**Смирнов Иван ИУ7-22Б - 2023г.**

**Отчет**

**Задание №3.1**

**Отладка**

*Целью* работы является умение студента самостоятельно производить трассировку приложения.

1. **Задание №1**
2. Чтобы посмотреть программу в отладчике, необходимо скомпилировать программу с ключом -g. Если попытаться открыть программу в отладчике, не указав данный ключ, то отладчик напишет, что не видит отладочной информации в исполняемом файле (No debugging symbols found in app.exe).
3. Чтобы запустить программу под отладчиком необходимо прописать команду “gdb app.exe” с учетом того, что в app.exe содержится отладочная информация. Чтобы досрочно завершить программу необходимо написать “q” или комбинацию клавиш “Ctrl+D”. В случае отсутствия дальнейших точек останова можно написать команду “continue”, в таком случае программа выполнится до конца.
4. Для просмотра информации о том, на какой точке останова программа была остановлена, необходимо прописать команду “info breakpoints”. Рядом с названием точки останова будет надпись “breakpoint already hit 1 time”, а также строка кода программы и её номер.
5. Посмотреть значение переменной можно с помощью команды “print {назв. перем.}”. Чтобы изменить значение переменной необходимо прописать команду “set var {назв. перем.}={значение}”.
6. Чтобы выполнить программу в пошаговом режиме, существуют 4 команды:

Step – выполняет текущую строку и останавливается на следующем операторе для выполнения (если строка – функция, то step останавливается в её начале).

Next - выполняет текущую строку и останавливается на следующем операторе для выполнения (если строка – функция, то next её выполняет и останавливается на следующем операторе).

Continue – продолжает обычное выполнение программы до следующей точки останова.

Finish – выполняет команды next без остановки, пока не достигнет конца текущей функции.

1. Последовательность вызова функций покажет команда “bt” (работает на основе текущего кадра стека).
2. Чтобы установить точку останова в программе необходимо прописать команду “break {номер строки}”. Программа остановится на строке с указанным номером.
3. Команда “tbreak” создает временную точку останова. Точка останова считается временной, если в ходе отладки программы отладчик останавливается на данной точке только 1 раз.
4. Выключить/включить точку останова – команда “disable/enable [номер|диапазон]”. Для игнорирования нескольких срабатываний используем команду “ignore {номер} {кол\_во итераций}”.
5. Чтобы задать условие на точке останова необходимо прописать команду “break {номер строки} if {условие}”. Если необходимо добавить условие к существующей точке останова – использует команду “condition {номер} {условие}”.
6. Точка останова останавливает программу всякий раз, когда её выполнение достигает определенной точки. Точка наблюдения – это специальная точка останова, которая останавливает программу при изменении значения выражения.
7. Поскольку точка наблюдения выдает старое значение переменной и новое, то логично её использовать в сложном выражении, в котором вы могли забыть приоритет операций языка Си.
8. Содержимое области памяти можно посмотреть с помощью команды “x [/nfu] [адрес]”, где n – сколько единиц памяти должно быть выведено, f – спецификатор формата, u – размер выводимой единицы памяти.

Исправленная программа (1):

#include <stdio.h>

long long unsigned factorial(unsigned n);

int main(void)

{

    unsigned n;

    long long unsigned result;

    printf("Input n: ");

    if (scanf("%u", &n) != 1)

    {

        printf("Input error");

        return 1;

    }

    result = factorial(n);

    printf("factorial(%u) = %llu\n", n, result);

    return 0;

}

long long unsigned factorial(unsigned n)

{

    long long unsigned result = 1;

    while (n>0)

    {

        result \*= n;

        n--;

    }

    return result;

}

Исправленная программа (2):

#include <stdio.h>

#define N 5

double get\_average(const int a[], size\_t n);

int get\_max(const int \*a, size\_t n);

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));

    printf("The max is %d\n", get\_max(arr, N));

    return 0;

}

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 get\_max(const int \*a, size\_t n)

{

    int max = a[0];

    for (size\_t i = 1; i < n; i++)

        if (a[i] > max)

            max = a[i];

    return max;

}

Исправленная программа (3):

#include <stdio.h>

int div(int a, int b, int \*c);

int main(void)

{

    int a = 5, b = 2;

    int c, err;

    err = div(a, b, &c);

    if (err != 0)

        return 1;

    printf("%d div %d = %d\n", a, b, div(a, b, &c));

    a = 10;

    b = 0;

    err = div(a, b, &c);

    if (err != 0)

        return 1;

    printf("%d div %d = %d\n", a, b, div(a, b, &c));

    return 0;

}

int div(int a, int b, int \*c)

{

    if (b == 0)

        return 1;

    \*c = (a / b);

    return 0;

}

1. **Задание №2**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Ubuntu 22.04.01 LTS (размер [Б])** | **Windows 11**  **(размер [Б])** |
| char | 1 | 1 |
| int | 4 | 4 |
| unsigned | 4 | 4 |
| long long | 8 | 8 |
| short | 2 | 2 |
| int32\_t | 4 | 4 |
| int64\_t | 8 | 8 |

1. **Задание №3**

Заданы переменные a1, a2, a3, a4 соответствующих типов char, int, unsigned, long long. Значения пременных равно 100:

(gdb) x /1xb &a1

0x7fffffffdf2f: 0x64

(gdb) x /4xb &a2

0x7fffffffdf30: 0x64 0x00 0x00 0x00

(gdb) x /4xb &a3

0x7fffffffdf34: 0x64 0x00 0x00 0x00

(gdb) x /8xb &a4

0x7fffffffdf38: 0x64 0x00 0x00 0x00 0x00 0x00 0x00 0x00

(gdb) info locals

a1 = 100 'd'

a2 = 100

a3 = 100

a4 = 100

Значения переменных равно -100:

(gdb) x /1xb &a1

0x7fffffffdf2f: 0x9c

(gdb) x /4xb &a2

0x7fffffffdf30: 0x9c 0xff 0xff 0xff

(gdb) x /4xb &a3

0x7fffffffdf34: 0x9c 0xff 0xff 0xff

(gdb) x /8xb &a4

0x7fffffffdf38: 0x9c 0xff 0xff 0xff 0xff 0xff 0xff 0xff

(gdb) info locals

a1 = -100 '\234'

a2 = -100

a3 = 4294967196

a4 = -100

Вывод: все типы, кроме unsigned, хранят знак (unsigned принимает только положительные значения).

1. **Задание №4**

Код программы:

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#define OK 0

int main(void)

{

int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int \*pn = a;

printf("adress - %p, value - %d\n", pn, \*pn);

(\*pn)++;

printf("adress - %p, value - %d\n", pn, \*pn);

pn++;

printf("adress - %p, value - %d\n", pn, \*pn);

return OK;

}

Вывод:

ivan@ivan-VirtualBox:~/Рабочий стол/study/sem-2$ ./app.exe

adress - 0x7ffdde0c47d0, value - 1

adress - 0x7ffdde0c47d0, value - 2

adress - 0x7ffdde0c47d4, value – 2

Убедимся с помощью gdb, что pn – указатель:

adress - 0x7fffffffdf10, value - 1

Breakpoint 1, main () at main.c:12

12 (\*pn)++;

(gdb) x /10uw a

0x7fffffffdf10: 1 2 3 4

0x7fffffffdf20: 5 6 7 8

0x7fffffffdf30: 9 10

(gdb) x /10xg a

0x7fffffffdf10: 0x0000000200000001 0x0000000400000003

0x7fffffffdf20: 0x0000000600000005 0x0000000800000007

0x7fffffffdf30: 0x0000000a00000009 0x9b151030f44e2100

0x7fffffffdf40: 0x0000000000000001 0x00007ffff7c29d90

0x7fffffffdf50: 0x0000000000000000 0x0000555555555169

(gdb) cont

Continuing.

adress - 0x7fffffffdf10, value - 2

Breakpoint 2, main () at main.c:14

14 pn++;

(gdb) x /10uw a

0x7fffffffdf10: 2 2 3 4

0x7fffffffdf20: 5 6 7 8

0x7fffffffdf30: 9 10

(gdb) x /10xg a

0x7fffffffdf10: 0x0000000200000002 0x0000000400000003

0x7fffffffdf20: 0x0000000600000005 0x0000000800000007

0x7fffffffdf30: 0x0000000a00000009 0x9b151030f44e2100

0x7fffffffdf40: 0x0000000000000001 0x00007ffff7c29d90

0x7fffffffdf50: 0x0000000000000000 0x0000555555555169

(gdb) cont

Continuing.

adress - 0x7fffffffdf14, value - 2

Breakpoint 3, main () at main.c:16

16 return OK;

(gdb) x /10uw a

0x7fffffffdf10: 2 2 3 4

0x7fffffffdf20: 5 6 7 8

0x7fffffffdf30: 9 10

(gdb) x /10xg a

0x7fffffffdf10: 0x0000000200000002 0x0000000400000003

0x7fffffffdf20: 0x0000000600000005 0x0000000800000007

0x7fffffffdf30: 0x0000000a00000009 0x9b151030f44e2100

0x7fffffffdf40: 0x0000000000000001 0x00007ffff7c29d90

0x7fffffffdf50: 0x0000000000000000 0x0000555555555169

1. **Задание №5**

Точка наблюдения нужна тогда, когда переменная изменяется неявно. Дана программа:

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#define OK 0

void func(void\* a)

{

\*(int\*)a = 30;

}

int main(void)

{

short a = 10;

short b = 20;

func(&a);

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

return OK;

}

Подразумевается, что программа изменит значение переменной a, а значение переменной b останется преждним. Однако, запустив программу получаем следующее:

ivan@ivan-VirtualBox:~/Рабочий стол/study/sem-2$ ./app.exe

a=30, b=0

Значение переменной b почему-то изменилось на 0, хотя нигде не прописано действий с ней. Поставим на неё точку наблюдения:

(gdb) watch b

Hardware watchpoint 2: b

(gdb) cont

Continuing.

Hardware watchpoint 2: b

Old value = 0

New value = 20

main () at main.c:16

16 func(&a);

(gdb) cont

Continuing.

Hardware watchpoint 2: b

Old value = 20

New value = 0

func (a=0x7fffffffdf34) at main.c:10

10 }

(gdb) cont

Continuing.

a=30, b=0

Первая остановка отвечает за инициализацию переменной b и присвоения числа 20. Вторая остановка присваивает значение 0. Мы видим, что это происходит в функции func, значит в ней и кроется ошибка. Для её решения необходимо поменять в функции тип int на тип short.

1. **Задание №6**

Ниже приведена таблица команд gdb и соответствующих команд qt creator:

|  |  |  |
| --- | --- | --- |
| Смысл команды | GDB | QT CREATOR |
| Начать отладку | gdb app.exe | F5 или “Отладка > Начать отладку” |
| Поставить точку останова | break 17 | Навестись на поле слева от строки 17 и нажать ЛКМ (или нажать F9) |
| Удалить точку останова | delete {номер} | Щелкнуть на красную точку слева от строки или “Точки останова > Удалить точку останова” |
| Остановить отладку | q | SHIFT+F5 |
| Выполняет текущую строку (не учитывая функцию) | step | F11 |
| Выполняет текущую строку (учитывая функцию) | next | F10 |
| Продолжить отладку | continue | F9 или “Отладка > Продолжить” |
| Стек | bt | “Отладка > Виды > Стек” |
| Значение переменной | print {перем.} | В виде “Локальные и наблюдаемые переменные” можно посмотреть информацию |

**Список литературы:**

1. Курс “Проектно-технологическая практика (знакомство с Linux)”:

<https://e-learning.bmstu.ru/iu7/course/view.php?id=76>

1. Крус “Проектно-технологическая практика (тестирование, отладка и профилирование ПО)”:

<https://e-learning.bmstu.ru/iu7/course/view.php?id=73>

1. Официальная документация gcc:

<https://gcc.gnu.org/onlinedocs/gcc/Invoking-GCC.html>

1. Документация “Отладка с помощью GDB”

<http://linux.yaroslavl.ru/docs/altlinux/doc-gnu/gdb/gdb.html>

1. Документация GDB “Остановка и продолжение остановки”

<https://www.opennet.ru/docs/RUS/gdb/gdb_6.html>

1. Документация GDB “Изменение исполнения”

<https://www.opennet.ru/docs/RUS/gdb/gdb_12.html>

1. Документация GDB “Исследование данных”

<https://www.opennet.ru/docs/RUS/gdb/gdb_9.html>

1. Документация “Отладка с помощью QT Creator”

<http://doc.crossplatform.ru/qtcreator/1.2.1/creator-debugging.html>

1. Документация “Функция printf и форматы вывода”

<https://kaf401.rloc.ru/Informatics/formats.htm>

1. Документация ‘x-command”

<https://visualgdb.com/gdbreference/commands/x>