n = 10;
    init(arr, n);
    int i;
    for (i = 0; i < n; ++i)
        printf("%d\n", arr[i]);
    return 0;
}
```

Пример 2

```
#include <stdio.h>

struct ArrayPair
{
    int first[4];
    int second[4];
};
int main()
{
    struct ArrayPair ap = {{1, 2, 3, 4}, {1, 2, 3, 4}};
    int i;
    for (i = 0; i <= 4; ++i) {
        ap.first[i] = i;
    }
    for (i = 0; i <= 4; ++i) {
        printf("%d <=> %d\n", ap.first[i], ap.second[i]);
    }
    return 0;
}
```

Пример 2

```
(gdb) b main
Breakpoint 1 at 0x400534: file example2.c, line 10.
(gdb) r
Starting program: example2
Breakpoint 1, main () at example2.c:10
10 struct ArrayPair ap = {{1, 2, 3, 4}, {1, 2, 3, 4}};
(gdb) watch ap.second[0]
Hardware watchpoint 2: ap.second[0]
(gdb) c
Continuing.
Hardware watchpoint 2: ap.second[0]

Old value = 1
New value = 4
main () at example2.c:12
12        for (i = 0; i <= 4; ++i) {
(gdb) inspect i
$1 = 4
```

Задание 2

```
#include <stdio.h>

typedef struct {
    char str[4];
    int num;
} NumberRepr;

void format(NumberRepr* number) {
    sprintf(number->str, "%4d", number->num);
}

int main() {
    NumberRepr number = { .num = 1025 };
    format(&number);
    printf("str: %s\n", number.str);
    printf("num: %d\n", number.num);
    return 0;
}
```

Пример 3

```
#include <stdio.h>

#define QUOTE_(WHAT) #WHAT
#define QUOTE(WHAT) QUOTE_(WHAT)
#define CSC_TRACE(format, ...) \
    printf("%s: "format, __FILE__":"QUOTE(__LINE__), \
    ## __VA_ARGS__)

void f()
{
    int answer = 42;
    CSC_TRACE("the answer is %d\n", answer);
}

int main()
{
    f();
    return 0;
}
```

Пример 3

```
(gdb) break main
Breakpoint 1 at 0x400523

(gdb) run

Breakpoint 1, main () at csc_trace.c:16
16          f();
(gdb) macro expand CSC_TRACE("Hello")
expands to: printf("%s: ""Hello",__FILE__":""__LINE__")
```

Задание 3

```
#define SQR(x) x * x

int main()
{
    int y = 5;
    int z = SQR(y + 1);
    printf("z = %d\n", z);
    return 0;
}
```

Пример 4

```
#include <stdio.h>
#include <stdlib.h>

int char_cmp(const void* a, const void* b)
{
    return *(const char*)a - *(const char*)b;
}

int main()
{
    char greeting[] = "Hello!";

    qsort(greeting, sizeof(greeting),
        sizeof(*greeting), char_cmp);
    printf("%s\n", greeting);

    return 0;
}
```

Пример 4

```
(gdb) b 14
Breakpoint 1 at 0x40061f: file sort_string.c, line 14.
(gdb) run

Breakpoint 1, main () at sort_string.c:14
14          printf("%s\n", greeting);
(gdb) print greeting
$1 = "\000!Hello"
```

Задание 4

```
void bubble_sort(int* array, int size)
{
    int i, j;
    for (i = 0; i < size - 1; ++i) {
        for (j = 0; j < size - i; ++j) {
            if (array[j] > array[j + 1]) {
                swap(&array[j], &array[j + 1]);
            }
        }
    }
}
```