Отчет по лабораторной работе № 9 по курсу
Фундаментальная информатика

Студент группы М8О-104Б-22 Ляпин Иван Алексеевич, № по списку 00

Контакты www, e-mail, icq, skype shad0w2020@mail.ru

Работа выполнена: « 29 » октября 2022 г.

Преподаватель: асп. каф. 806 Потенко М.А.

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 202 __ г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Отладчик системы программирования ОС UNIX

2. **Цель работы:** Освоить азы отладчика lldb (аналог GNU Debugger для macOS)

3. **Задание** (*вариант № *): Научиться применять основные команды отладчика в заранее написанной программе.

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____с ОП _____Мб,
НМД _____Мб. Терминал _____адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Apple M1 Pro с ОП 16384 Мб, НМД 524 288 Мб. Монитор Liquid Retina XDR
Другие устройства _____

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства _____, наименование _____версия _____
интерпретатор команд _____версия _____
Система программирования _____версия _____
Редактор текстов _____версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства macOS, наименование macOS Monterey версия 12.3
интерпретатор команд zsh версия 2.12.5
Система программирования C версия _____
Редактор текстов nano версия _____
Утилиты операционной системы Терминал _____
Прикладные системы и программы Xcode
Местонахождение и имена файлов программ и данных /Users/ivan/Desktop _____

6. **Идея, метод, алгоритм** решение задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

Перед работой с отладчиком необходимо скомпилировать файл программы, который мы писали в лабораторной работе №9.
После вызова отладчика в терминале, я воспользуюсь командой "help", отвечающая за все справочные материалы.
Затем выведу код программы через команду "list", таким образом, можно проанализировать код программы.
Также можно создать breakpoint и провести с ним несколько операций, таких как: создание, удаление, включение, выключение, задать положение(номер строки), выполнение программы вопреки break(continue)
Командой "im loo -t ИМЯ ПЕРЕМЕННОЙ" можно узнать её тип.
Чтобы запустить программу, следует воспользоваться командой "run"(также можно сокращать некоторые команды, такие как "run" - > "r" для удобства)
В отладчике можно задать любые произвольные аргументы с помощью команды "setting set target.run-args".
Аналогично можно и просмотреть аргументы.
Также можно посмотреть значения аргументов переменных, что проинициализировались до breakpoint‘a.
Еще следует воспользоваться возможностью менять значения переменных в процессе выполнения программы и выполнять различные действия над ними.
А также благодаря отладчику можно будет пошагово выполнять команды в коде.
С помощью команды "quit" можно завершить работу отладчика.

7. **Сценарий выполнения работы** (план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты либо соображения по тестированию)

1)Перед началом работы скомпилируем файл 9-ой лабораторной работы с помощию команды "gcc", причем необходимо указать ключ "-g"(gcc -g 9labb.c)
2)Вызовем отладчик командой "lldb a.out"
3)Напишем команду "help" для пользования справочником.
4)С помощью команды "list 9labb.c" выведем содержимое файла.
5)Создадим breakpoint в функции main командой "break main".
6)Запустим программу командой "run".
7)С помощью команды breakpoint и префикосов к ней
выполним несколько действий:
"list"-список break‘ов.
"disable"-выключатель break‘a.
"enable"-включить break‘a.
"delete"-удаление break‘a.
8)Используя команду "frame variable" узнаем значения переменных до указанной нами остановки.
9)Командой "expr i" изменив значение переменной i и выведем его командой "print".
10)Возобновим программу командой "continue".
11)Командами "setting set target.run-args", "setting show target.run-args" зададим и выведем аргументы программы.
12)Далее, командой "next" будем несколько раз выполнять по новой строчке программы.
13)Командой "bt"("backtrace") выведем стек вызовов.
14)Прописав "quit" в консоль, завершим работу с отладчиком lldb.

Пункты 1-7 отчета составляются строго до начала лабораторной работы.

Допущен к выполнению работы. **Подпись преподавателя** _____

8. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)

```
ivan@MacBook-Pro-Ivan Desktop % gcc -g 91abb.c
ivan@MacBook-Pro-Ivan Desktop % lldb a.out
(lldb) target create "a.out"
Current executable set to '/Users/ivan/Desktop/a.out' (arm64).
(lldb) help
Debugger commands:
  apropos           -- List debugger commands related to a word or subject.
  breakpoint        -- Commands for operating on breakpoints (see 'help b' for shorthand.)
  command           -- Commands for managing custom LLDB commands.
  disassemble       -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and stack frame.
  expression        -- Evaluate an expression on the current thread.  Displays any returned value with LLDB's default formatting.
  frame             -- Commands for selecting and examing the current thread's stack frames.
  gdb-remote        -- Connect to a process via remote GDB server.  If no host is specified, localhost is assumed.
  gui               -- Switch into the curses based GUI mode.
  help              -- Show a list of all debugger commands, or give details about a specific command.
  kdp-remote        -- Connect to a process via remote KDP server.  If no UDP port is specified, port 41139 is assumed.
  language          -- Commands specific to a source language.
  log               -- Commands controlling LLDB internal logging.
  memory            -- Commands for operating on memory in the current target process.
  platform          -- Commands for manage and create platforms.
  plugin            -- Commands for managing LLDB plugins.
  process           -- Commands for interacting with processes on the current platform.
  quit              -- Quit the LLDB debugger.
  register          -- Commands to access registers for the current thread and stack frame.
  reproducer        -- Commands for manipulating reproducers. Reproducers make it possible to capture full debug sessions with all its dependencies. The resulting reproducer is used to replay the debug session
                       while debugging the debugger.
                       Because reproducers need the whole the debug session from beginning to end, you need to launch the debugger in capture or replay mode, commonly though the command line driver.
                       Reproducers are unrelated record-replay debugging, as you cannot interact with the debugger during replay.
  script            -- Invoke the script interpreter with provided code and display any results.  Start the interactive interpreter if no code is supplied.
  session           -- Commands controlling LLDB session.
  settings          -- Commands for managing LLDB settings.
  source            -- Commands for examining source code described by debug information for the current target process.
  statistics        -- Print statistics about a debugging session
  swift-healthcheck -- Show the LLDB debugger health check diagnostics.
  target            -- Commands for operating on debugger targets.
  thread            -- Commands for operating on one or more threads in the current process.
  trace             -- Commands for loading and using processor trace information.
  type              -- Commands for operating on the type system.
  version           -- Show the LLDB debugger version.
  watchpoint        -- Commands for operating on watchpoints.
Current command abbreviations (type 'help command alias' for more info):
  add-dsym  -- Add a debug symbol file to one of the target's current modules by specifying a path to a debug symbols file or by using the options to specify a module.
  attach    -- Attach to process by ID or name.
  b         -- Set a breakpoint using one of several shorthand formats.
  bt        -- Show the current thread's call stack.  Any numeric argument displays at most that many frames.  The argument 'all' displays all threads.
  c         -- Continue execution of all threads in the current process.
  call      -- Evaluate an expression on the current thread.  Displays any returned value with LLDB's default formatting.
  continue  -- Continue execution of all threads in the current process.
  detach    -- Detach from the current target process.
  di        -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and stack frame.
  dis       -- Disassemble specified instructions in the current target.  Defaults to the current function for the current thread and stack frame.
  display   -- Evaluate an expression at every stop (see 'help target stop-hook'.)
  down      -- Select a newer stack frame.  Defaults to moving one frame, a numeric argument can specify an arbitrary number.
  env       -- Shorthand for viewing and setting environment variables.
  exit      -- Quit the LLDB debugger.
  f         -- Select the current stack frame by index from within the current thread (see 'thread backtrace'.)
  file      -- Create a target using the argument as the main executable.
  finish    -- Finish executing the current stack frame and stop after returning.  Defaults to current thread unless specified.
  history   -- Dump the history of commands in this session.
               Commands in the history list can be run again using "!<INDEX>".  "!!<OFFSET>" will re-run the command that is <OFFSET> commands from the end of the list (counting the current command).
  image     -- Commands for accessing information for one or more target modules.
  j         -- Set the program counter to a new address.
  jump      -- Set the program counter to a new address.
  kill      -- Terminate the current target process.
  l         -- List relevant source code using one of several shorthand formats.
  list      -- List relevant source code using one of several shorthand formats.
  n         -- Source level single step, stepping over calls.  Defaults to current thread unless specified.
               aggregate variables can be specified such as 'var->child.a'.  The -> and [] operators in 'frame variable' do not invoke operator overloads if they exist, but directly access the specified element.
               If you want to trigger operator overloads use the expression command to print the variable instead.
               It is worth noting that except for overloaded operators, when printing local variables 'expr local_var' and 'frame var local_var' produce the same results.  However, 'frame variable' is more
               efficient, since it uses debug information and memory reads directly, rather than parsing and evaluating an expression, which may even involve JITing and running code in the target program.
  vo        -- Show variables for the current stack frame. Defaults to all arguments and local variables in scope. Names of argument, local, file static and file global variables can be specified. Children of
               aggregate variables can be specified such as 'var->child.x'.  The -> and [] operators in 'frame variable' do not invoke operator overloads if they exist, but directly access the specified element.
               If you want to trigger operator overloads use the expression command to print the variable instead.
               It is worth noting that except for overloaded operators, when printing local variables 'expr local_var' and 'frame var local_var' produce the same results.  However, 'frame variable' is more
               efficient, since it uses debug information and memory reads directly, rather than parsing and evaluating an expression, which may even involve JITing and running code in the target program.
  x         -- Read from the memory of the current target process.
For more information on any command, type 'help <command-name>'.
(lldb) list 91abb.c
   1   /*
   2           Лапин Иван Алексеевич, ИВО-104б-22, вариант №4.
   3
   4   */
   5   #include <stdio.h>
   6   #include <math.h>
   7
   8   #define I 26   //  i - начальная координата
   9   #define J 0    //  j - начальная координата
   10  #define L -3   //  l - начальное значение
   11  #define R1 25   //  Квадрат радиуса малой окружности
(lldb) b main
Breakpoint 1: where = a.out`main + 32 at 91abb.c:31:9, address = 0x0000000100003c68
(lldb) run
Process 794 launched: '/Users/ivan/Desktop/a.out' (arm64)
Process 794 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100003c68 a.out`main(argc=1, argv=0x000000016fdff998) at 91abb.c:31:9
   28   int main (int argc, const char * argv[])
   29   {
   30       /* Объявляем переменные, принимающие результат и присваиваем им результат */
-> 31       int i_1 = I, j_1 = J, l_1 = L, k = 0, i, j, l;
   32       /* Входим в цикл, где будет происходить перебор k от 0 до 50 на проверку попадания*/
   33
   34       for (k = 0; k < NUMBER_STEPS; k++) {
Target 0: (a.out) stopped.
(lldb) breakpoint list
Current breakpoints:
1: name = 'main', locations = 1, resolved = 1, hit count = 1
  1.1: where = a.out`main + 32 at 91abb.c:31:9, address = 0x0000000100003c68, resolved, hit count = 1

(lldb) breakpoint disable 1
1 breakpoints disabled.
(lldb) run
There is a running process, kill it and restart?: [Y/n] Y
Process 794 exited with status = 9 (0x00000009)
Process 799 launched: '/Users/ivan/Desktop/a.out' (arm64)
i = 0, j = 198, l = 0 .
Промах!Координаты: (0,198). Динамический параметр движения: 0. Шаг №50.
Process 799 exited with status = 0 (0x00000000)
(lldb) breakpoint enable 1
1 breakpoints enabled.
(lldb) r
Process 803 launched: '/Users/ivan/Desktop/a.out' (arm64)
Process 803 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100003c68 a.out`main(argc=1, argv=0x000000016fdff998) at 91abb.c:31:9
   28   int main (int argc, const char * argv[])
   29   {
   30       /* Объявляем переменные, принимающие результат и присваиваем им результат */
-> 31       int i_1 = I, j_1 = J, l_1 = L, k = 0, i, j, l;
   32       /* Входим в цикл, где будет происходить перебор k от 0 до 50 на проверку попадания*/
   33
   34       for (k = 0; k < NUMBER_STEPS; k++) {
Target 0: (a.out) stopped.
(lldb) frame variable
(int) argc = 1
(const char **) argv = 0x000000016fdff998
(int) i_1 = 1070867281
(int) j_1 = 93044
(int) l_1 = 1
(int) k = 1876449024
(int) i = 1
(int) j = 786528
(int) l = 1
(lldb) expr i = 10
(int) $0 = 10
(lldb) print i
(int) $1 = 10
(lldb) continue
Process 803 resuming
i = 0, j = 198, l = 0 .
Промах!Координаты: (0,198). Динамический параметр движения: 0. Шаг №50.
Process 803 exited with status = 0 (0x00000000)
(lldb) breakpoint delete 1
1 breakpoints deleted; 0 breakpoint locations disabled.
(lldb) settings set target.run-args 1 2 3
(lldb) settings show target.run-args
target.run-args (arguments) =
  [0]: "1"
  [1]: "2"
  [2]: "3"
(lldb) r
Process 846 launched: '/Users/ivan/Desktop/a.out' (arm64)
i = 0, j = 198, l = 0 .
Промах!Координаты: (0,198). Динамический параметр движения: 0. Шаг №50.
Process 846 exited with status = 0 (0x00000000)
(lldb) b main
Breakpoint 2: where = a.out`main + 32 at 91abb.c:31:9, address = 0x0000000100003c68
(lldb) im loo -t i
(lldb) r
Process 858 launched: '/Users/ivan/Desktop/a.out' (arm64)
Process 858 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 2.1
    frame #0: 0x0000000100003c68 a.out`main(argc=4, argv=0x000000016fdff978) at 91abb.c:31:9
   28   int main (int argc, const char * argv[])
   29   {
   30       /* Объявляем переменные, принимающие результат и присваиваем им результат */
-> 31       int i_1 = I, j_1 = J, l_1 = L, k = 0, i, j, l;
   32       /* Входим в цикл, где будет происходить перебор k от 0 до 50 на проверку попадания*/
   33
   34       for (k = 0; k < NUMBER_STEPS; k++) {
Target 0: (a.out) stopped.
(lldb) im loo -t i
(lldb) print i
(int) $2 = 1
(lldb) next
Process 858 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003c80 a.out`main(argc=4, argv=0x000000016fdff978) at 91abb.c:34:10
   31       int i_1 = I, j_1 = J, l_1 = L, k = 0, i, j, l;
   32       /* Входим в цикл, где будет происходить перебор k от 0 до 50 на проверку попадания*/
   33
-> 34       for (k = 0; k < NUMBER_STEPS; k++) {
   35           /* Пусть i, j - конечные значения исходных i_1, j_1, после операций над ними */
   36
   37           i = i_1;
Target 0: (a.out) stopped.
(lldb) next
Process 858 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003c9c a.out`main(argc=4, argv=0x000000016fdff978) at 91abb.c:37:13
   34       for (k = 0; k < NUMBER_STEPS; k++) {
   35           /* Пусть i, j - конечные значения исходных i_1, j_1, после операций над ними */
   36
-> 37           i = i_1;
   38
   39           j = j_1;
   40
Target 0: (a.out) stopped.
(lldb) next
Process 858 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003ca0 a.out`main(argc=4, argv=0x000000016fdff978) at 91abb.c:39:13
   36
   37           i = i_1;
   38
-> 39           j = j_1;
   40
   41           l = l_1;
   42
Target 0: (a.out) stopped.
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
  * frame #0: 0x0000000100003ca0 a.out`main(argc=4, argv=0x000000016fdff978) at 91abb.c:39:13
    frame #1: 0x0000000180011b80 dyld`start + 516
(lldb) quit
Quitting LLDB will kill one or more processes. Do you really want to proceed: [Y/n] Y
ivan@MacBook-Pro-Ivan Desktop %
```

9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. **Замечания автора** по существу работы: Программа, что использовалась в данной лабораторной работе была написана заранее в утилите "Xcode", и соответствующий алгоритм был подробно расписан в отчете к лабораторной работе № 9.

11. **Выводы:** Благодаря данной лабораторной работе я научился анализировать программы с помощью отладчика lldb. С помошью него, можно довольно быстро найти ошибку в коде и затем исправить. lldb - отладчик, являющийся аналогом популярного debugger'а gdb, его особенностью является относительная новизна, факт того, что он написан на языке c++, а также то, что он поддерживает работу с процессорами на ARM ахитектуре в отличии от gdb.

Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента