Машинно-ориентированный текстовый интерфейс отладчика GDB

В.А. Галатенко[1], К.А. Костюхин[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, galat@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kost@niisi.ras.ru

Аннотация. Интерактивный отладчик GDB является основным средством отладки программ, разрабатываемых на инструментальной платформе Linux. Предоставляя широкие возможности для отладки, GDB, тем не менее, не способен удовлетворить всех потребностей разработчиков, например, многопроцессорных систем. Одним из решений является рассматриваемый в этой статье текстовый интерфейс GDB/MI, посредством которого сторонние разработчики могут расширять функциональность отладчика.

Ключевые слова: отладка, многопроцессорные системы, текстовый интерфейс, графический интерфейс, GDB, MI.

1. Введение

Разработчики многопроцессорных систем нередко сталкиваются с проблемой интерактивной отладки как системы в целом, так и отдельных ее компонентов. Учитывая, что отладчик GDB [1], де-факто, является стандартным отладчиком для unix-подобных систем, то выглядит вполне естественным попытаться использовать именно его.

Авторы видят две возможности развития этой идеи: предоставление пользователю графического интерфейса, использующего запущенные экземпляры GDB по одному, как минимум, на каждый узел многопроцессорной системы, и модификация самого GDB для поддержки многопроцессорной отладки. Второй способ является более трудоемким и требует поддержки со стороны сообщества Free Software Foundation, поэтому в данной работе мы сосредоточимся на первом способе. Ключевым моментом здесь является машинно-ориентированный текстовый интерфейс GDB/MI [2]. Фраза «машинно-ориентированный» говорит о том, что этот интерфейс создан для обеспечения коммуникации между отладчиком GDB и другими программами, например, так называемыми графическими надстройками (GUI frontends).

Идеи о создании графических надстроек для GDB, который в своем арсенале имеет только интерфейс командной строки (Command Line Interface, CLI), витали в воздухе уже давно. До того, как появился MI, программы типа DDD (Data Display Debugger [3]), предоставляющие графический отладочный интерфейс разработчикам, использовали стандартные ввод и вывод процесса GDB для того, чтобы, имитируя пользователя, подавать CLI-команды ему на вход и перехватывать с последующей обработкой результат выполнения команд на выходе. У этого подхода есть один существенный недостаток. При изменении, даже незначительном, формата вывода результата выполнения команды графическая надстройка оказывается не в состоянии правильно интерпретировать результат.

Поэтому, начиная с версии 5, в GDB появился дополнительный интерфейс GDB/MI (далее просто MI), основным преимуществом которого является его независимость от формата отображения результата выполнения команды GDB. GDB/MI (расшифровывается как GDB Machine Interface) представляет собой набор структурированных строк, из которых состоят как запросы к GDB от внешних программ, так и их результаты выполнения.

2. Общее описание MI

Перед началом сеанса отладки необходимо запустить GDB с аргументом

--interpreter=mi

Означающим, что все данные, поступающие на стандартный ввод, GDB будет интерпретировать, как команды MI. Опционально можно указать номер версии MI (mi1, mi2 или mi3). По умолчанию используется последняя версия интерфейса.

Взаимодействие внешней программы, использующей MI, с отладчиком GDB состоит из трех основных частей:

- команды, отправленные в GDB (входные команды);
- результаты выполнения этих команд;
- оповещения от GDB о произошедших в отлаживаемой программе или в самом отладчике событиях.

Информационные команды, которые не влияют на статус выполнения отлаживаемой программы, получают в качестве результата всю требуемую информацию. Команды, которые запускают отлаживаемую программу на выполнение, получают только результат: успешно запущена или нет. Оповещения могут показывать изменившийся статус отлаживаемой программы: остановлена, завершена, произошла смена текущего потока управления.

Полный перечень всех команд MI приведен в [2]. В этой работе мы ограничимся общим описанием синтаксиса команд, а также базовых команд, наиболее важных для организации процесса отладки.

3. Синтаксис команд MI

3.1. Входные команды

В общем случае синтаксис входных команд MI выглядит следующим образом.

Команда -> CLI-команда | MI-команда
CLI-команда -> [*токен*] *<команда GDB>*
MI-команда ->[*токен*]-*команда* [-*параметр1*] [*значение1*] … [-*параметрN*] [*значениеN*] [*аргумент1*] … [*аргументN*] [-- *опция*]

Где

токен – [0..9]*, опциональная последовательность десятичных цифр (если токен указан, то результат выполнения команды также будет его содержать).
<команда GDB> - обычная команда отладчика GDB.
команда – одна из команд MI, описанных ниже, а также в [2].
параметр1…параметрN – опциональный набор параметров команды.
значение1…значениеN – опциональный набор значений параметров.
аргумент1…аргументN – опциональный набор аргументов команды.
опция – дополнительная опция команды.

3.2. Ответ отладчика

Ответ отладчика состоит из нескольких записей приведенного в этом разделе формата. Последовательность записей всегда завершается строкой *(gdb)*.

Ответ -> информационная запись | результат (gdb)

Где

результат -> [*токен*]^*статус*[,*данные*]
информационная запись -> асинхронный ответ | дополнительная информация
статус -> "done"| "running"| "connected"| "error"| "exit"
данные -> переменная=значение
переменная – имя внутренней переменной, содержащей информацию по запросу.
значение – набор сгруппированных данных.
асинхронный ответ - *<результат выполнения отлаживаемой программы до останова или завершения> | =<дополнительное сообщение отладчика> | +<статус целевой программы>.
дополнительная информация - ~<вывод консоли отладчика> | @<вывод целевой программы> | &<вывод журнала>.

3.3. Пример

Пример ниже иллюстрирует приведенное описание.

```
$ gdb -q --interpreter=mi ./example
=thread-group-added,id="i1"
~"Reading symbols from ./example..."
~"done.\n"
(gdb)
-break-insert main
^done,bkpt={number="1",type="breakpoint",
disp="keep",enabled="y",addr="0x08048457",
func="main",file="example.c", fullname="/opt/ex/ex-
ample.c",line="10", thread-
groups=["i1"],times="0",original-location="main"}
(gdb)
-exec-run
=thread-group-started,id="i1",pid="2956"
=thread-created,id="1",group-id="i1"
^running
*running,thread-id="all"
(gdb)
=breakpoint-modified,bkpt={number="1",
type="breakpoint",disp="keep",enabled="y",
addr="0x08048457",func="main", file="exam-
ple.c",fullname="/opt/ex/example.c", line="10",thread-
groups=["i1"],times="1",original-location="main"}
~"\n"
~"Breakpoint 1, main () at example.c:10\n"
~"10\t\tint i, k=9;\n"
*stopped,reason="breakpoint-
hit",disp="keep",bkptno="1",
frame={addr="0x08048457",func="main",args=[],file
="example.c",fullname="/opt/ex/example.c",
line="10"},thread-id="1",stopped-threads="all",core="0"
(gdb)
-exec-continue
^running
*running,thread-id="all"
(gdb)
~"[Inferior 1 (process 2956) exited with code 012]\n"
=thread-exited,id="1",group-id="i1"
=thread-group-exited,id="i1",exit-code="012"
*stopped,reason="exited",exit-code="012"
(gdb)
-gdb-exit
^exit
```

4. Основные команды MI

4.1. Работа с точками прерывания

-break-after – аналог GDB-команды *ignore*.

Игнорировать точку прерывания указанное количество ее срабатываний.

-break-commands – аналог GDB-команды *commands*. Задает список команд GDB, которые будут выполнены при срабатывании указанной точки прерывания.

-break-condition – аналог GDB-команды *condition*. Задает условия срабатывания точки прерывания.

-break-delete – аналог GDB-команды *delete*. Удаляет одну или несколько точек прерывания.

-break-disable – аналог GDB-команды *disable*. Временно отключает точку прерывания.

-break-enable – аналог GDB-команды *enable*. Включает ранее отключенную точку прерывания.

-break-insert – аналог GDB-команды *break, tbreak*. Устанавливает точку прерывания.

-break-list – аналог GDB-команды *info break*. Выдает информацию по установленным точкам прерывания.

-break-watch – аналог GDB-команды *watch*. Устанавливает точку прерывания по данным.

4.2. Работа с потоками управления

-thread-info – аналог GDB-команды *info thread*. Выдает информацию по отлаживаемым потокам управления.

-thread-select – аналог GDB-команды *thread*. Устанавливает заданный поток управления в качестве текущего (по умолчанию к нему будут применяться все последующие команды отладки).

4.3. Управление ходом выполнения программы

-exec-continue – аналог GDB-команды *continue*. Запускает отлаживаемый поток управления.

-exec-finish – аналог GDB-команды *finish*. Запускает отлаживаемый поток управления до выхода из текущего кадра стека.

-exec-interrupt – аналог GDB-команды *interrupt*. Останавливает выполнение программы.

-exec-jump – аналог GDB-команды *jump*. Запускает отлаживаемый поток управления с указанного адреса.

-exec-next – аналог GDB-команды *next*. Запускает отлаживаемый поток управления до перехода на другую строку исходного текста без остановки в вызываемых в процессе выполнения функциях.

-exec-next-instruction – аналог GDB-команды *nexti*. Запускает отлаживаемый поток управления до перехода на другую машинную команду без остановки в вызываемых в процессе выполнения функциях.

-exec-return – аналог GDB-команды *return*. Производит немедленный возврат из текущего кадра стека.

-exec-run – аналог GDB-команды *run*. Создает новый поток управления с указанной точкой входа.

-exec-step – аналог GDB-команды *step*. Запускает отлаживаемый поток управления до перехода на другую строку исходного текста с остановкой в вызываемых в процессе выполнения функциях.

-exec-step-instruction – аналог GDB-команды *stepi*. Запускает отлаживаемый поток управления до перехода на другую ассемблерную команду с остановкой в вызываемых в процессе выполнения функциях. Выполняет ровно одну машинную команду.

-exec-until – аналог GDB-команды *until*. Запускает отлаживаемый поток управления до достижения заданного адреса.

4.4. Работа со стеком

-stack-info-frame – аналог GDB-команды *info frame*. Выдает информацию о заданном кадре стека.

-stack-list-frames – аналог GDB-команды *backtrace*. Выдает информацию о стеке вызовов текущего потока управления.

-stack-list-locals – аналог GDB-команды *info locals*. Выдает информацию о локальных переменных текущего кадра стека.

-stack-select-frame – аналог GDB-команды *frame*. Делает заданный кадр стека текущим.

4.5. Работа с данными

-data-disassemble – аналог GDB-команды *disassemble*. Выдает дизассемблированный фрагмент памяти целевого потока.

-data-evaluate-expression – аналог GDB-команды *print* или *call*. Вычисляет выражение и выдает его результат.

-data-list-changed-registers – полезная команда для графических надстроек. Выдает список регистров, изменившихся с момента последнего останова отлаживаемого потока.

-data-list-register-names – выдает список имен доступных для отладочных действий регистров целевого процессора.

-data-list-register-values – аналог GDB-команды *info registers*. Выдает список целевых регистров с их значениями.

-data-read-memory – аналог GDB-команды *x*. Выдает содержимое целевой памяти в указанном формате и диапазоне адресов.

-data-read-memory-bytes – пытается прочесть и вернуть содержимое всех доступных ячеек памяти в указанном диапазоне адресов. Если содержимое памяти по каким-то адресам прочесть не удается, то соответствующие адреса помечаются как нечитаемые.

-data-write-memory-bytes – записывает массив

байтов в целевую память по указанному диапазону адресов.

4.6. Работа с таблицами имен

-symbol-info-functions – аналог GDB-команды *info functions*. Выводит список всех глобальных функций, удовлетворяющих заданному шаблону.

-symbol-info-types – аналог GDB-команды *info types*. Выводит список всех определенных в модуле типов, удовлетворяющих заданному шаблону.

-symbol-info-variables – аналог GDB-команды *info variables*. Выводит список всех глобальных переменных, удовлетворяющих заданному шаблону.

-symbol-list-lines – Выводит список всех строк текста программы из заданного в качестве аргумента исходного файла, содержащих исполняемый код.

4.7. Выбор целевой системы

-target-select – аналог GDB-команды *target*. Устанавливает для отладчика указанную целевую систему.

-target-disconnect – аналог GDB-команды *disconnect*. Завершает сеанс отладки текущей целевой системы.

4.8. Команды поддержки MI

-info-gdb-mi-command – возвращает true или false в зависимости от того, поддерживает ли указанную команду отладчик, или нет.

-list-features – возвращает список поддерживаемых отладчиком настроек и команд MI протокола.

-list-target-features – возвращает список расширенных свойств целевой системы, которые может использовать отладчик, например, поддерживает ли целевая система асинхронное выполнение отлаживаемых потоков, или обратное выполнение (reverse execution).

4.9. Вспомогательные команды MI

-gdb-exit – аналог GDB-команды *quit*. Производит завершение работы отладчика.

-gdb-set – аналог GDB-команды *set*. Устанавливает значение переменной отладчика или меняет его настройки.

-gdb-show – аналог GDB-команды *show*. Отображает значение переменной отладчика или его настроек.

-gdb-version – аналог GDB-команды *show version*. Отображает версию используемого отладчика.

-enable-timings – переключает режим отображения времени выполнения команды отладчика. Включенный режим позволяет разработчикам оптимизировать свой код, например, смотреть, как долго выполнялся фрагмент программы между двумя точками останова.

5. Недостатки интерфейса MI

Разработчикам, планирующим внедрять интерфейс MI в свои отладочные комплексы, следует иметь ввиду, что MI является именно интерфейсом, а не протоколом отладки. В частности, можно выделить следующие его недостатки.

Во-первых, отсутствует строгая спецификация MI. В официальной документации [2] есть ряд неточностей, с которыми сталкивались авторы. В частности, в описании команды *-exec-step-instruction* нет упоминания о том, что в качестве аргумента можно задавать число машинных команд, которые должны быть выполнены (по умолчанию выполняется только одна машинная команда).

Во-вторых, если вывод отлаживаемой программы направлен в тот же поток, что и вывод используемого отладчика GDB, то появляется возможность у самой отлаживаемой программы «запутать» процесс отладки. Например, отправив в какой-то момент на свой вывод строку «^exit». Тогда используемая надстройка сделает вывод, что GDB завершил выполнение, и прервет сеанс отладки. Избежать этого можно, как разведя выводы отлаживаемой программы и GDB, так и применяя для каждой команды MI случайно сгенерированные числовые токены, описанные в п. 3.1.

В-третьих, пользователь или разработчик надстройки над GDB должен самостоятельно проверять и устанавливать набор символов (character set), используемых в GDB и/или в отлаживаемой программе, чтобы иметь возможность правильно обрабатывать вывод GDB.

6. Заключение

В работе был рассмотрен машинно-ориентированный текстовый интерфейс отладчика GDB. Несмотря на указанные недостатки, авторы рекомендуют использовать его при разработке собственных или адаптации существующих надстроек для коммуникации с GDB. Кроме того, средства контролируемого выполнения также могут использовать этот интерфейс для полуавтоматического взаимодействия с интерактивным отладчиком, своевременно выявляя ошибки, отказы или атаки, и обеспечивая, тем самым, выполнение целевыми комплексами своих задач.

В дальнейшем планируется адаптация интерфейса для графической надстройки отладчика реального времени, входящего в состав комплекса контролируемого выполнения.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Machine Oriented Text Interface to GDB

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The GDB interactive debugger is the main debugging tool for programs developed on the Linux platform. While providing ample opportunities for debugging, GDB, however, is not able to meet all the needs of, for example, multiprocessor systems developers. One of the solutions is the GDB/MI text interface discussed in this article, with which third-party developers can extend the functionality of the debugger.

Keywords: debugging, multiprocessor systems, CLI, GUI, GDB, MI

Литература

1. GDB: The GNU Project Debugger, https://www.sourceware.org/gdb/.
2. The GDB/MI Interface, https://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html#GDB_002fMI.
3. DDD: The Data Display Debugger, https://www.gnu.org/software/ddd/.