GDB-СЕРВЕР

Версия 1.0
28.04.2018

ОГЛАВЛЕНИЕ

1. ПРИМЕР ОТЛАДОЧНОЙ СЕССИИ С ИСПОЛЬЗОВАНИЕМ GDB-СЕРВЕРА

В данном разделе приводится пример отладки тривильной программы под плату NVCom-02T с использованием gdb-сервера.

Предполагается, что в переменной окружения PATH есть инструменты для архитектуры mips. Корневая директория, в которую распакован архив с gdb и gdb-сервером, далее будет называться mdb_tools.

1.1 Исходный код примера

Создаем ассемблерный исходный код:

```
$ cat > start.s
start:
     nop
     nop
     nop
     nop
```

Собираем:

```
mipsel-elf32-gcc -e 0x80001000 -Wl,-Ttext=0x80001000 -nostartfiles start.s -o main
```

1.2 Запуск gdb-сервера

Запускаем gdb-сервер на симуляторе:

```
$ mdb_tools/bin/mgdbserver -s@NVCom-02T.cfg --inferiorio :17489
Listening on port 17489
```

Через опцию **-s** передается имя конфигурационного файла.

Запуск gdb-сервера в общем случае выглядить так:

```
$ mdb_tools/bin/mgdbserver [backend-options] --inferrioio :port_number
```

Опция **–inferriorio** перенаправляет вывод выполняемых команд к gdb, **port_number** - tcp порт, к которому будет присоединяться gdb. Для задания настроек симулятора или эмулятора нужно задать **backend-options**.

В случае отладки на эмуляторе нужно задать имя устройста в виде регулярного выражения при помощи опции **-d**. Регистр имени устройста игнорируется. Из списка устройств выбирается первое, совпавшее с регулярным выражением (совпадение может быть неполным). Пример:

```
$ mdb_tools/bin/mgdbserver -dnvcom --inferriorio :17489
Devices:
  0. NVCom-02T on EZ-USB0
Openning device  0. NVCom-02T on EZ-USB0...
Finished.
Listening on port 17489
```

Также вместо регулярного выражения может быть использован номер устройства:

```
$ mdb_tools/bin/mgdbserver -d0 --inferriorio :17489
```

1.3 Запуск gdb

Создаем следующий скрипт инициализации gdb:

```
$ cat > gdbinit
set architecture mips
set remotetimeout 10
file main
target remote :17489
#NVCom-02TEM-3U
monitor set 0xb82f4004 0xffffffff
monitor set 0xb82f4000 0xb01230
monitor set 0xb82f1000 0x003000fc
monitor set 0xb82f1014 0x030d0030
monitor set 0xb82f1018 0x00f50222
monitor set 0xb82f101c 0x1
load main
```

Для работы с gdb-сервером в gdb нужно использовать удаленную отладочную цель (**target remote**). Т.к. подключение к плате может быть достаточно медленным, то нужно увеличить время ожидания ответа от gdb-сервера через переменную remotetimeout (значение задается в секундах). Загрузка отлаживаемой программой осуществляется через команду **load**. Команда **file** нужна для загрузки символьной информации.

Запускаем gdb:

```
$ mdb_tools/bin/gdb -x gdbinit
The target architecture is assumed to be mips
0xbfc00000 in ?? ()
Loading section .text, size 0x10 lma 0x80001000
Start address 0x80001000, load size 16
Transfer rate: 128 bits in <1 sec, 16 bytes/write.
(gdb)
```

1.4 Сессия отладки

1.4.1 Особенности target remote

При подключении к gdb-серверу gdb считает, что мы уже отлаживаем программу.

Поэтому можно обращаться к регистрам ядер через штатные команды gdb без предварительного выполнения команды **run**:

```
(gdb) p/x $pc
$1 = 0x80001000
(gdb)
```

Более того, команда **run** отсутствует в контексте удаленной отладочной цели. Для запуска на выполнение нужно использовать команду **continue**:

```
(gdb) b *start + 4
Breakpoint 1 at 0x80001004: file start.s, line 3.
(gdb) c
Continuing.
Breakpoint 1, start () at start.s:3
3               nop
(gdb)
```

Команды, обычно выполняемые в блоке hook-run, можно описать в блоке target hookpost-remote. Пример:

```
define target hookpost-remote
  monitor set 0xb82f4004 0xffffffff
  monitor set 0xb82f4000 0xb01230
  monitor set 0xb82f1000 0x003000fc
  monitor set 0xb82f1014 0x030d0030
  monitor set 0xb82f1018 0x00f50222
  monitor set 0xb82f101c 0x1
  load main
end
```

Вышеприведенный код исполниться автоматически после исполнения команды **target remote**.

1.4.2 Работа с периферийными устройствами

Работа с периферийными устройствам осуществляется при помощи команды **monitor**: 1) monitor info devices - выводит список устройств; 2) monitor info device [device_name] - выводит список регистров и их значений для устройства с имененем **device_name**; 3) monitor info device [device_name] [register_name] - выводит значение регистра **register_name**, находящегося в устройстве **device_name**; 4) monitor info device [device_name] [register_name] [value] - присваивает значение **value** регистру **register_name** устройства **device_name**.

Выводим список периферийных устройств:

```
(gdb) monitor info devices
dsp_common
ether
ether_dma
i2c
memch_dma0
memch_dma1
memory
mfbsp
system
timers
uart
usbic
usbic_dma
vpin
vpin_dma
vpout
```

Выводим значения регистров устройства system:

```
(gdb) monitor info device system
clken                0xffffffff
clksel               0x301230
csr                  0x10041
irqm                 0x0
maskr0               0x0
maskr1               0x0
maskr2               0x0
qstr0                0x0
qstr1                0x0
qstr2                0x0
```

Получаем значение регистра clken:

```
(gdb) monitor info device system clken
clken                0xffffffff
```

Устанавливаем значение регистра clken:

```
(gdb) monitor info device system clken 0
(gdb)
```

1.5 Завершение сессии отладки

Gdb-сервер завершает работу при окончании сессии отладки в gdb. Закрыть текущее соединение явным образом можно при помощи команды **disconnect**:

```
(gdb) disconnect
Ending remote debugging.
(gdb)
```

2. ТИПОВЫЕ СЦЕНАРИИ РАБОТЫ С GDB-СЕРВЕРОМ

2.1 Включение/выключение удаленного монитора (mdbmon).

Монитор mdbmon - режим перехвата системных вызовов write и exit. Для включения можно выполнить следующую команду из терминала gdb:

```
$(gdb) monitor enable mdbmon
```

Выключается режим mdbmon через команду **monitor disable**:

```
$(gdb) monitor disable mdbmon
```

По умолчанию этот режим выключен.

2.2 Подключение к удаленному mjtagserver'у

Запускаем mjtagserver на удаленной машине:

```
$ mjtagserver -a hostname:port
```

В этом случае mjtagserver откроет порт **port** на сетевом интерфейсе **hostname** и будет ждать соединения для начала отладочной сессии. В качестве **hostname** можно использовать 0.0.0.0 (any address).

Далее, чтобы mgdbserver подсоединился к удаленному mjtagserver'у на локальной машине, нужно при старте выполнить команду mdb **connect**:

```
$ mgdbserver -dnvcom --bi "connect remote_hostname:port" :17489
```

Аргумент **remote_hostname** - адрес удаленной машины, **port** - порт, переданный mjtagserver'у.

2.3 Передача аргументов трассы симулятору

Команду трассы симулятора можно передать через опцию **–bi**:

```
$ mgdbserver -s@NVCom-02T.cfg --bi "trace trace_command" :17489
```

Аргумент **trace_command** - команда трассы симулятора (см. документацию по трассе симулятора).

2.4 Подключение без ресета к плате

Подключиться к плате без сброса можно выполнив команду mdb **conf set noreset** через опцию **–bi**:

```
$ mgdbserver -d0 --bi "conf set noreset" :17489
```

2.5 Задание множества ядер, используемых при отладке

При старте gdb-сервера есть возможность задать множество ядер, видимых для gdb. Это делается посредством опции **–cores**, которая принимает список номеров ядер, завершающийся пустой опцией **–**:

```
$ mgdbserver -d0 --cores 0 1 -- :17489
```

Нужно иметь ввиду, что в данном случае меняется только представление для gdb - на уровне бекенда отладка будет производится на всех ядрах.

2.6 Задание пути к библиотеке симулятора

При старте gdb-сервера есть возможность задать путь к библиотеке симулятора через опцию **–backend-args**:

```
$ mgdbserver --backend-args sim @NVCom-02T.cfg /home/username/libs/libsim3x.x64.so
↪:17489
```

2.7 Прерывание отлаживаемой программы

В некоторых случаях необходимо остановить отлаживаюмую программу, минуя интерфейс gdb. Например, при отладки из-под IDE, которая не передает сигнал SIGINT отладчику. В этом случае нужно запустить gdb-сервер с опцией **–control-port**:

```
$ mgdbserver -d0 --control-port PORT_NUMBER :17489
```

Gdb-сервер откроет дополнительное соединение на порту **PORT_NUMBER**, после чего отлаживаемую программу можно будет остановить, вызвав скрипт send.py со следующими аргументами:

```
$ python2.7 <install_dir>/data-directory/python/gdb/send.py break PORT_NUMBER
```

Директория <install_dir> - корневая директория установочного архива.

2.8 Режим ожидания следующего соединения

По умолчанию после завершения отладочной сессии gdb-сервер также завершает работу. Опция **–multi** включает режим ожидания следующего соединения. Пример:

```
$ mgdbserver -d0 --multi :17489
```

Подключения и отключения от gdb-сервера в консоли gdb:

```
(gdb) target remote :17489
Remote debugging using :17489
0xbfc00000 in ?? ()
(gdb) disconnect
Ending remote debugging.
(gdb) target remote :17489
Remote debugging using :17489
0xbfc00000 in ?? ()
```

Завершить работу gdb-сервера можно выполнив команду монитора в интерактивной консоли gdb:

```
(gdb) monitor exit
```

2.9 Отладка на эмуляторе с использованием одного ядра

Запускаем gdb-сервер, указав нужное ядер через опцию **–cores**:

```
$ mgdbserver -d0 --cores CORE_NUMBER -- :22222
```

Далее после подключения gdb к gdb-серверу выполняем слудующие команды:

```
(gdb) monitor core CORE_NAME
(gdb) monitor iocore CORE_NUMBER
```

Где CORE_NAME - имя ядра в описании mdb, CORE_NUMBER - номер нужного ядра. После выполнения вышепреведенных настроек чтение/запись памяти, регистров, запуск на выполнение будут затрагивать только выбранное ядро.

2.10 Запуск gdb-сервера в скрипте инициализации gdb

Для запуска gdb-сервера нужно добавить следующие строки в скрипт инициализации gdb:

```
python
from gdb.mgdbserver import start_server
start_server('-s@NVCom-02T.cfg', '--inferriorio', ':22222')
end

target remote :22222
```

Функция *start_server* принимает все обычные опции gdb-сервера. Каждая опция должна передаваться отдельным аргументом в виде строки. Вышеприведенный код запуска gdb-сервера эквивалентен следующему:

```
mgdbserver -s@NVCom-02T.cfg --inferiorio :22222
```

3. ОПИСАНИЕ ОПЦИЙ И КОМАНД GDB-СЕРВЕРА

3.1 Опции командной строки

Формат командной строки следующий: mgdbserver [OPTIONS] ADDRESS FILENAME [ARGS ...].

3.1.1 Адрес

Параметр ADDRESS задается в виде HOST:PORT. HOST можно опустить. Пример:

```
mgdbserver -d0 0.0.0.0:17489
```

3.1.2 Отлаживаемый файл

В качестве FILENAME передается файл отлаживаемой программы. Этот параметр также можно опустить. В этом случае отлаживаемый файл можно будет загрузить позже, используя команду отладчика GDB load. Пример:

```
mgdbserver -d0 0.0.0.0:17489 foo.elf
```

3.1.3 Аргументы отлаживаемой программы

Аргументы, следующие после файла, будут переданы в функцию main отлаживаемой программы. В качестве нулевого аргумента будет передан путь к отлаживаемому файлу. Пример:

```
mgdbserver -d0 0.0.0.0:17489 foo.elf -f foo bar
```

3.1.4 Опции

3.1.4.1 Выбор бекенда

Для отладки нужно выбрать бекенд (симулятор или эмулятор). Для этого можно передать следующие опции:

- –sim-config|-s config-name, где config-name - путь к конфигурационному файлу симулятора или строка вида @ИмяКонфига.cfg;
- –device-number|-d device, где device - или номер устройства(нумерация начинается с нуля), или регулярное выражение, описывающее имя устройства. Эта опция выбирает в качестве бекенда эмулятор;

- –backend-args backend-name [args ...], где backend-name - имя бекенда (sim - для симулятора, em - для эмулятора), args - аргументы специфичные для бекэнда (см. раздел *Формат конфигурационного файла*)

3.1.4.2 Опции логирования

Для управления логированием есть следующие опции: - –log-file FILENAME - перенаправление вывода gdb-сервера в файл по пути FILENAME; - –log-level|l log-level- выбор уровня логирования, log-level - одно из значений Info, Error, Debug. По умолчанию вывод gdbserver'а оправляется в STDOUT, уровень логирования - Info. Логирование иерархично, т.е. если выбран уровень логирования Info, то вывод соотвестующий уровням Error и Debug выводиться не будет.

3.1.4.3 Передача команд бекенду

Бекенду можно передавать текстовые команды (команды трассы для симулятора, команды mdb для эмулятора). Для этого есть следующие команды: - –bi command - передача инициализирующей команды; - –bc command - передача обычной команды. Подробнее см. в разделах *Формат конфигурационного файла* и *Команды gdb-сервера*. Примеры:

```
$ mgdbserver -d0 --bc initddr
$ mgdbserver -s@dsponly-solar-dsp-60eva.cfg --bi "trace -Ueva"
```

3.1.4.4 Конфигурационный файл

При помощи опции -f можно передать gdb-серверу конфигурационный файл. Например: mgdbserver -f mgdbserver.cfg. О формате конфигурационного файла см. в разделе *Формат конфигурационного файла*. Конфигурационный файл можно распечатать через команду –print-config. При передачи этой команды gdb-сервер завершает работу сразу после обработки опций.

3.1.4.5 Другие опции

- –multi - при использовании этой опции gdb-сервер по завершения отладочной сессии ждет нового подключения со стороны gdb;
- –inferrior - перенаправляет вывод команд удаленного монитора (см. *monitor*) на консоль gdb;
- –control-port port_number - открывает порт port_number для приема контрольных команд (см. *interrupt_server*);
- –cores core_number0 core_number1 ... - определяет множество ядер, используемых в отладке (см. *Типы компонента target*).

3.2 Формат конфигурационного файла

Формат конфигурационного файла gdb-сервера представляет собой ряд записей следующего вида:

```
component name [args ...]
init command
init command
...
cmd command
cmd command
```

Токен **component** может принимать значения **backend**, **target**, **server**, **logger**. Параметр **name** определяет тип компонента, например **sim** или **em** для компонента **backend**. Аргументы **args** - параметры необходимые для создания компонента, разделенные пробелами. Далее идет необязательный список команд компонента.

Команды разбивается на два типа: обычные и инициализирующие. Инициализирующие команды должны выполняться при создании компонента, иначе они не будут иметь эффекта. Список команд приведен в разделе 3.

Записи **backend**, **target**, **server** должны идти друг за другом. Местоположение записи **logger** произвольно.

3.2.1 Типы компонента backend

На данный момент есть два типа: sim (симулятор) и em (эмулятор).

3.2.1.1 Аргументы бекенда sim

Данный тип компонента принимает следующие аргументы: config_name library?. Аргумент config_name представляет собой путь к конфигурационному файлу или строку вида @Имя-Конфига.cfg. Необязательный аргумент library представляет собой путь к библиотеке симулятора. Если аргумент library не указывать, то библиотека симулятора будет искаться в директории исполняемого файла gdb-сервера. Ожидается, что имя библиотеки эквивалентно SimCore.dll на ОС Windows или libSimCore.so на ОС Linux.

Пример:

```
backend sim @NVCom-02.cfg /home/username/libsimcore.lib64.so
```

3.2.1.2 Аргументы бекенда em

Данный тип компонента принимает один аргумент - номер или имя устройства. Нумерация устройств начинается с нуля, порядок соответствуют порядку вывода устройств командой mdb **listdevs**. Пример:

```
backend em 0
```

Имя устройства задается через регулярное выражение. Регистр регулярного выражения игнорируется. Совпадение необязательно должно быть полным. Пример:

```
# совпадет NVCom01, NVCom02-T
backend em nvcom

# совпадет MCom-02
backend em mcom
```

3.2.2 Типы компонента target

На данный момент поддерживается только один тип таргета - baremetal. Аргументы - номер ядер, которые будет представлены в gdb в виде потоков. В отсутствии аргументов будут использованы все ядра. Пример:

```
# в gdb будут представлены потоки mips, dsp0, dsp1
target baremetal

# в gdb будет только один поток mips
target baremetal 0

# в gdb будут два потока: dsp0, dsp1
target baremetal 1 2
```

3.2.3 Типы компонента server

На данный момент поддерживается только один тип сервера - tcp. В качестве аргумента принимается номер порта. Пример:

```
server tcp 22222
```

3.2.4 Коментарии

Пустые и начинающиеся с символа # строки игнорируются. Пример:

```
# backend
backend sim foo.cfg
# target
target baremetal
# server
server tcp 22222
```

3.2.5 Конфигурационный файл и опции командной строки

Получить конфигурационный файл, равносильный по действию набору опций командной строки, можно добавив опцию –print-config. Пример:

```
$ mgdbserver -d 0 :22222 file.elf --print-config > gbserver.cfg
```

Конфигурационный файл можно дополнять опциями командной строки, указынными после опции с конфигурационным файлом. Например, можно изменить порт, сохранив все остальные настройки:

```
$ mgdbserver -f config.cfg :2346
```

3.3 Команды gdb-сервера

3.3.1 Команды эмулятора (backend em)

В качестве команд эмулятора можно использовать все валидные команды mdb. Ряд команд использовать нежелательно, иначе поведение gdb-сервера неопределено. Список таких команд: - opendev - step - run - closedev

Определенные команды будут иметь эффект только в случае использования их, как инициализирующих. Список таких команд: - conf set noreset - connect

3.3.2 Команды симулятора

3.3.2.1 Команда передачи параметров трассы симулятору

Синтаксис: trace trace_command. Может использовать только как инициализирующая. Пример:

```
$ mgdbserver -sNVCom-02T.cfg -bi "trace some_command" :22222
```

3.3.2.2 Установка значения регистра

Синтаксис: set register_name value Команда . Имя регистра задается также, как и в интерпретаторе FreeShell. Пример:

```
monitor set dsp0.dcsr 0x4000
```

3.3.2.3 Запись слова в память

Синтаксис: set memory_address value. Пример:

```
monitor set 0x20000000 10
```

Это команда может использоваться только после создания компонента.

3.3.2.4 Пауза после установки симулятора

Синтаксис: timeout value. Инициализирующая команда. Позволяет задать паузу в миллисекундах после останова симулятора.

3.3.2.5 Сброс модели симулятора

Синтаксис: reset. Сбрасывает модель симулятора. Должна использоваться после создания компонента.

3.3.2.6 Печать структуры GlobalInfo

Ситнаксис: ginfo. Распечатывает структуру GlobalInfo. Должна использоваться после создания компонента.

3.3.2.7 Загрузка исполняемого файла

Синтаксис: loadelf filename. Загружает elf-файл в память. Должна использоваться после создания компонента.

3.3.3 Команды видоизменения описания устройства

На уровне компонента backend предоставляются команды для удаления и переименовывания устройств, групп, регистров. Эти команды должны выполняться до создания компонента.

3.3.3.1 dr

Синтаксис: dr arch_expr device_name_expr new_device_name. Переимование устройства. Пример:

```
# переименование устройства с архитектурой mips и именем risc в risc0
dr mips risc risc0
```

3.3.3.2 dd

Синтаксис: dd arch_expr device_name_expr. Удаление устройства. Пример:

```
# удаление устройства с архитектурой misp и именем risc1
dd mips risc1
```

3.3.3.3 gr

Синтаксис: gr arch_expr device_name_expr new_group_name. Переименование имени группы регистров. Пример:

```
# у групп, принадлежащие к устройствам архитектуры elcore, убирается префикс dspN
gr elcore .* dsp\d_([a-zA-Z0-9_]+) $1
```

3.3.3.4 gd

Синтаксис: gd arch_expr device_name_expr new_group_name. Удаление группы регистров. Пример:

```
# удаление группы rf из описания устройства dsp0
gd elcore dsp0 rf
```

3.3.3.5 mr

Синтаксис: mr arch_expr device_name_expr name_expr new_name. Переименование региона памяти. Пример:

```
# регион памяти устройства dsp0 переименовывается из xyram0 в xyram
mr elcore dsp0 xyram_0 xyram
```

3.3.3.6 md

Синтаксис: md arch_expr device_name_expr name_expr. Удаление региона памяти. Пример:

```
# удаление всех регионов памяти
md .* .* .*
```

3.3.3.7 rr

Синтаксис: rr arch_expr device_name_exp group_expr reg_name new_name. Аргументы device_name_exp, arch_exp, group_exp, reg_name - регулярные выражения, применяемые к имени устройства, имени архитектуры, имени группы и имени регистра соответственно. При совпадении всех этих параметров имя регистра меняется на new_name. Аргумент new_name может содержать ссылки на группы в регулярном выражении reg_name. Пример:

```
#переименование регистра zero в r0
rr mips .* cpu zero r0
#переименование регистров f01,f02,...,f09 в f1,...,f9
rr mips .* fpu f0(\d) f$1
```

3.3.3.8 rd

Синтаксис: rd arch_expr device_expr group_expr name_expr. Удаление регистра. Пример:

```
# удаление регистра vpc из всех устройств с архитектурой elcore
rd elcore .* pcu vpc
```

3.3.4 Команды таргета baremetal

На данный момент таргет baremetal не имеет инициализирующих команд. Описание других команд представлен ниже.

3.3.4.1 symbol

Синтаксис: symbol name. Аргумент name - имя символа. Ряд выполненных команд symbol формирует список символов, значение которых будут запрошено у gdb при начале отладочной сессии.

3.3.5 Команды сервера tcp

3.3.5.1 interrupt_server

Синтаксис: interrupt_server port_number. Команда позволяет прерывать выполнение gdb-сервера и отлаживаемой программы через отсылку текстовых сообщений на порт port_number. Пример:

```
interrupt_server 22223
```

Пример завершения работы gdb-сервера через отсылку сообщения при помощи netcat:

```
nc 127.0.0.1 22223
sigterm
```

Пример прерывания отлаживаемой программы:

```
nc 127.0.0.1 22223
sigint
```

Пример получения статуса gdb-сервера:

```
nc 127.0.0.1 22223
status
accepting
```

Значения, возвращаемые gdb-сервером при получении сообщения status: - accepting - gdb-сервер ожидает соединения; - debugging - соединение между gdb и gdb-сервером было установлено.

3.3.5.2 enable persistent_mode

После завершения текущей отладочной сессии gdb-сервер будет ждать следующего соединения. Аналогично по действию опции –mulit.

3.4 Полезные команды gdb

3.4.1 target remote

Синтаксис: target remote host:port. Создание удаленного таргета. Пример:

```
target remote 10.0.2.2:22222
```

3.4.2 load

Синтаксис: load filename Команда удаленной загрузки. Команды file, exec-file не загружают программу в память. Поэтому при использовании таргета remote нужно либо использовать эту команду, либо передавать исполняемый файл gdb-серверу.

3.4.3 set remotetimeout

Синтаксис: set remotetimeout value, где value - значение в секундах. Установка временного промежутка ожидания ответа со стороны сервера. В случае отладки на эмуляторе необходимо выставлять эту величину не меньше 10 секунд. Аналогично по действию командной опции gdb -l. Пример:

```
set remotetimeout 10
```

Или:

```
gdb -l 10 -x remote_gdbinit
```

3.4.4 monitor

Синтаксис: monitor command Выполнение команды на сервере. Аргумент command должен быть валидной командой сервера. Для того, чтобы вывод команд отображался в консоли gdb, нужно запускать gdb-сервер с опцией **–inferiorio**.

Ниже представлены ряд доступных команд gdb-сервера.

3.4.4.1 info devices

Синтаксис: monitor info devices. Печатает список периферийных устройств.

3.4.4.2 info device name

Синтаксис: monitor info device device_name. Печатает список имен и значений регистров устройства с названием name.

3.4.4.3 info device name reg_name

Синтаксис: monitor info device device_name reg_name. Печатает значение регистра reg_name, принадлежащего к устройству name.

3.4.4.4 info device name reg_name value

Синтаксис: monitor info device name reg_name value. Устанавливает значение регистра reg_name, принадлежащего к устройству name.

3.4.4.5 exit

Синтаксис: monitor exit. Завершает работу gdb-сервера.

3.4.4.6 info args

Синтаксис: monitor info args. Печатает аргументы отлаживаемой программы, переданные gdb-серверу.

3.4.4.7 exit process

Синтаксис: monitor exit-code code. Устанавливает код завершения code отлаживаемой программы. Пример использования:

```
(gdb) monitor exit-code 10
(gdb) c
Continuing.
[Inferior 1 (Remote target) exited with code 012]
(gdb)
```

3.4.5 target extended-remote

Синтаксис: target extended-remote host:port Создание удаленного таргета с расширенным набором команд. В частности поддерживается команда run.

3.4.6 set remote exec-file

Синтаксис: set remote exec-file filename. Позволяет установить исполняемый файл filename при использовании отладочной цели extended-remote. Путь к файлу должен быть валидным на машине с запущенным gdb-сервером.

4. INDICES AND TABLES

- genindex
- modindex
- search