Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б08-мм

Реализация gdb сервера для эмулятора целевых процессоров

Чернобровкина Юлия Владиславовна

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преп. каф. ИАС, К. К. Смирнов

Консультант:
Старший разработчик ПО I категории ООО «Софтком», П. А. Бабанов

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. RSP протокол 5

2.2. Передача пакетов 5

2.3. RSP команды 6

2.4. API эмулятора ЭМ-1 6

3. Реализация сервера **8**

3.1. Установка TCP соединения 9

3.2. Пакетный интерфейс 9

3.3. Обработка запросов GDB 11

3.4. Взаимодейтсвие сервера с API эмулятора 11

Заключение **13**

Список литературы **14**

Введение

Одним из ключевых этапов разработки программного обеспечения является отладка. Этот процесс может занимать значительное время и стоить программисту многих усилий. Современный рынок программных средств предоставляет широкий ассортимент инструментов для отладки – отладчиков.

Среди наиболее популярных можно выделить GNU Debugger (GDB) – переносимый отладчик проекта GNU. По версии сайта techberg.ru [1] GDB является одним из мощнейших отладчиков Linux с широким набором функций. Однако непосредственный запуск отладчика GDB невозможен для ряда систем, таких как встроенные системы, симуляторы и эмуляторы. Тем не менее отладка этих систем все еще нужна. Для решения этой проблемы программисту достаточно реализовать сервер, поддерживающий GDB Remote Serial Protocol (RSP) – протокол высокого уровня, позволяющий отладчику GDB удаленно подключаться к целевой платформе. Любая удаленная целевая платформа, которая реализует протокол RSP, может взаимодействовать с GDB и таким образом позволяет программисту использовать функциональность GDB в полном объеме. Создание GDB сервера – распространенная практика для некоторых видов систем, к примеру свои GDB серверы имеют симуляторы QEMU [4] и gem5 [3].

Одним из направлений работы компании Софтком является эмуляция целевых процессоров. Для повышения качества эмуляторов и удобства работы с ними необходимо реализовать некоторые компоненты окружения эмулятора, в том числе сделать возможной отладку запущенной в эмуляторе программы. Для этого было решено написать GDB сервер, реализующий RSP протокол, чтобы иметь возможность отлаживать код, который исполняется в эмуляторе, с помощью отладчика GDB.

Таким образом, целью данной работы является создание GDB сервера для эмуляторов процессоров компании Софтком.

1. Постановка задачи

Целью работы является создание GDB сервера для эмуляторов целевых процессоров компании Софтком.

Для достижения цели были поставлены следующие задачи:

1. изучить RSP протокол и команды GDB;
2. изучить интерфейс эмулятора;
3. реализовать GDB сервер:
   (a) реализовать соединение между сервером и клиентом на основе TCP соединения;
   (b) разработать структуры и функции для представления пакетов и обмена ими с GDB;
   (c) реализовать обработчик команд и запросов в соответствии с требованиями RSP протокола.

2. Обзор

2.1. RSP протокол

GDB Remote Serial Protocol (RSP) – это высокоуровневый протокол, который позволяет GDB удаленно подключаться к серверу, реализующему этот протокол. Обмен данными между сервером и клиентом может происходить с помощью разных типов соединения: протокол поддерживает TCP/IP и UDP/IP соединения.

При взаимодействии GDB выступает в качестве клиента, а целевая система осуществляет серверную часть. Подробное описание протокола можно найти на сайте [5].

2.2. Передача пакетов

Обмен данными между GDB и сервером происходит через пакеты. GDB передает пакет, в котором содержится запрос на предоставлении информации либо на совершение какого-либо действия. В зависимости от типа запроса сервер может передать свой пакет.

Структура пакета RSP соединения представлена ниже [2].

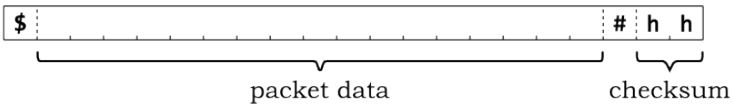

Рис. 1: структура пакета RSP соединения. Рисунок взят из источника https://www.embecosm.com/appnotes/ean4/embecosm-howto-rsp-server-ean4-issue-2.html (дата доступа: 14 декабря 2023 г.).

Пакет начинается с символа «$», далее следуют данные, представленные в виде двух шестнадцатеричных чисел на каждый байт. В конце пакета стоит символ «#» и контрольная сумма, также в виде пары шестнадцатеричных чисел. Контрольная сумма вычисляется как сумма всех символов данных, взятая по модулю 256.

По умолчанию, как только сервер или клиент получил пакет, он должен отправить подтверждение: «+» в случае успешного получения

пакета и «-», если передача пакета прошла с ошибками и требуется выслать пакет повторно. Этот механизм необходим GDB, если сервер использует ненадежные соединения (например UDP/IP). Если же соединение устойчивое/надежное, такое как TCP/IP, механизм подтверждения пакета может быть отключен.

В случае прерывания соединения со стороны GDB (если пользователь нажал ctr-C) на сервер передается символ 0x03. По своему усмотрению сервер может отследить сигнал прерывания и обработать его, либо же не отслеживать его вовсе.

2.3. RSP команды

Команды, которые GDB передает серверу, представляются в виде строк. Каждая команда передается в своем пакете. В зависимости от ожидаемого ответа сервера пакеты можно разбить на следующие группы:

1. не требующие ответа;
2. требующие простого подтверждения. Подтверждением может быть OK, Enn (nn – номер ошибки) или пустой пакет. Это команды: !, A, D, G, H, M, P, Qxxxx, T, vFlashErase, vFlashWrite, X, z, Z;
3. ожидающие пакет с результатом или кодом ошибки. Это команды: ?, c, C, g, m, p, qxxxx, s, S, vxxxx.

Минимальный набор команд, который должен поддерживать GDB сервер, это команды «?» для сообщения причины остановки, «g» и «G» для доступа к регистрам, «m» и «M» для доступа к памяти.

2.4. API эмулятора ЭМ-1

В данной работе обработка запросов GDB активно задействует интерфейс эмулятора, разрабатываемого компанией Софтком, поэтому в этом разделе кратко рассказано о возможностях эмулятора, доступных через его API.

Интерфейс эмулятора позволяет загружать исполняемый файл, запускать исполнение кода или выполнить несколько инструкций. Можно получить доступ к регистрам процессора и оперативной памяти, добавить и удалить точку останова. Ниже представлена часть из технического описания интерфейса эмулятора.

Таблица 1: типы данных эмулятора

Номер	Имя	Описание
1	N	Целое число (8 байт)
2	S	Паскалевская строка, длина 0-255 (1 байт), передается ссылкой (8 байт)
3	A	Массив: длина натуральное число (8 байт) элемент массива – номер типа элемента (8 байт), значение N или ссылка на S (8 байт) ...
100	V	Значение параметра (элемент списка параметров): номер параметра – натуральное число (8 байт) тип параметра – число (8 байт) ссылка на имя параметра – тип 2 (8 байт) значение параметра для типа 1 или ссылка на значение для типов 2 и 3 (8 байт)
101	C	Список параметров конфигурации: число элементов списка – натуральное число или 0 (8 байт) элемент списка – тип 100 (32 байта) ...
200	M	Список блоков оперативной памяти: число блоков (8 байт) блок – начало (8 байт), конец (8 байт), ссылка на S – имя владельца (8 байт) ... Пустое имя владельца означает, что блок свободен.
1000	E	Информация об ошибке: код ошибки (8 байт) ссылка на краткий комментарий типа S (8 байт)

Таблица 2: техническое описание некторых функций эмулятора

Действие	Команда эмулятора	Функция эмулятора	Комментарии
Загрузить библиотеку или исполняемый файл	1 P	7 1 параметр – ссылка на имя файла (тип S). Возвращаемое значение – 0 (выполнено) или -1 (ошибка).	В случае ошибки печатается код ошибки и краткий комментарий (функция -1)
Показать значения всех регистров процессора	r	8 Возвращаемое значение – ссылка на тип A, элементами которого являются значения регистров.	Порядок следования значений регистров является фиксированным для каждой модели процессора.
Установить/удалить точку останова	b P bx P	16 2 параметра P1 – адрес точки останова P2 – 0 (удалить), 1 (установить) Возвращаемое значение – 0 (выполнено) или -1 (ошибка)	В случае ошибки печатаеся код ошибки и краткий комментарий (функция -1).

Для вызова конкретной функции из технического описания используется функция uemu_dsp(uint32_t num, ...) с переменным числом агрументов, где первый агрумент – номер вызываемой функции.

3. Реализация сервера

Диаграмма последовательностей и диаграмма развертывания для GDB сервера представлены ниже.

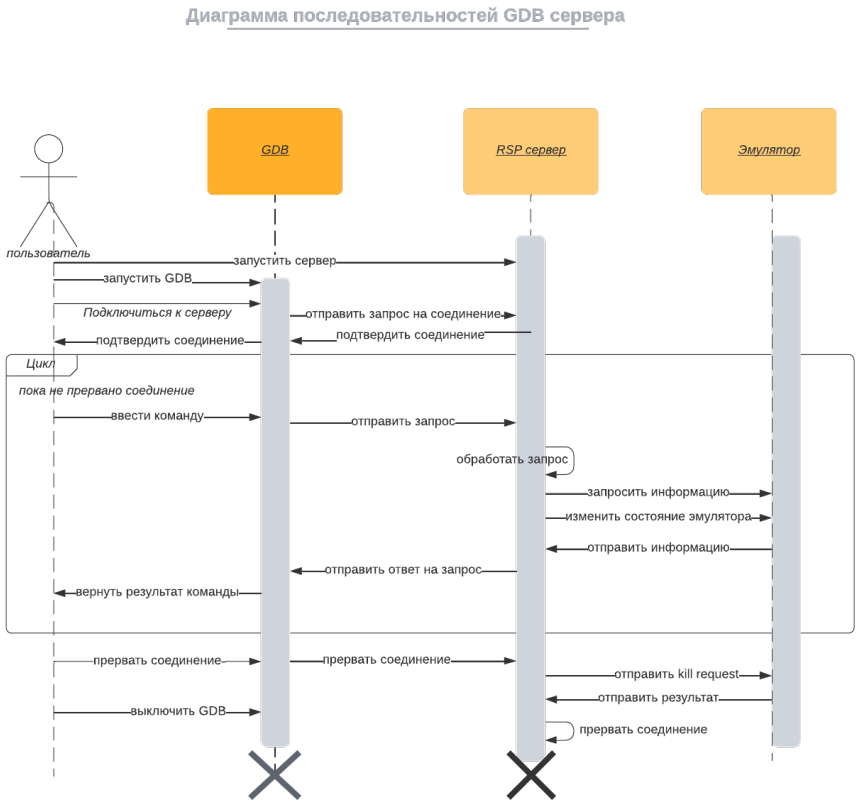

Рис. 2: диаграмма последовательностей GDB сервера

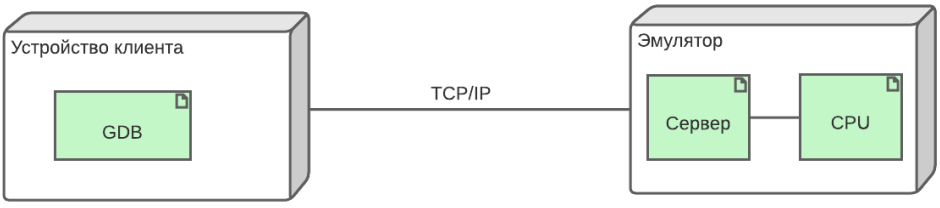

Рис. 3: диаграмма развертывания GDB сервера

3.1. Установка TCP соединения

Первым этапом разработки GDB сервера стала реализация серверной части программы.

В качестве типа соединения было выбрано TCP/IP соединение, так как оно обеспечивает упорядоченную отправку пакетов, проверяет наличие ошибок и гарантирует доставку каждого пакета. По этим показателям среди поддерживаемых RSP соединений протокол TCP/IP более надежный и потому лучше подходит для реализации сервера.

3.1.1. Установление соединения с удаленным хостом

Для создания сервера используются функции из стандратных библиотек для работы с сокетами. Таким образом, чтобы соединиться с сервером, пользователь должен запустить на своей машине GDB и вызвать удаленную отладку, указав IP адрес сервера в качестве параметра. Ниже приведен пример соединения GDB с сервером, запущенным на той же машине на порте 9002:

Листинг 1: пример подключения к серверу со стороны хоста

```
gdb
target remote 127.0.0.1:9002
```

Если сервер находится в состоянии прослушивания входящих соединений, после ввода этих команд в терминале произойдет подключение gdb к серверу.

3.2. Пакетный интерфейс

3.2.1. Представление данных пакета

Пакеты состоят из char, тем не менее трактовать пакет как строку не следует, так как пакет может содержать символ конца строки. Решено было представить пакеты в виде структуры struct Packet с полями data и len.

Листинг 2: структура, представляющая данные пакета

```
struct Packet
{
    char data[MAX_BUF_SIZE];
    size_t len;
    uint8_t CheckSum();
};
```

Данные пакета хранятся в статическом массиве char data[] размера MAX_BUF_SIZE, длина пакета находится в поле int len. Выделять память динамически нет необходимости, так как в работе одновременно могут находиться не более одного пакета, для удаления пакета достаточно записать значение ноль в поле len.

3.2.2. Отправление и получение пакета

Чтение пакета реализовано в методе GetPkt(). Метод считывает символ, отправленный gdb, ожидая символа начала пакета – «$». Все последующие символы функция сохраняет в структуру Packet, до тех пор пока не встретится символ конца пакета – «#». Далее с помощью метода CheckSum() вычисляется контрольная сумма полученного пакета и сверяется с указанной в пакете. В зависимости от правильности контрольной суммы на сторону gdb отправляется подтверждение «+» или «-».

Отправление пакета реализовано в методе PutPkt(). Функция отправляет на сторону gdb содержимое структуры Buf в корректном для пакета виде, то есть с символом «$» в начале и «#» и контрольной суммой в конце пакета. Далее функция считывает подтверждение, отправленное gdb, и в случае отрицательного ответа повторяет отправку пакета.

3.2.3. Вспомогательные функции

CheckSum() – метод, вычисляющий контрольную сумму пакета, а именно сумму всех символов, входящих в структуру Buf, взятую по модулю 256.

GetChar(), PutChar() – методы для получения и отправления одиночного символа соответственно. Они взаимодействуют с сокетам хоста через библиотечные функции read() и write() для получения и отправки символа.

PackStr() – записывает строку в си стиле в структуру Packet. Используется для записи ответа на запросы со стороны gdb.

HexToChar(), CharToHex() – функции для преобразования шестнадцатеричного числа в char и наоборот.

3.3. Обработка запросов GDB

Последовательная обработка запросов gdb реализована в методе ProcessRequests(), который в бесконечном цикле вызывает метод HandleRequest(), в свою очередь обрабатывающий одиночный запрос. Метод HandleRequest() получает от GDB следующий пакет с помощью метода GetPkt() и передает первый символ оператору switch, чтобы определить, какое действие ожидается от сервера. В случае, если переданный символ соответствует поддерживаемому запросу, вызывается нужная функция. Для неопознанных пакетов сервер выведет предупреждение.

Обработка основных запросов осуществляется через интерфейс эмулятора.

3.4. Взаимодейтсвие сервера с API эмулятора

Для доступа к интерфейсу эмулятора к проекту подключена динамическая библиотека libuemu_v8.so. Данная библиотека экспортирует две функции: uemu_init() и uemu_dsp(int num, ...). Ниже представлена их сигнатура.

Листинг 3: сигнатура функций из API эмулятора

```
uint32_t uemu_init(void);
uint32_t uemu_dsp (uint32_t num, ...);
```

Функция uemu_init() инициализирует эмулятор, функция uemu_dsp(int num, ...) дает доступ к основному API из обзора, где num - номер функции в соответствии с документом. Ниже представлен пример вызова функции из документации эмулятора через uemu_dsp(int num, ...), а также инициализация эмулятора.

Листинг 4: инициализация эмулятора и вызов функции из интерфейса

```
uemu_init();
uemu_dsp(7, &elf_image, elf_size, &ed);
```

В методе StartServer() инициализируется эмулятор с помощью uemu_init() и загружается исполняемый файл, который нужно отлаживать.

Заключение

В ходе учебной практики были выполнены следующие задачи:

1. изучен RSP протокол и команды GDB;
2. изучен интерфейс эмулятора;
3. реализован GDB сервер:
   (a) реализовано соединение между сервером и клиентом на основе TCP соединения;
   (b) разработаны структуры и функции для представления пакетов и обмена ими с GDB;
   (c) реализован обработчик команд и запросов в соответствии с требованиями RSP протокола.

Работа велась в репозитории компании, открытая часть кода доступна по ссылке: https://github.com/Chernobrovkina-Yulya/gdb-server

Список литературы

[1] 10 лучших отладчиков Linux для инженеров-программистов. — URL: https://techberg.ru/10-luchshih-otladchikov-linux-dlya-injenerov-programmistov-uprost (дата обращения: 13 декабря 2023 г.).

[2] Bennett Jeremy. Howto: GDB Remote Serial Protocol. — 2008. — URL: https://www.embecosm.com/appnotes/ean4/embecosm-howto-rsp-server-ean4-issue-2.html (дата обращения: 14 декабря 2023 г.).

[3] Bruce Bobby R. — Debugging Simulated Code, 2023. — URL: https://www.gem5.org/documentation/general_docs/debugging_and_testing/debugging/debugging_simulated_code (дата обращения: 13 декабря 2023 г.).

[4] Features/gdbstub. — URL: https://wiki.qemu.org/Features/gdbstub (дата обращения: 13 декабря 2023 г.).

[5] GDB Remote Serial Protocol. — URL: https://sourceware.org/gdb/current/onlinedocs/gdb.html/Remote-Protocol.html#Remote-Protocol (дата обращения: 13 декабря 2023 г.).