Основы отладки с GDB

Сборка программы для поддержки отладки (параметр -g)

```
c++ -Wall -g prog.c++ -o prog
```

Запуск интерактивной среды отладчика

```
gdb prog
```

Основные команды отладчика GDB

- **next [n]** – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;
- **step [s]** – пошаговое выполнение программы;
- **continue [c]** – продолжает выполнение программы;
- **finish** – выполняет программу до выхода из текущей функции; отображает возвращаемое значение,если такое имеется;
- **until [u]** - выйти из текущего цикла или продолжить выполнение до строки «x» при указании номера строки;
- **break [b]** – устанавливает точку останова, которая может быть задана номером строки текущего файла, названием функции или выражением [имя-файла.ext]:[номер-строки];
- **watch** *expression* – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения expression изменится;
- **catch** — останавливает выполнение программы на определённых системных или языковых событиях (возникновение исключений, вызов системных функций, загрузка/выгрузка библиотек и т.д.).
- **delete [d]** – удаляет точку останова или контрольное выражение;
- **clear** – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
- **backtrace [bt]** – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;
- **display** – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
- **info breakpoints** – выводит список всех имеющихся точек останова;
- **info watchpoints** – выводит список всех имеющихся контрольных выражений;
- **kill** - принудительное завершение отлаживаемого процесса;
- **list** – выводит исходный код; [имя-файла.ext]:[start-line],[end-line];
- **print** – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
- **run** – запускает программу на выполнение в отладочном режиме; предварительно целесообразно задать точки останова;
- **set** – устанавливает новое значение переменной;
- **tui enable|disable** - открывает или закрывает текстовый пользовательский интерфейс с текстом программы;
- **tui refresh [refresh]** - обновляет вывод текстового пользовательского интерфейса.

Пример изучения тонкостей Си с помощью gdb (см. ссылку)

Настройка gdb-принтера для Qt

Переменные в Qt, будучи сложными объектами, при выводе на экран не дают человеко читаемой информации, требуется специальная настройка вывода, которая реализована в проекте pretty-printers.

Настройка:

- создать каталог ~/.gdb/pretty-printers
- скачать в каталог ~/.gdb/pretty-printers файлы qt.py и helper.py
- создать в домашнем каталоге пользователя файл ~/.gdbinit следующего содержания:

```
python
import sys, os
sys.path.insert(0, os.path.expanduser('~/.gdb/') + 'pretty-printers/')
from qt import register_qt_printers
register_qt_printers (None)
end
```

- запустить gdb, настройки должны автоматически подхватиться из ~/.gdbinit
  - При возникновении ошибки «Scripting in the «Python» language is not supported in this copy of GDB» необходимо убедиться, что установлена не минимальная версия gdb.
  - Также для тестирования файл .gdbinit можно загрузить в каталог проекта и после запуска gdb подгружать настройки командой source .gdbinit

Источники: Setup GDB with Qt pretty printers , qt5printers .

Recording && Replaying

Одним из абсолютно необходимых компаньонов gdb является RR , это отладчик 'post mortem', но, в отличие от основного файла, вы можете воспроизвести действие в обратном порядке, если это необходимо. Вы можете двигаться в обратном направлении, шагать в обратном направлении, создавать точки наблюдения, снова бежать вперед и т. д.

Отладка в VIM'e

.vimrc

```
" Настройка отладчика (загрузка плагина, расположение окон)
autocmd FileType c,cc,cpp,h,hpp,s packadd termdebug
autocmd FileType c,cc,cpp,h,hpp,s cabbrev gdb Termdebug
let g:termdebug_popup = 0
let g:termdebug_wide = 1
```

Запуск отладчика в VIM

```
:gdb
```

или без настройки алисаов команды

```
:Termdebug
```

Установка/удаление точки остановки в vim'e

```
:Break
:Clear
```

Пробелы в пути проекта
Отладка через termdebug не будет корректно работать, если полный путь к фалам проекта содержит пробелы и/или кириллицу. В первом случае, будет явная ошибка, во втором не будут отображаться точки останова в тексте программы

```
-break-insert: Garbage following <location>
```

Дополнительная информация по ссылке.

Видео материалы по GDB

- Плейлист Отладчик GDB на YouTube канале C++ Practice (11 видео).
- Кирилл Владимирович Кринкин «Отладка в Linux. Краткие сведения о gdb».