Паламарчук Андрей ИУ7-23Б

# **Отладка**

## **Отладка при помощи gdb**

## **Задача №1**

Скомпилируем и запустим программу задачи №1, не сложно заметить, что она работает некорректно:

$ gcc -std=c99 task\_01.c

$ ./a.out

Input n: 5

factorial(5) = 0

$

Перекомпилируем наш исполняемый файл для работы с отладчиком **gdb**, а после установим точку останова на функции factorial, затем поставим точку наблюдения на переменную n.

$ gcc -std=c99 -g3 task\_01.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38  
...  
Reading symbols from ./a.out...  
(gdb) break factorial  
Breakpoint 1 at 0x4011b2: file task\_01.c, line 26.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Input n: 5  
  
Breakpoint 1, factorial (n=5) at task\_01.c:26  
26        long long unsigned result = 1;  
(gdb) watch n  
Hardware watchpoint 2: n  
(gdb) continue  
Continuing.  
  
Hardware watchpoint 2: n  
  
Old value = 5  
New value = 4  
factorial (n=4) at task\_01.c:28  
28        while (n--)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: n  
  
Old value = 4  
New value = 3  
factorial (n=3) at task\_01.c:28  
28        while (n--)  
(gdb)  
Continuing.

Hardware watchpoint 2: n  
  
Old value = 3  
New value = 2  
factorial (n=2) at task\_01.c:28  
28        while (n--)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: n  
  
Old value = 2  
New value = 1  
factorial (n=1) at task\_01.c:28  
28        while (n--)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: n  
  
Old value = 1  
New value = 0  
factorial (n=0) at task\_01.c:28  
28        while (n--)  
(gdb)

После обнуления переменной n цикл не заканчивается, умножая на ноль всё произведение.

Изменим функцию factorial:

|  |  |
| --- | --- |
| Исходная функция | Исправленная функция |
| long long unsigned factorial(unsigned n)  {  long long unsigned result = 1;  while (n--)  result \*= n;  return result;  } | long long unsigned factorial(unsigned n)  {  long long unsigned result = 1;  while (n)  {  result \*= n;  n--;  }  return result;  } |

Проверим корректность работы программы:

$ gcc -std=c99 task\_01.c

$ ./a.out

Input n: 5

factorial(5) = 120

$

Программа работает корректно.

## **Задача №2**

Скомпилируем и запустим программу задачи №2:

$ gcc -std=c99 task\_02.c

$ ./a.out  
Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [1] is 5  
Value [2] is 1663166608  
Value [3] is 32527  
Value [4] is 0  
The average is 0  
The max is 0  
$

Программа работает некорректно. Перекомпилируем наш исполняемый файл для работы с отладчиком **gdb**.

Будем отслеживать наш массив с помощью точки наблюдения.

$ gcc -std=c99 –g3 task\_02.c

$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38  
...  
Reading symbols from ./a.out...  
(gdb) break main  
Breakpoint 1 at 0x40113e: file task\_02.c, line 14.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Breakpoint 1, main () at task\_02.c:14  
14        printf("Enter %d numbers:\n", N);  
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.37-1.fc38.x86\_64  
(gdb) watch arr  
Hardware watchpoint 2: arr  
(gdb) continue  
Continuing.  
Enter 5 numbers:  
Enter the next number: 9  
  
Hardware watchpoint 2: arr  
  
Old value = {0, 0, -134322032, 32767, 0}  
New value = {0, 9, -134322032, 32767, 0}  
0x00007ffff7e337e0 in \_\_vfscanf\_internal () from /lib64/libc.so.6  
(gdb)  
Continuing.  
Enter the next number: 8  
  
Hardware watchpoint 2: arr  
  
Old value = {0, 9, -134322032, 32767, 0}  
New value = {0, 8, -134322032, 32767, 0}  
0x00007ffff7e337e0 in \_\_vfscanf\_internal () from /lib64/libc.so.6  
(gdb)  
Continuing.  
Enter the next number: 7  
  
Hardware watchpoint 2: arr  
  
Old value = {0, 8, -134322032, 32767, 0}  
New value = {0, 7, -134322032, 32767, 0}  
0x00007ffff7e337e0 in \_\_vfscanf\_internal () from /lib64/libc.so.6  
(gdb)  
Continuing.  
Enter the next number: 6  
  
Hardware watchpoint 2: arr  
  
Old value = {0, 7, -134322032, 32767, 0}  
New value = {0, 6, -134322032, 32767, 0}  
0x00007ffff7e337e0 in \_\_vfscanf\_internal () from /lib64/libc.so.6  
(gdb)  
Continuing.  
Enter the next number: 5  
  
Hardware watchpoint 2: arr  
  
Old value = {0, 6, -134322032, 32767, 0}  
New value = {0, 5, -134322032, 32767, 0}  
0x00007ffff7e337e0 in \_\_vfscanf\_internal () from /lib64/libc.so.6  
(gdb)  
Continuing.  
Value [1] is 5  
Value [2] is -134322032  
Value [3] is 32767  
Value [4] is 0  
The average is 0  
The max is -134322032

Можно заметить, что программа записывает входные значения только во второй элемент. Это означает, что есть ошибка в записи массива.

Изменим цикл записи массива:

|  |  |
| --- | --- |
| Исходный | Исправленный |
| for (i = 0; i < N; i++)  {  printf("Enter the next number: ");  if (scanf("%d", &arr[1]) != 1)  {  printf("Input error");  return 1;  }  } | for (i = 0; i < N; i++)  {  printf("Enter the next number: ");  if (scanf("%d", &arr[i]) != 1)  {  printf("Input error");  return 1;  }  } |

Массив вводится корректно.

Проверяем. Видим ошибку вывода массива:

$ gcc -std=c99 task\_02.c

$ ./a.out  
Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [1] is 8  
Value [2] is 7  
Value [3] is 6  
Value [4] is 5  
The average is 0  
The max is 5  
$

Будем отслеживать индекс i в цикле for после ввода массива:

$ gcc -std=c99 -g3 task\_02.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 26  
Breakpoint 1 at 0x4011b6: file task\_02.c, line 26.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
  
Breakpoint 1, main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.37-1.fc38.x86\_64  
(gdb) watch i  
Hardware watchpoint 2: i  
(gdb) continue  
Continuing.  
  
Hardware watchpoint 2: i  
  
Old value = 5  
New value = 1  
main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
(gdb)  
Continuing.  
Value [1] is 8  
  
Hardware watchpoint 2: i  
  
Old value = 1  
New value = 2  
0x00000000004011e3 in main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
(gdb)  
Continuing.  
Value [2] is 7  
  
Hardware watchpoint 2: i  
  
Old value = 2  
New value = 3  
0x00000000004011e3 in main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
(gdb)  
Continuing.  
Value [3] is 6  
  
Hardware watchpoint 2: i  
  
Old value = 3  
New value = 4  
0x00000000004011e3 in main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
(gdb)  
Continuing.  
Value [4] is 5  
  
Hardware watchpoint 2: i  
  
Old value = 4  
New value = 5  
0x00000000004011e3 in main () at task\_02.c:26  
26        for (i = 1; i < N; i++)  
(gdb)  
Continuing.  
The average is 0  
The max is 5

Массив выводится с 1 индекса, а не с 0.

Изменим цикл вывода массива:

|  |  |
| --- | --- |
| Исходный | Исправленный |
| for (i = 1; i < N; i++)  printf("Value [%zu] \  is %d\n", i, arr[i]); | for (i = 0; i < N; i++)  printf("Value [%zu] \  is %d\n", i, arr[i]); |

Массив выводится корректно.

Продолжаем, следующая ошибка связана со средним числом.

Начнем с функции расчёта среднего. Будем отслеживать индекс i.

$ gcc -std=c99 -g3 task\_02.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 40  
Breakpoint 1 at 0x401252: file task\_02.c, line 40.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [0] is 9  
Value [1] is 8  
Value [2] is 7  
Value [3] is 6  
Value [4] is 5  
  
Breakpoint 1, get\_average (a=0x7fffffffdf10, n=5) at task\_02.c:40  
40        for (size\_t i = 0; i > n; i++)  
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.37-1.fc38.x86\_64  
(gdb) watch i  
Hardware watchpoint 2: i  
(gdb) continue  
Continuing.  
  
Hardware watchpoint 2: i  
  
Old value = 140737488347224  
New value = 0  
get\_average (a=0x7fffffffdf10, n=5) at task\_02.c:40  
40        for (size\_t i = 0; i > n; i++)  
(gdb)  
Continuing.  
  
Watchpoint 2 deleted because the program has left the block in  
which its expression is valid.  
0x00000000004011fb in main () at task\_02.c:29  
29        printf("The average is %g\n", get\_average(arr, N));  
(gdb)  
Continuing.  
The average is 0  
The max is 5

Условие в цикле было не выполнено, ошибка найдена.

Изменим цикл:

|  |  |
| --- | --- |
| Исходный | Исправленный |
| for (size\_t i = 0; i > n; i++)  temp += a[i]; | for (size\_t i = 0; i < n; i++)  temp += a[i]; |

Поиск среднего работает корректно:

$ gcc -std=c99 task\_02.c

$ ./a.out

Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [0] is 9  
Value [1] is 8  
Value [2] is 7  
Value [3] is 6  
Value [4] is 5  
The average is 7  
The max is 5

$

Следующая ошибка связана с поиском максимума. Будем отслеживать переменную max.

[Natalia@fedora my\_lab\_3]$ gcc -std=c99 -g3 task\_02.c  
[Natalia@fedora my\_lab\_3]$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 51  
Breakpoint 1 at 0x4012f7: file task\_02.c, line 51.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [0] is 9  
Value [1] is 8  
Value [2] is 7  
Value [3] is 6  
Value [4] is 5  
The average is 7  
  
Breakpoint 1, get\_max (a=0x7fffffffdf10, n=5) at task\_02.c:51  
51        for (size\_t i = 1; i < n; i++)  
(gdb) watch max  
Hardware watchpoint 2: max  
(gdb) continue  
Continuing.  
  
Hardware watchpoint 2: max  
  
Old value = 9  
New value = 8  
get\_max (a=0x7fffffffdf10, n=5) at task\_02.c:51  
51        for (size\_t i = 1; i < n; i++)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: max  
  
Old value = 8  
New value = 7  
get\_max (a=0x7fffffffdf10, n=5) at task\_02.c:51  
51        for (size\_t i = 1; i < n; i++)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: max  
  
Old value = 7  
New value = 6  
get\_max (a=0x7fffffffdf10, n=5) at task\_02.c:51  
51        for (size\_t i = 1; i < n; i++)  
(gdb)  
Continuing.  
  
Hardware watchpoint 2: max  
  
Old value = 6  
New value = 5  
get\_max (a=0x7fffffffdf10, n=5) at task\_02.c:51  
51        for (size\_t i = 1; i < n; i++)  
(gdb)  
Continuing.  
  
Watchpoint 2 deleted because the program has left the block in  
which its expression is valid.  
0x0000000000401225 in main () at task\_02.c:31  
31        printf("The max is %d\n", get\_max(arr, N));  
(gdb)  
Continuing.  
The max is 5

В переменной max сохраняется минимум, а не максимум. Ошибка в условии.

Изменим условие:

|  |  |
| --- | --- |
| Исходное | Исправленное |
| if (max > a[i])  max = a[i]; | if (max < a[i])  max = a[i]; |

Проверим корректность работы программы:

$ gcc -std=c99 task\_02.c

$ ./a.out

Enter 5 numbers:  
Enter the next number: 9  
Enter the next number: 8  
Enter the next number: 7  
Enter the next number: 6  
Enter the next number: 5  
Value [0] is 9  
Value [1] is 8  
Value [2] is 7  
Value [3] is 6  
Value [4] is 5  
The average is 7  
The max is 9

$

Программа работает корректно.

## **Задача №3**

Скомпилируем и запустим программу задачи №3:

$ gcc -std=c99 task\_03.c

$ ./a.out

5 div 2 = 2

Исключение в операции с плавающей точкой (образ памяти сброшен на диск)

$

Программа завершается с ошибкой во время 2-го вызова функции div. Установим точку останова на строку, где это происходит. Проверим действия программы с помощью *step*.

$ gcc -std=c99 -g3 task\_03.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 14  
Breakpoint 1 at 0x401172: file task\_03.c, line 14.  
(gdb) r  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
5 div 2 = 2  
  
Breakpoint 1, main () at task\_03.c:14  
14        printf("%d div %d = %d\n", a, b, div(a, b));  
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.37-1.fc38.x86\_64  
(gdb) step  
div (a=10, b=0) at task\_03.c:21  
21        return a / b;  
(gdb)  
  
Program received signal SIGFPE, Arithmetic exception.  
0x00000000004011af in div (a=10, b=0) at task\_03.c:21  
21        return a / b;  
(gdb)  
  
Program terminated with signal SIGFPE, Arithmetic exception.  
The program no longer exists.

Происходит деление на ноль, что приводит к ошибке. Исправим это с помощью добавления проверки.

Изменим функцию div:

|  |  |
| --- | --- |
| Исходная | Исправленная |
| int div(int a, int b) {  return a / b; } | int div(int a, int b) {  if (b == 0)  return 1;  return a / b; } |

Проверим корректность работы программы:

$ gcc -std=c99 task\_03.c

$ ./a.out

5 div 2 = 2

10 div 0 = 1

$

Программа работает корректно.

## **Таблица размеров типов данных**

|  |  |  |
| --- | --- | --- |
| Тип переменной | Размер в Fedora Linux x64 | Размер в Windows 10 x64 |
| char | 1 | 1 |
| int | 4 | 4 |
| unsigned | 4 | 4 |
| long long | 8 | 8 |
| short | 2 | 2 |
| int32–t | 4 | 4 |
| int64–t | 8 | 8 |

## **Представление переменных различных типов в памяти**

Напишем небольшую программу, чтобы узнать, как положительные и отрицательные переменные различных типов представлены в памяти.

#include <stdio.h>

int main(void)  
{

char pos\_char = 35;  
char neg\_char = -35;  
int pos\_int = 104;  
int neg\_int = -104;  
unsigned pos\_unsigned = 27;  
unsigned neg\_unsigned = -27;  
long long pos\_long = 339;  
long long neg\_long = -339;

return 0;  
}

Узнаем, как различные переменные представлены в памяти с помощью отладчика **gdb**.

Установим точку останова на 12 строке, тогда мы сможем узнать все переменные, ведь они будут проинициализированы.

Запустим нашу программу командой *run* и выведем все значения в двоичном представлении(команда *x*).

$ gcc -std=c99 -g3 var\_example.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 12  
Breakpoint 1 at 0x40113e: file var\_example.c, line 12.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out

Breakpoint 1, main () at var\_example.c:12  
12        return 0;  
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.37-1.fc38.x86\_64  
(gdb) x  
Argument required (starting display address).  
(gdb) x /1tb &pos\_char  
0x7fffffffdf2f:    00100011  
(gdb) x /1tb &neg\_char  
0x7fffffffdf2e:    11011101  
(gdb) x /4tb &pos\_int  
0x7fffffffdf28:    01101000    00000000    00000000    00000000  
(gdb) x /4tb &neg\_int  
0x7fffffffdf24:    10011000    11111111    11111111    11111111  
(gdb) x /4tb &pos\_unsigned  
0x7fffffffdf20:    00011011    00000000    00000000    00000000  
(gdb) x /4tb &neg\_unsigned  
0x7fffffffdf1c:    11100101    11111111    11111111    11111111  
(gdb) x /8tb &pos\_long  
0x7fffffffdf10:    01010011    00000001    00000000    00000000  
00000000    00000000    00000000    00000000  
(gdb) x /8tb &neg\_long  
0x7fffffffdf08:    10101101    11111110    11111111    11111111  
11111111    11111111    11111111    11111111

Отрицательные варианты значений - это инвертированные положительные + 1.

## **Представление массива целых данных в памяти**

Напишем небольшую программу, чтобы узнать, представление массива целых данных в памяти. Длина массива 10, размер одного int равен 4 байта, длина массива 40 байт.

#include <stdio.h>

int main(void)

{

int arr[] = {1, 7, 53, 98, 133, 177, 265, 568, 1356, 4582};

int \*f\_el = &arr[0];

printf("Первый элемент массива равен %d\n", \*f\_el);

printf("Десятый элемент массива равен %d\n", \*(f\_el + 9));

return 0;

}

Представление массива в памяти:

$ gcc -std=c99 -g3 arr\_example.c  
$ gdb ./a.out  
GNU gdb (GDB) Fedora Linux 13.1-2.fc38

...  
Reading symbols from ./a.out...  
(gdb) break 11  
Breakpoint 1 at 0x4011b5: file arr\_example.c, line 11.  
(gdb) run  
Starting program: /home/Natalia/practic\_PTP/Task\_3/my\_lab\_3/a.out  
Первый элемент массива равен 1  
Десятый элемент массива равен 4582  
  
Breakpoint 1, main () at arr\_example.c:11  
11        return 0;  
(gdb) x /40tb arr  
0x7fffffffdf00:    00000001    00000000    00000000    00000000  
00000111    00000000    00000000    00000000  
0x7fffffffdf08:    00110101    00000000    00000000    00000000  
01100010    00000000    00000000    00000000  
0x7fffffffdf10:    10000101    00000000    00000000    00000000  
10110001    00000000    00000000    00000000  
0x7fffffffdf18:    00001001    00000001    00000000    00000000  
00111000    00000010    00000000    00000000  
0x7fffffffdf20:    01001100    00000101    00000000    00000000  
11100110    00010001    00000000    00000000

Операция сложения указателя и целого числа эквивалента прибавлению к адресу произведения этого целого числа и размера типа указателя.

## **Использование точки наблюдения**

Программа, что считает степень числа (а в данном случае – десятую степень двойки) и что содержит ошибку, которую очень удобно исправить благодаря точке наблюдения:

#include <stdio.h>  
  
#define N 5  
  
  
double get\_average(const int a[], size\_t n)  
{  
    double temp = 0.0;  
  
    for (size\_t i = 0; i < n; i--)  
        temp += a[i];  
    temp /= n;  
  
    return temp;  
}  
  
  
int main()  
{  
    int arr[N];  
    size\_t i;  
  
    printf("Enter %d numbers:\n", N);  
  
    for (i = 0; i < N; i++)  
    {  
        printf("Enter the next number: ");  
        if (scanf("%d", &arr[i]) != 1)  
        {  
            printf("Input error");  
            return 1;  
        }  
    }  
  
    for (i = 0; i < N; i++)  
        printf("Value [%zu] is %d\n", i, arr[i]);  
  
    printf("The average is %g\n", get\_average(arr, N));  
  
    return 0;  
}

Такую программу будет легко отладить, если проследить за переменной i с помощью команды *watch*.

## **Сопоставление gdb и Qt Creator**

|  |  |  |
| --- | --- | --- |
| **Суть команды/действия** | **gdb** | **Qt Creator** |
| Выбор файла для отладки | file <имя\_файла> | Файл main.c |
| Вывод кода программы | list | Окно редактора кода |
| Выполнение программы вплоть до точки останова | run | Кнопка «Начать отладку запускающего проекта» или клавиша F5 |
| Установить точку останова | break | Нажать в редакторе кода слева от номера нужной строки |
| Выполнить следующую строку, минуя функцию | next | В окне отладчика кнопка «Перейти через» |
| Выполнить следующую строку, войдя в функцию | step | В окне отладчика «Войти в» |
| Вывод значения переменной на данный момент | print <имя\_переменной> | Отображается в отдельном окне, что справа от редактора кода |
| Выполнение программы до следующей точки останова | continue | В окне отладчика кнопка «Продолжить GDB для “<имя\_проекта>”» |
| Выход из функции | finish | В окне отладчика «Выйти из функции» |
| Изменение значения переменной | set <имя\_переменной>=<новое\_значение> | В окне отображения переменных есть возможность изменить их значение |
| Анализ стека | bt  frame <номер> | Находится в окне «Стек» |
| Просмотр потоков программы | info thread | Находится в окне «Потоки» |
| Просмотр точек останова | info breakpoints | Находится в окне «Точки останова» |
| Просмотр аргументов и локальных переменных | info frame  info locals  info args | ПКМ по окну, где находятся переменные, после «открыть редактор памяти», а затем «открыть просмотрщик памяти на адрес стека» |
| Установка точек наблюдения | watch <выражение>  rwatch <выражение>  awatch <выражение> | В окне отображения переменных данная информация обновляется автоматически |
| Вывод содержимого блоков памяти | x [/nfu] <адрес>, где  n – сколько единиц памяти должно быть выведено  f – спецификатор формата  u – размер выводимой единицы памяти | ПКМ по окну, где находятся переменные, после «открыть редактор памяти», а затем «открыть просмотрщик памяти на адрес объекта» |