МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«__» _____ 2017г.

ОТЛАДКА ПРОГРАММ С ПОМОЩЬЮ GDB

Методические указания по выполнению лабораторной работы по дисциплинам «Основы реверсинжениринга программных средств», «Методы защиты программного обеспечения» для студентов специальности 10.03.01

Курск 2017

УДК 004.056.55 (076.5)

Составитель А.Л. Марухленко

Рецензент
Кандидат технических наук, доцент *И.В. Калуцкий*

Отладка программ с помощью GDB: методические указания по выполнению лабораторных работ по дисциплине «Основы реверсинжениринга программных средств», «Методы защиты программного обеспечения» / Юго-Зап. гос. ун-т; сост. А.Л. Марухленко. Курск, 2017. - 17с.

Рассматривается метод отладки программ с помощью GDB. Указывается порядок выполнения лабораторной работы и содержание отчета.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по образованию в области информационной безопасности (УМО ИБ).

Предназначены для студентов специальности 10.03.01.

Текст печатается в авторской редакции

Подписано в печать 01.11.2017. Формат 60х84 1/16.
Усл.печ. л. 1,0. Уч.-изд.л. 0,9. Тираж 30 экз. Заказ _____. Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ОГЛАВЛЕНИЕ.

1. Цель работы: 4
2. Теоретические сведения 4
3. Задание на лаболаторную работу 15
4. Контрольные вопросы.....16
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....17

1. ЦЕЛЬ РАБОТЫ

Научиться использовать инструментальные средства, помогающие провести отладку приложений.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1) Общие сведения о разработке программного обеспечения

Стадии и этапы разработки программного обеспечения определены в Единой системе программной документации (ЕСПД) ГОСТом 19.102-77 сновная разработка программного обеспечения осуществляется на этапе формирования рабочего проекта. Этот этап включает в себя процессы: разработки программного обеспечения, его отладки, создания программной документации и проведения испытаний. Последовательность и/или одновременность выполнения процессов зависит от применяемой технологии разработки программного обеспечения.

В терминах государственного стандарта под *отладкой* понимается процесс, позволяющий получить программное обеспечение, которое при заданных входных данных дает определённый результат. Этот процесс заключается в выполнении двух процедур: «тестирования» и «локализации исправления ошибок».

Тестирование — это последовательность действий, направленных на обнаружение несоответствий функционирования программного обеспечения требованиям технического задания на его разработку. Конкретные причины несоответствий определяются в процедуре *локализации и исправления ошибок*. Именно эту процедуру, по аналогии с англоязычным термином debug, в российской литературе называют дебагингом или отладкой. Очевидно, что процедуры тестирования и локализации и исправления ошибок дополняют друг друга.

Испытание – это процесс демонстрации программного обеспечения с целью фиксации факта выполнения им всех функций, определённых в техническом задании. Испытание – это элемент сдачи готового проекта заказчику. По сути, этот процесс аналогичен процедуре тестирования программного обеспечения.

2) Процедура локализации и исправления ошибок

Рассмотрим процедуру локализации и исправления ошибок подробнее. Далее по тексту термины «процедура локализации и исправления ошибок» и «отладка» будут считаться синонимами.

Это интересно. Термин баг (англ. *bug* —насекомое) применяется для обозначения ошибки

в программном обеспечении. Считается, что этот термин в компьютерной индустрии появился в 1945 году благодаря учёной из Гарвардского университета Грейс Хоппер, которая во время тестирования вычислительной машины Mark II Aiken Relay Calculator, нашла в ней мотылька, застрявшего между контактами электромеханического реле. Извлечённое насекомое было

вклеено скотчем в технический дневник, с сопроводительной надписью: «First actual case of bug being found» («первый реальный случай, когда был найден жук»). В действительности слово «bug» задолго до этого употреблялось персоналом телеграфных и телефонных компаний в отношении неполадок с электрооборудованием и радиотехникой. Во время Второй мировой войны словом «bugs» назывались проблемы с радарной электроникой.

Отладка заключается в выполнении (пошаговом) программы и анализе её поведения, состояния вычислительных ресурсов и их изменений. В процессе отладки используются заранее определённые входные данные. В некоторых случаях производится принудительное изменение состояния вычислительных ресурсов и поведения программы.

Процедура локализации и исправления ошибок может осуществляться с применением:

– механизма исключений. Операционная система вносит ряд ограничений на допустимые действия, которые может выполнять программное обеспечение. Программа принудительно завершается или выдает некоторое сообщение при возникновении недопустимых событий (исключений);

– журналирования (вывода) текущего состояния программного обеспечения. Обычно эта процедура производится с помощью расположенных в критических точках программы операторов вывода. Программа может самостоятельно завершиться при возникновении некоторых событий;

– отладчиков — специальных системных программ, которые позволяют пошагового выполнить программное обеспечение и понаблюдать за изменением состояния вычислительных ресурсов.

Описанные выше способы отладки расположены в порядке увеличения функционала, доступного разработчику программного обеспечения. В случае отладки с применением механизма исключений возможно лишь удостовериться, что программа не выполняет никаких недопустимых действий. Проверить каким образом изменяются вычислительные ресурсы в процессе выполнения программы и вмешаться в этот процесс невозможно. Во втором случае программное обеспечение самостоятельно сообщает какие изменения им производятся, но вмешаться в сам процесс выполнения программы невозможно. При этом, если в процессе отладки возникает необходимость получить дополнительные сведения об изменениях вычислительных ресурсов, то это становится возможным только после изменения исходного кода программы, её перекомпоновки и повторного запуска программы на выполнение. С применением отладчика доступен самый полный функционал

– на любом этапе выполнения программы возможно посмотреть состояние всех вычислительных ресурсов и, при необходимости, изменить их или даже изменить саму исполняющуюся программу.

Следует отметить, что с применением указанных выше процедур процесс отладки программного обеспечения может осуществляться либо поэтапно (т.е. каждая отдельная функционально завершенная его часть, например, процедура, функция, модуль и т.п., проверяется отдельно), либо единовременно. Очевидно, что в степень сложность отладки в первом случае может быть значительно меньше, чем во втором.

3) Отладчик программного обеспечения GDB. Общие сведения

На рынке системного программного обеспечения представлены различные отладчики, отличающиеся реализуемым функционалом, поддерживаемыми языками программирования, средами исполнения программ и предназначенные для использования в различных операционных системах. В среде GNU/Linux наибольшее распространение получил отладчик **GDB** (сокр. от англ. GNU Debugger), изучению которого посвящена лабораторная работа.

Отладчик GDB первоначально был разработан Ричардом Столмененом в 1988 году. За основу GDB был взят отладчик DBX, поставлявшийся в то время с дистрибутивом операционной системы BSD. В настоящее время разработка отладчика координируется Управляющим комитетом GDB (GDB Steering Committee). В действующей версии (7.6) отладчик поддерживает приложения, написанный на языках: Си, Си++, Фортран, Java, Chill, ассемблер и Модула-2.

Работа GDB осуществляется в командном режиме. Существуют надстройки и интегрированные среды разработки программного обеспечения, использующие GDB и реализующие графический интерфейс к нему (например DDD, Eclipse и т.п.). В данной лабораторной работе рассматривается «родной» командный интерфейс отладчика (см. рисунок **1**).

Отметим, что далее в примерах сеансов работы с командной строкой и отладчиком GDB, символом $ в начале строки обозначается приглашение командной оболочки, символами (gdb) –

приглашение отладчика GDB. В примерах командных строк символами [] обозначаются необязательные параметры, – - короткая опция (состоящая из одного символа, короткие опции могут группироваться), -- - длинная опция, | - альтернативные значения параметра, <> - нажатие клавиш или их комбинации. Подробнее о всех параметрах командной строки изучаемых программ можно найти в справочнике MAN.

```
$ gdb ./a.out
(gdb) run
Starting program: /home/user/a.out
```

Рисунок **1** – Пример сеанса работы отладчика GDB

Для запуска отладчика в интерактивном режиме используется команда (см. рисунок **2**), в которой не указано ни одного параметра. Если пользовать хочет указать исполняемый файл программы, которую необходимо отладить, то он указывает её первым параметром. В этом случае отладчик самостоятельно создаст процесс, в котором будет исполняться и отлаживаться указанная программа. Если необходимо произвести отладку уже исполняющегося процесса, то пользователь указывает в качестве второго параметра его номер. В случае, если необходимо проанализировать причину аварийной остановки программы, то вторым параметром пользователь указывает имя файла с дампом памяти (core). Дополнительно пользователь может указать опции, уточняющие работу отладчика GDB (см. таблицу ниже).

```
$ gdb [program] [core | process_id] [опции]
```

Рисунок **2** – Команда запуска отладчика GDB

Таблица 1 – Некоторые опции командной строки отладчика GDB

Опция	Назначение
`-help`	Вывод подсказки по параметрам командной строки
`-s file | --symbols file`	Указание файла, содержащего отладочную информацию исследуемой программы
`-x file | --command file`	Требование исполнения последовательности команд, расположенных в файле.
`-batch`	Запуск отладчика в пакетном режиме. Используется для автоматизированного тестирования программного обеспечения.
`-t device | -tty device`	Устройство для перенаправления ввода/вывода исследуемой программы
`--args arg1 agr2 …`	Аргументы командной строки для исследуемой программы. Эта опция всегда является последней.

4) Работа в командном режиме GDB

а. командном режиме отладчик ожидает от пользователя указаний что ему следует сделать. Каждое указание (команда) – это последовательность символов, расположенных в одной строке, набор которых заканчивается нажатием клавиши <ENTER>. Если указание пустое, т.е. не содержащее ни одного символа, то отладчик повторяет предыдущее указание (за исключением некоторых случаев). Длинна строки не имеет ограничений. Символ # имеет специальное значение – это начало комментария. Все символы строки, следующие за этим символом игнорируются.

Каждое указание начинается с команды, которая отделяется от остальной части строки символом пробел. После команды могут следовать параметры, уточняющие команду. Многие команды имеют как длинные имена, так и короткие аналоги.

Для удобства ввода в GDB предусмотрен механизм подсказок по командам, который доступен при нажатии клавиши <TAB>. Набрав начальные символы и нажав <TAB> пользователь получит список команд, которые начинаются с набранных символов. Если такая команда только

одна, то она автоматически подставится в командную строку. Такой же способ можно использовать для подстановки имен переменных и функций в исследуемой программе. Для этого надо перед начальном строки поставить символ ‘ (апостроф).

Для получения подсказки по командам в GDB используется команды help, apropos, complete (см. рисунок **3**). Первая команда выдает краткую справку, вторая – список команд, в которых встречается указанная комбинация символов, третья – список команд, начинающихся на указанную последовательность символов (аналог нажатия клавиши <TAB>). Подробную информацию по командам следует искать в документации, расположенной на официальном сайте проекта.

```
(gdb)help set width
Set number of characters where GDB should wrap lines of its output.
This affects where GDB wraps its output to fit the screen width.
Setting this to zero prevents GDB from wrapping its output.
```

Рисунок **3** – Пример использования команды help.

5) Подготовка программ для отладки с помощью GDB

Отладчик GDB может запустить любую программу, записанную в форматах a.out, COFF, ECOFF, XCOFF, ELF, SOM. Запустив отладку пользователь имеет возможность посмотреть состояние ресурсов,

а также получить информацию об исполняемом коде программы в виде ассемблерных инструкций (см. рисунок 4). Очевидно, что такой способ анализа программы не всегда удобен. Для того, чтобы отладчик мог выводить информацию в понятной разработчику форме (на соответствующем языке высокого уровня), необходимо ему сообщить где находятся файлы с исходными кодами и указать связь между машинными инструкциями и строками программы. Эта информация называется *символьной или отладочной* и располагается она либо в том же исполняемом файле, либо отдельно.

Для того, чтобы в исполняемый файл был дополнен символьной информацией при компиляции программы с помощью GCC следует указать опцию –g (см. рисунок **5**). Дополнительно к опции можно указать параметр gdb, определяющий что символьная информация предназначается для использования отладчиком GDB, а также задать уровень детализации отладочной информации (в примере задан максимальный уровень детализации – 3).

Очевидно, что отладочная информация увеличивает размер исполняемого файла. Допускается разделения исполняемого кода программы и отладочной информации. В дальнейшем символьная информация может быть загружена отладчиком также отдельно.

Выделить отладочную информацию из исполняемого кода можно с помощью утилиты objcopy (см. рисунок **6**.). В первой строке примера создается файл file.debug, в который копируется символьная информация из файла file. Во второй строке в файле file делается служебная запись о том, что символьная информация находится в другом файле. В третьей строке символьная информация удаляется из файла file. Вывести информацию о символьной информации, имеющейся в файле, можно с помощью утилиты objdump с опцией –g.

6) Запуск и остановка процессов в отладчике GDB

Чтобы начать отладку программы в отладчике GDB используется команда run (см. рисунок **7**). Исполняемый файл, который следует запустить для отладки, задается либо в параметрах командной строки запуска отладчика (как показано на рисунке), либо командой file или exec-file. При этом первая команда подразумевает одновременное чтение отладочной информации из указанного файла, а вторая нет (для чтения отладочной информации используется команда symbol-file).

Запуск программ может потребовать определённой настройки их окружения. По умолчанию, вновь создаваемый процесс копирует окружение GDB. Отладчик GDB позволяет для запускаемых программ задать: параметры командной строки, значения переменных среды окружения, рабочий каталог и устройства, ассоциируемые с потоками ввода-вывода, создаваемыми по умолчанию для любого процесса.

```
$ gdb ./f –silent
Reading symbols from /home/user/f...(no debugging symbols
found)...done.
(gdb) break *0x400538
Breakpoint 1 at 0x400538
(gdb) r
Starting program: /home/user/f
Breakpoint 1, 0x0000000000400538 in ?? ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-
4.fc19.x86_64
(gdb) x/20i $pc-16
    0x400528:  jmpq   0x4004a0
    0x40052d:  nopl   (%rax)
    0x400530:  push   %rbp
    0x400531:  mov    %rsp,%rbp
    0x400534:  sub    $0x10,%rsp
 => 0x400538:  addl   $0xc7,-0x4(%rbp)
    0x40053f:  mov    -0x4(%rbp),%eax
    0x400542:  mov    %eax,%esi
    0x400544:  mov    $0x4005f0,%edi
    0x400549:  mov    $0x0,%eax
    0x40054e:  callq  0x400410 <printf@plt>
    0x400553:  mov    $0x0,%eax
    0x400558:  leaveq
    0x400559:  retq
    0x40055a:  nopw   0x0(%rax,%rax,1)
    0x400560:  push   %r15
    0x400562:  mov    %edi,%r15d
    0x400565:  push   %r14
    0x400567:  mov    %rsi,%r14
    0x40056a:  push   %r13
(gdb)
```

Рисунок **4** – Пример отладки программы, не имеющей символьной (отладочной) информации.

```
$ gcc –Wall –ansi –pedantic –ggdb3 –o file file.c
```

Рисунок **5** – Командная строка компиляции программы с включением отладочной информации.

```
$ objcopy --only-keep-debug file file.debug $
objcopy --add-gnu-debuglink=file.debug file $
strip -g file
```

Рисунок **6** –Выделение отладочной информации
из исполняемого файла file в отдельный файл file.debug.

```
$ gdb –silent ./file
Reading symbols from /home/user/file...done.
(gdb)r
Starting program: /home/user/file
Program exited normally.
(gdb)q
```

Рисунок **7** – Запуск программы для отладки.

Задать *параметры командной строки* для запускаемого программного обеспечения можно двумя способами:

- указать необходимые параметры в команде run;
- задать параметры командой set args (посмотреть ранее заданные параметры можно с использованием команды show args).

Переменные среды окружения задаются командой set environment. Вывод действующих значений переменных среды окружения осуществляется командой show environment.

Текущий (рабочий) каталог, в котором операционная система будет искать файлы, заданные относительными именами, задается командой cd. По умолчанию текущим является каталог,

в котором был запущен отладчик. Вывести путь к текущему каталогу можно с помощью команды pwd. В процессе работы программы, естественно, рабочий каталог может изменяться произвольным образом.

Задать файл, с которым должны быть связаны *стандартные потоки ввода и вывода процесса*, задается командой tty. Кроме того в команде run, по аналогии с командной строкой, потоки ввода и вывода могут перенаправляться с помощью символов > и <.

Команда run используется для создания нового процесса с использованием заданного исполняемого файла. Если необходимо произвести отладку уже выполняющегося процесса, то для «подключения» к нему используется команда attach. Исполняемый файл в этом случае используется как источник символьной (отладочной) информации. После подключения к процессу его работа приостанавливается (см. рисунок **8**).

```
$ gdb –silent ./file
Reading symbols from /home/user/file...done.
(gdb)attach 1234
Attaching to program: /home/user/file, process 1234
0x00000036b80d89b0 in __read_nocancel () from /lib64/libc.so.6
(gdb)continue
Continuing.
…
Program exited normally.
(gdb)q
```

Рисунок **8** – Подключение к существующему процессу и его отладка (процесс в момент подключения ожидал ввод информации с терминала).

Прекратить отладку процесса можно либо завершив его (в том числе принудительно), либо отключившись от него. Последнее действие делается с помощью команды detach. Если процесс был создан GDB, то прекращение отладки приведет к принудительному завершению процесса.

Отладчик GDB позволяет *анализировать сразу несколько процессов* (так называемых подчинённых процессов). Чтобы создать ещё один процесс для отладки надо создать дополнительную секцию внутри текущей сессии с помощью команды add-inferior. В качестве параметров команды может быть задана опция с указанием количества создаваемых секций (-copies n) и имя исполняемого файла, который будет анализироваться в новой секции (-exec file). Создать точную копию другой секции можно с помощью команды clone-inferior. Получить список секций можно с помощью команды info inferiors. Переключаться между анализируемыми процессами можно с помощью команды inferior указывая номер нужной секции. Очевидно, что использование таких секций оправдано, например, при отладке многопроцессного приложения.

Отладка программного обеспечения, которое самостоятельно создает новые процессы имеет ряд особенностей. Во-первых, по умолчанию, после вызова функции fork отладчик продолжает взаимодействовать только с родительским процессом. Изменить это поведение можно

с использованием команды set follow-fork-mode. Во-вторых, по умолчанию, дочерний процесс не попадает в режим отладки (т.е. выполняется в режиме detach). Изменить это поведение можно с использованием команды set detach-on-fork. Пример сессии GDB при отладке многопроцессного приложения приведен ниже (см. рисунок **9**).

```
$ gdb -silent ./fork
Reading symbols from /home/user/fork...done.
(gdb) set detach-on-fork off
(gdb) r
Starting program: /home/user/fork
[New process 23260]
parent finishes
Program exited with code 020.
(gdb) info inferiors
  Num  Description       Executable
   2   process 23260     /home/user/fork
 * 1   <null>            /home/user/fork
(gdb) inferior 2
[Switching to inferior 2 [process 23260] (/home/user/fork)]
[Switching to thread 2 (process 23260)] (gdb) c

Continuing.
child finishes
Program exited with code 017.
(gdb)
```

Рисунок **9** – Отладка многопроцессного приложения в «неотключаемом» режиме

(программа создает родительский и дочерний процессы.

Оба процесса выводят информацию о своем завершении).

Отладка приложений, использующих нити для реализации параллельного выполнения алгоритма производится аналогичным образом. Переключаться между нитями можно с помощью команды thread. Выдать список имеющихся нитей (находящихся в режиме отладки) можно с помощью команды info threads.

7) Остановка и возобновление программ

Программа запущенная для отладки по умолчанию будет выполняться до тех пор, пока не вызовет исключение или самостоятельно не завершиться. Чтобы остановить программу в нужном месте используется команда break. Эта команда создает *точку останова*. В качестве параметра команды передается одно из следующих значений: абсолютный номер строки в исходном коде программы (формат номера: имя_файла:номер_строки, например file.c:12), относительный номер строки в текущем файле, имя функции (при входе в эту функцию будет осуществлена остановка программы), абсолютный адрес ячейки памяти.

Впроцессе отладки может быть задано несколько точек останова, при достижении любой из которых процесс будет остановлен. Также точка останова может быть снабжена дополнительным условием, которое должно удовлетворяться для того, чтобы программа была остановлена в этой точке. Получить список установленных точек останова можно с помощью команды info

breakpoints. Удалить имеющуюся точку останова можно с помощью команды delete breakpoints, которой в качестве параметра указываются через пробел номера точек останова, которые необходимо удалить. Точки останова можно временно отключить. Для этого используется команда disable breakpoints. Включить обратно точку останова можно с помощью команда enable breakpoints.

Пример отладочной сессии, в которой использованы точки останова приведен ниже (см. рисунок **10**).

```
$ gdb -silent ./file
Reading symbols from /home/user/file...done.
(gdb) list
1       #include <stdio.h>
2
3       int main(void){
4         int i = 0, j;
5         for (i = 0; i < 10; i ++){
6           j = i;
7         }
8         printf ("1234\n");
9       }
(gdb) break 6 if i > 6
Breakpoint 1 at 0x4004dc: file file.c, line 6.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004004dc in main at file.c:6
        stop only if i > 6
(gdb) r
Starting program: /home/user/file
Breakpoint 1, main () at file.c:6
6           j = i;
(gdb) disable breakpoints 1
(gdb) c
Continuing.
1234
Program exited with code 05.
(gdb)
```

Рисунок **10** – Отладка приложения с применением точки останова.

Дополнительно к принудительной остановки процесса по достижению заданного места в программе могут задаваться точки останова при изменении значения заданного выражения (команда watch) и возникновения определённых событий (команда catch).

Отладка приложений может требовать нескольких запусков GDB. Чтобы при каждом запуске отладчика заново не настраивать точки останова их можно сохранить в файл (команда save breakpoints) и затем при необходимости загрузить (команда source).

После достижения точки останова выполнение программы может быть продолжено несколькими способами: до тех пор пока не появится следующая точка останова (команда continue), выполнить одну или несколько строк исходного кода программы (команда step) или только в текущей функции программы (команда next). Дополнительно могут задаваться условия пропуска некоторых строк исходного кода программы при отладке (команда skip).

8) Получение информации о ресурсах и их изменение в процессе отладки программ

и процессе отладки программы необходимо контролировать состояние вычислительных ресурсов. Условно можно выделить два типа таких ресурсов: стек данных, стек вызовов функций оперативная память. Кроме этого в процессе отладки при наличии отладочной информации можно вывести исходный код анализируемой программы.

Стек данных используется для хранения информации в процессе работы программы. В таком стеке, например, сохраняется состояние процесса в случае его остановки или при переключении на другой процесс. В ходе выполнения программы может создаваться несколько стеков данных (фреймов). Получить информацию о стеках данных можно с помощью команды info frames. Переключаться между стеками данных можно с помощью команды frame.

Стек вызовов функций хранит информацию о последовательности входов в функции в процессе исполнения программы. Получить значение этого стека можно с помощью команды backtrace.

Выдать информацию об исходном коде исследуемой программы можно с использованием команды list. По умолчанию выдается 10 строк кода, начинай с текущей исполняемой строки. Выдать содержимое оперативной памяти можно с применением команды examine.

При достижении точки останова пользователь может вывести информацию о любой ячейке оперативной памяти заданной по адресу или по имени. Кроме этого возможно вывести значение некоторого выражения. Вывести эту информацию можно с помощью команды print. Чтобы такая информация выводилась автоматически при достижении любой точки останова используется команда display.

Важно отметить, что вывод команд print, display, examine может быть отформатирован определённым образом. Требования к формату вывода указывается сразу после команды через символ «слеш» (/). В качестве требований может быть задан формат чисел (x – в шестнадцатеричной системе счисления, o – в восьмеричной системе счисления, t – в бинарном виде, d – как целое число со знаком, u – целое число без знака и т.д.), а также количество выдаваемой информации

(например, 10 ячеек памяти, или 5 элементов массива). Пример вывода информации в разном формате приведен ниже (см.).

```
$ gdb -silent ./file
Reading symbols from /home/user/file...done.
(gdb) display/t i
(gdb) display i
(gdb) display
2: i = 7
1: /
t
i
=
11
1
(g
db
)
```

Рисунок **11** – Пример вывода информации о вычислительных ресурсах.

3. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Подготовьте файл с исходным кодом программы, который представлен ниже:

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

int fill_array (char *
  str, char ch){ int i;
  for (i = 0; i < 21; i++)
  str[i] = ch; return (0);
}

int main (int argc, char ** argv){
  char str1[10], str2[10];

  fill_array (str2, 'A'); fill_array (str1, 'B');
  printf ("Массив символов [%s], len = %d\n", str1,
  strlen(str1)); printf ("Массив символов [%s], len =
  %d\n", str2, strlen(str2));

  return (0);
}
```

2. Используя компилятор GCC скомпилируйте подготовленную программу с включением

отладочной информации (всех уровней). На сколько изменяется размер исполняемого файла в зависимости от уровня детализации отладочной информации?

3. Выделите отладочную информацию в отдельный файл с включением ссылки на него

в исполняемый файл. Проведите отладку программы в GDB.

4. Выделите отладочную информацию в отдельный файл без включения ссылки на него

в исполняемый файл. Проведите отладку программы в GDB.

5. Используя отладчик GDB, объясните результат выполнения программы. Почему программа выдает в обоих случаях длину строки, отличную от длины массива? Почему длины строк различные? Что произойдет если в функции fill_array условие окончания цикла изменить на i<25?

6. Используя программу, которую Вы разрабатывали в рамках лабораторной работы «Многонитиевые программы» продемонстрируйте процесс отладки подобных программ с применением GDB.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое отладка программного обеспечения?
2. В каком режиме работает отладчик GDB?
3. Какие параметры командной строки использует GDB?
4. Каким образом должны быть подготовлены программы, чтобы их можно было исследовать с помощью GDB?
5. Как влияет уровень детализации отладочной информации на размер исполняемого файла?
6. Можно ли выделить отладочную информацию в отдельный файл? Если да, то каким образом?
7. Что такое «точка останова»? Сколько точек останова можно задать в процессе отладки программы? Можно ли задать условие срабатывания точки останова?
8. Какую информацию можно получить в процессе отладки программы?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бычков М.И. Основы программирования на VBA для Microsoft Excel:[Текст] учебное пособие, 2010 - 99с.

2. Кукушкина Е.В. Начальные сведения о языке программирования Visual Basic for Application: [Текст] учебная литература для ВУЗов, 2014 – 111с.

3. Абрамян М. Э. Практикум по параллельному программированию с использованием электронного задачника Programming Taskbook for MPI: [Текст] учебное пособие, 2010 – 172с.

4. Паппас, К. Х. Отладка в C++ [Текст] : руководство для разработчиков / Пер. с англ.; Под ред. В. В. Тимофеева. - М. : БИНОМ, 2001. - 512 с.

5. Этапы создания программы на языке Ассемблер. Изучение интегрированной среды отладчика Turbo Debug [Электронный ресурс] : методические указания к лабораторной работе №1 / Юго-Зап. гос. ун-т ; сост.: Е. А. Титенко, Д. И. Родионов, Е. А. Петрик. - Курск : ЮЗГУ, 2010. - 14 с.