Слуцкий Никита | гр. 053506 (ФКСиС, ИиТП)

Лабораторная работа № 6

**Задание 1**

1 . Когда вы находитесь в сеансе gdb, как вы устанавливаете аргументы, которые будут передаваться программе при её запуске.

После запуска gdb можно запустить программу, используя “r ags”.

Если код запускается с помощью *$ main.exe …args*, то в gdb это будет происходить как *$ gdb* main, *$ r …args*

2 . Как создать точку останова ?

Можно создать по номеру строки или по названию процедуры.

*$ b main* или *$ b 3*

3 . Как выполнить следующую строку кода в Си-программе после остановки в точке останова ?

С помощью *n*. При нажатии *n* отладчик пройдёт функцию, не заходя в неё.

4 . Если следующая строка кода является вызовом функции, вы выполните весь вызов функции сразу, если воспользуетесь своим ответом на вопрос №3. Как вы сообщите gdb, что хотите вместо этого перейти внутрь функции ?

Необходимо сделать “шаг внутрь” (*step*-*in*) или нажать клавишу *s*.

5 . Как продолжить выполнение программы после остановки на точке останова ?

*Continue* (*c*) продолжит выполнение работы до следующей точки останова или ошибки.

6 . Как можно распечатать значение переменной в gdb ? (или даже выражение типа 1 + 2)

С помощью команды *print*.

7 . Как настроить gdb так, чтобы оно отображал значение переменной после каждого шага ?

С помощью команды *watch*.

8. Как показать список всех переменных и их значений в текущей функции ?

С помощью команды *info args*.

9. Как выйти из gdb ?

Команда *quit* или ввести знак конца файла.

**Задание 2**

Заменил прямой ввод с клавиатуры на чтение из файла.

**Задание 3**

1 . Почему важны инструменты вроде Valgrind и санитайзеров, какая в них польза ?

Отслеживание утечек памяти, валидности указателей и т.д.

2 . Как запустить программу в Valgrind ?

Во-первых, это нужно делать в Linux. Во-вторых, *valgrind ./segfault\_ex*

3 . Как использовать санитайзер G++ ?

Флаг *-fsanitize=address* включает санитайзер, *-static-libasan* статически связывает программу с библиотекой санитайзера. Флаг *-g* нужен для отображения символов отладки при запуске. После того, как программа собрана с включённым санитайзером, она запускается обычным способом.

4 . Как интерпретировать сообщения об ошибках ?

Для простейшего подхода получение информации об ошибках состоит в том, чтобы читать вывод, например:   
*==17413== at 0ч47Ф2349: ns\_name\_ntop (ns\_name.c: 147)*можно взглянуть на имя файла, номер строки, имя функции.

5 . Почему неинициализированные переменные могут приводить к появлению Гейзенбагов ?

Использование неинициализированных переменных схоже с использованием неинициализированной памяти и может приводить к ошибкам разного рода в процессе работы программы.   
Предположим, что мы не инициализировали переменную в функции. Объявили, но без значения. В отдельных случаях там может оказаться нулевое значение (например, глобальные переменные по умолчанию имеют значение 0). Но в общем случае программа может вести себя неправильно из-за того, что, например, в int переменные при отсутствии дефолтного значения попадает “мусорное” значение, например, 461818864.

***Вопросы***

1 . Почему не произошёл segfault При no\_segfault\_ex ?

Потому что мы не присваиваем элемент по индексу, превышающему размер массива, а всего лишь пытаемся получить значение по этому индексу. Segfault вылетел бы при попытке получить именно доступ к памяти, а не просто значение в ней.

2 . Почему no\_segfault\_ex выдаёт несогласованные выходные данные ?

Потому что в определении цикла мы задаём *j < sizeof(a)*, а так нельзя, потому что *sizeof* возвращает размер массива в байтах, а не количество его элементов, и при итерации по массиву мы обращаемся к несуществующим элементам и, следовательно, при вычислении суммы мы получаем несогласованные выходные данные.

3 . Почему указан неправильный размер ? Как можно было исправить код, используя *sizeof* ?

*sizeof* вернёт размер всего массива в байтах. Можно (и нужно) разделить размер всего массива на размер одного элемента. *sizeof(a)/sizeof(int)*

**Задание 4**

1 . Какая цель является частью правила, удаляющего все скомпилированные программы ?

*clean:*

*-rm -rf core \*.o \*~ "#"\*"#" Makefile.bak $(BINARIES) \*.dSYM*

2 . Какая цель является частью правила, которое создаёт все скомпилированные программы ?

*all: $(BINARIES)*

3 . Какой компилятор сейчас используется ?

GNU

4 . Какой стандарт Си мы используем в настоящее время ?

*C11*

5 . Как мы можем ссылаться на переменную “FOO” в Make-файле ?

*$(FOO)*

6 . Для чего используется переменная CFLAGS ?

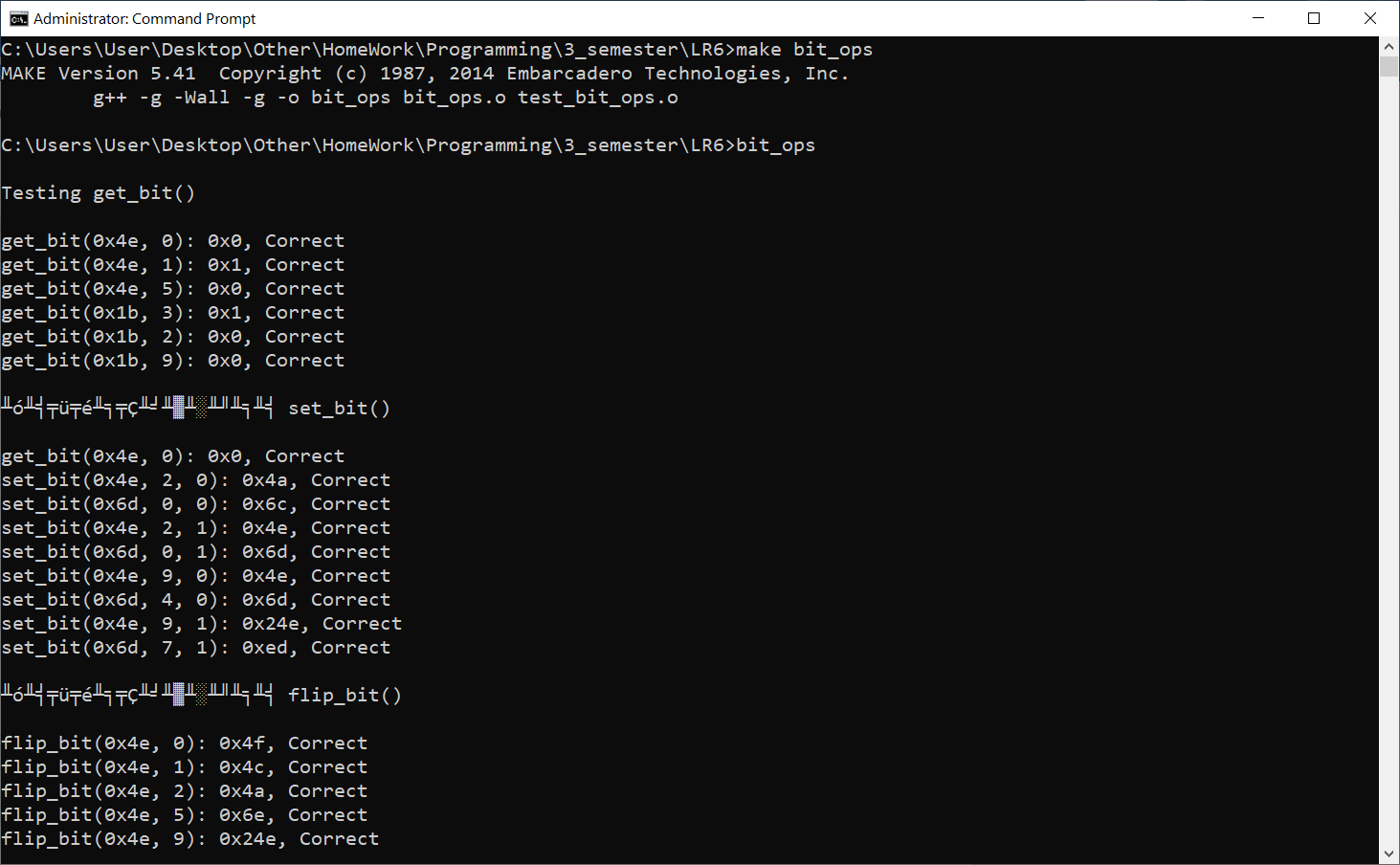
В переменной CFLAGS лежат флаги, которые передаются компилятору для компиляции.

7 . Какая строка создаёт программу bit-ops из её объектных файлов ?

*$(BIT\_OPS\_PROG): $(BIT\_OPS\_OBJS)*

*$(CC) $(CFLAGS) -g -o $(BIT\_OPS\_PROG) $(BIT\_OPS\_OBJS) $(LDFLAGS)*

**Задание 5**



Пример работы с GNUDeBugger (GDB):

