Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Трассировка загрузочного модуля в z/OS

Курсовая работа студента 444 группы

Щербакова Александра Владимировича

Научный руководитель **Вояковская Н.Н.**

(Профессор кафедры Системного программирования Математико-Механического факультета СПбГУ)

Санкт-Петербург

2014

Оглавление

Введение

z/OS – операционная система от компании IBM, работающая на платформе мэйнфрейм.

Хотя самой платформе уже очень много лет (если быть точнее, 07.04.14 мейфреймам исполнилось ровно 50 лет)[1], она всё еще жива и развивается. Причиной этому служат следующие факты о мэйнфреймах:

- Надежность, которой славятся мэйнфреймы;
- Заявленная TCO (Total Cost of Ownership – полная стоимость владения, куда входят затраты на электричество, обслуживание, занимаемое пространство и т.п.) является одной из самых небольших для серверного оборудования Enterprise класса;
- Полная (или почти полная) обратная совместимость – затраты на миграцию на новый мэйнфрейм со старого почти не буду включать в себя переработку ПО, так как практически все программы, написанные для ранних версий, будут работать и на более поздних.

Хотя платформа и развивается, специалистов в этой области становится всё меньше. Причиной этому является уменьшающееся количество образовательных программ, нацеленных на обучение работе на мэйнфреймах, взамен которым приходят всё больше новых платформ.

Кроме того, сама платформа, хоть и имеет хорошую документацию и поддержку, всегда оставалась проприетарной, даже в отношении ПО – очень небольшое количество ПО, работающего на этой платформе, выложено в открытый доступ, тем более вместе с исходным кодом.

Этой курсовой хотелось не только углубить свои знания в этой платформе, но и создать полезный программный продукт для неё с открытым исходным кодом.

---

[1] Подробнее - (4)

Постановка задачи

В системе есть загрузочный модуль и окружение для его запуска.

Необходимо проследить работу модуля, а именно, получить ответы на следующие вопросы:

1. Какие инструкции были выполнены, а какие – нет?
2. Как сопоставить данные, полученные в трассе, с исходным кодом?

Дело в том, что подобное программное обеспечение не было найдено, даже в качестве упоминания, в процессе выбора курсовой работы. Кроме того, такое программное обеспечение уже имеет потенциальных пользователей (например, компания EMC, которая предложила данную тему курсовой работы и имеет в своём составе подразделение, занимающееся ПО для мэйнфреймов).

Варианты решения

При написании данной курсовой работы рассматривались следующие методы решения поставленной задачи:

- Пошаговое выполнение команд из загрузочного модуля с помощью ассемблерной команды EX (“execute”);
- Предварительная порча загрузочного модуля и перехват прерываний;
- SLIP – стандартный инструмент для отладки, поставляемый IBM вместе с операционной системой z/OS.

Команда EX

Для этого метода требуется написать «песочницу» для динамической загрузки нужного модуля в память, а затем исполнения его шаг за шагом.

Плюсы:

1. Возможность получить трассу максимальной точности – вплоть до инструкции;
2. Так как инструкции будет отправлять на выполнение не операционная система, а специально написанный модуль, это позволит записывать все смещения от начала исследуемого загрузочного модуля, а также выбрать, какие конкретно данные нужно записать.

Минусы:

1. Значительное время увеличение работы каждой инструкции в исследуемом модуле;
2. Невозможность построить полную трассу: при асинхронном выполнении исследуемого модуля, мы будем получать информацию только по основному потоку;
3. Невозможность правильно обрабатывать прерывания с помощью исследуемого загрузочного модуля.

Очевидно, данное решение хоть и позволяет собрать достаточно подробную трассу, обладает рядом существенных недостатков, из которых два: невозможность собрать полную трассу и обработать все прерывания – не дают возможности рассматривать этот способ как рабочий.

Порча загрузочного модуля и перехват прерываний

Для использования этого метода требуется сначала обработать загрузочный модуль следующим образом:

1. Собрать всю информацию об инструкциях перехода в модуле (и сохранить её для дальнейшего использования);
2. Все инструкции переходов (первые 2 байта) заменить на X’0000’ (2 байта «нулей»). Это заставит систему вырабатывать прерывание по коду операции, так как не существует операции, начинающейся с X’0000’.

После этого, с помощью специально написанной «песочницы» нужно загрузить испорченный модуль в память и начать его исполнение, а прерывания перехватывать в «песочнице», исправляя модуль по мере надобности. (Также можно сразу загрузить модуль в память и уже в памяти его «испортить».)

Плюсы:

1. Не тратится время на обработку простых команд в исследуемом модуле – только на обработку прерываний по коду операции, что происходит только на операциях перехода;
2. Трассу можно получить с требуемым количеством информации;
3. Можно «портить» не весь модуль, а только конкретную его часть, чтобы получать информацию только о ней.

Минусы:

1. Необходимость определять, исправлять или нет код операции в модуле (например, может оказаться, что испорченная операция перехода принадлежит какому-то циклу, и, исправив её, мы можем потерять информацию о работе модуля). Альтернативой может служить отказ от исправления модуля вообще, с исполнением команд перехода в коде «песочницы»;

2. Необходимость вмешиваться в работу исследуемого модуля;
3. Проблема обработчика прерываний: если сам исследуемый модуль определяет свой обработчик прерываний, который следит и за прерываниями по коду операции, этот метод работать не будет.

SLIP

При поиске стандартных методов получения трасс для исполняемых модулей, было найдено средство SLIP, что поставляется компанией IBM вместе с операционной системой z/OS.

При использовании этого метода нужно определить подходящий SLIP trap («ловушку SLIP». Термин SLIP trap является стандартным и будет использоваться далее), запустить модуль на исполнение и получить трассу, которую можно просмотреть с помощью IPCS – Interactive Problem Control Facility – на выходе.

Плюсы:

1. Нет необходимости вмешиваться в работу исследуемого загрузочного модуля;
2. Собираться будет всевозможная информация, в том числе, и работа различных обработчиков прерываний, и асинхронное выполнение модуля;
3. Трассу, получаемую на выходе, можно посмотреть через IPCS – инструмент для просмотра и анализа трасс, снимков памяти и другой отладочной информации, поставляемый компанией IBM вместе с z/OS.

Минусы:

1. Так же, как и в остальных методах, записывается смещение от начала загрузочного модуля, хотя в один загрузочный модуль могут быть слинкованы несколько объектных;
2. В системе одновременно может работать только один SLIP trap нужного типа.

Исходя из плюсов и минусов всех предложенных методов, для реализации был выбран именно этот вариант сбора трассы работы загрузочного модуля. Дело в том, что, хоть у этого варианта есть существенный недостаток – невозможность параллельно запускать несколько копий и работать с

несколькими трассами одновременно, возможности, которыми он обладает намного превосходят его минусы, чего нельзя сказать об остальных рассмотренных вариантах. Используя этот вариант, мы, хоть и будем тратить больше времени, собирая трассы «по одной», всё равно оказываемся «в выигрыше», так как точность метода дает нам ровно те знания, ради которых мы и собираем трассу.

Подробнее о SLIP

SLIP – Serviceability Level Indication Processing – это стандартный инструмент для отладки, который позволяет предпринимать определяемый пользователем ряд действий (в том числе, запись трассы) при выполнении заранее обозначенных условий.

События – и, соответственно, типы SLIP trap, делятся на:

1. Error event traps (ловушки ошибок). Также их называют “non-PER” ловушки;
2. PER traps – ловушки, использующие PER (Program Event Recording) – отладочный инструмент, включающий программное и аппаратное обеспечение, который позволяет перехватывать следующие типы событий (соответственно, для каждого типа перехватываемых событий есть свой тип у SLIP trap):
    a. Instruction Fetching – загрузка инструкции на исполнение из определенного участка виртуальной памяти;
    b. Successful Branch – успешный переход в определенный участок виртуальной памяти;
    c. Storage Alteration – изменение памяти (запись в память). Подразделяют 2 типа Изменения памяти:
        i. Обычное изменение памяти – это изменение данных в определенном участке виртуальной памяти;
        ii. «Прямое» изменение памяти – это изменение на физической памяти, производимое с помощью инструкции STURA (Store Under Real Address).

Instruction Fetching позволяет отслеживать исполнение вплоть до каждой выполненной инструкции, как и первый предложенный вариант решения (использование EX).

Successful Branch позволяет отслеживать все переходы, совершенные в

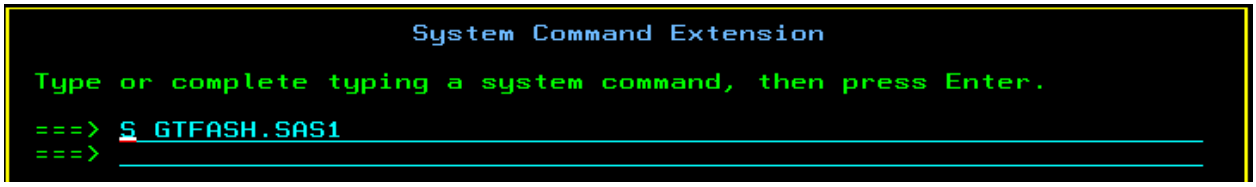
модуле – так же, как и второй предложенный вариант (порча модуля и перехват прерываний). Кроме того, так как не происходит изменения загрузочного модуля, можно отследить и работу различных обработчиков прерываний, определенных в самом модуле.

Так как отслеживать работу модуля вплоть до инструкции – слишком подробно, решено использовать Successful Branch SLIP trap для сбора трассы.

Как это работает?[2]

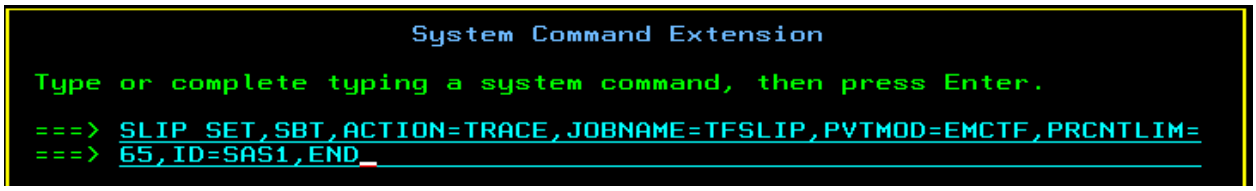
Чтобы получить трассу с помощью SLIP, нужно выполнить следующие действия:

1. Запустить GTF – Generalized Trace Facility (стандартный инструмент для сбора всевозможных трасс в z/OS, поставляемый компанией IBM вместе с операционной системой) – для сбора трассы;

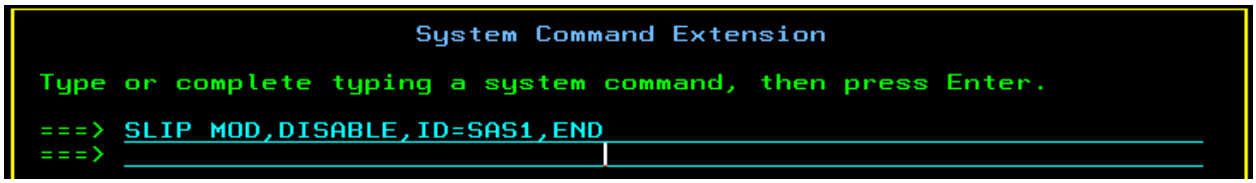


2. Поставить SLIP trap для отслеживания работы модуля:
   - Это делается с помощью команды[3]

$$SLIP\ SET, SBT, ACTION = TRACE, JOBNAME = jobname, PVTMOD = modname, ID = idid, PRCNTLIM = 65, END$$



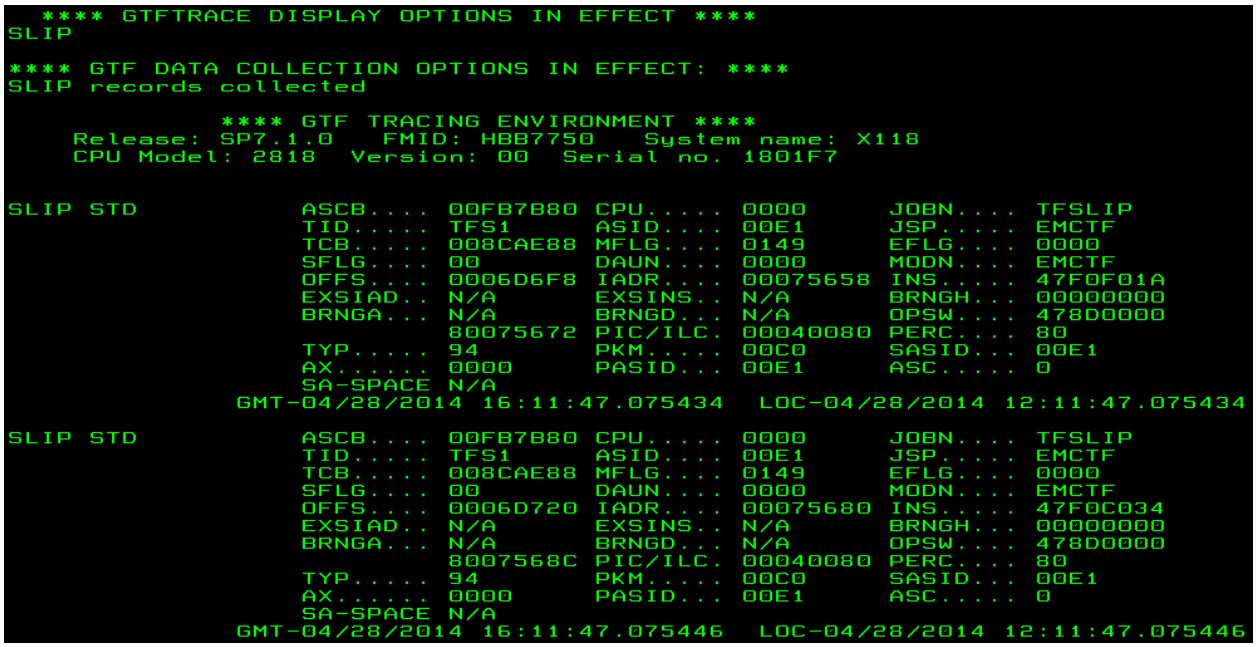
3. Запустить модуль;
4. После остановки работы модуля, остановить SLIP trap и GTF.



[2] Подробнее о стандартных средствах отладки, предоставляемых в z/OS, можно прочитать в (3)
[3] Подробнее о команде SLIP можно прочитать в (2)

В итоге, получаем трассу, которую можно прочесть с помощью IPCS.

Открыв трассу в IPCS, можно будет увидеть нечто подобное:



Как видно из представленного снимка экрана, хоть информация из трассы представлена в читаемом виде, сопоставить с кодом её будет довольно трудно:

1. Поле OFFS содержит лишь смещение относительно начала модуля;
2. Оно содержит данные о месте ***откуда*** произошел переход в интересующий нас модуль:
    a. Это плохо, так как по коду не всегда легко понять, ***куда*** именно будет совершён переход (например, при использовании перехода по регистру и таблицы функций).

Поэтому было решено написать программу, которая могла бы выводить трассу ***в удобочитаемом формате***, отвечать на вопрос, ***куда конкретно был совершен переход***, а также ***ограничивать трассу*** только определенными участками кода (точнее, листинга кода, так как часто в коде присутствуют

макросы, раскрывающиеся в списки инструкций, и рассматривать сам код, а не его листинг, не имеет смысла).

Возможности написанной программы

Программа, реализованная в рамках данной курсовой, обладает следующими возможностями:

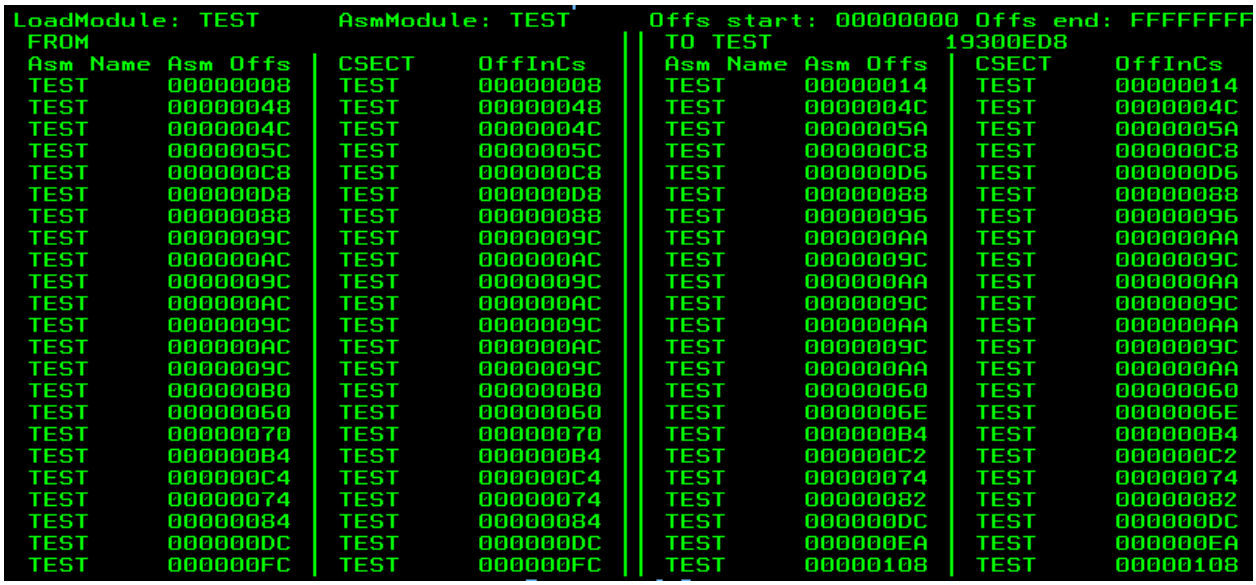
1. Автоматически распознает в трассе служебные записи от информационных;
2. По заданным параметрам выводит интересующую пользователя информацию, а именно:
    a. Имя модуля. Сужает трассу до информации об одном загрузочном модуле (выводиться будут только переходы, совершенные из него);
    b. Имя ассемблерного модуля. С помощью этого параметра трасса сужается и до переходов из/в интересующий ассемблерный модуль;
    c. Смещения в листинге ассемблерного модуля. Этот параметр позволяет отслеживать переходы только в участке, соответствующем только заданным границам. По умолчанию используются границы 00000000-FFFFFFFF;
    d. Таблица линковщика. Это таблица линковщика для данного модуля, содержащая информацию о том, куда был слинкован какой объектный модуль в данном загрузочном модуле.

В итоге получаем более читаемую трассу, в которой видно, в какой конкретно объектный модуль происходил переход и из какого. Выводимые смещения же удобно сопоставлять с листингом кода, просто вбивая их в поиске по файлу.

Далее представлены снимки экранов с примерами результатов работы написанной программы.

LoadModule: EMCTF AsmModule: TFINIT Offs start: 00016CF4 Offs end: 00016D2E

FROM				TO EMCTF	00007F60		
Asm Name	Asm Offs	CSECT	OffInCs	Asm Name	Asm Offs	CSECT	OffInCs
TFINIT	00000D74	TFINIT	00000D74	TFINIT	00016D2E	TFINIT	00016D2E
TFINIT	00000BFE	TFINIT	00000BFE	TFINIT	00016CF4	TFINIT	00016CF4
TFINIT	00016D2C	TFINIT	00016D2C	EMCTF	000047C2	EMCTF	000047C2



LoadModule: TEST AsmModule: TEST Offs start: 00000000 Offs end: FFFFFFFF

FROM				TO TEST	19300ED8		
Asm Name	Asm Offs	CSECT	OffInCs	Asm Name	Asm Offs	CSECT	OffInCs
TEST	00000008	TEST	00000008	TEST	00000014	TEST	00000014
TEST	00000048	TEST	00000048	TEST	0000004C	TEST	0000004C
TEST	0000004C	TEST	0000004C	TEST	0000005A	TEST	0000005A
TEST	0000005C	TEST	0000005C	TEST	000000C8	TEST	000000C8
TEST	000000C8	TEST	000000C8	TEST	000000D6	TEST	000000D6
TEST	000000D8	TEST	000000D8	TEST	00000088	TEST	00000088
TEST	00000088	TEST	00000088	TEST	00000096	TEST	00000096
TEST	0000009C	TEST	0000009C	TEST	000000AA	TEST	000000AA
TEST	000000AC	TEST	000000AC	TEST	0000009C	TEST	0000009C
TEST	0000009C	TEST	0000009C	TEST	000000AA	TEST	000000AA
TEST	000000AC	TEST	000000AC	TEST	0000009C	TEST	0000009C
TEST	0000009C	TEST	0000009C	TEST	000000AA	TEST	000000AA
TEST	000000AC	TEST	000000AC	TEST	0000009C	TEST	0000009C
TEST	0000009C	TEST	0000009C	TEST	000000AA	TEST	000000AA
TEST	000000B0	TEST	000000B0	TEST	00000060	TEST	00000060
TEST	00000060	TEST	00000060	TEST	0000006E	TEST	0000006E
TEST	00000070	TEST	00000070	TEST	000000B4	TEST	000000B4
TEST	000000B4	TEST	000000B4	TEST	000000C2	TEST	000000C2
TEST	000000C4	TEST	000000C4	TEST	00000074	TEST	00000074
TEST	00000074	TEST	00000074	TEST	00000082	TEST	00000082
TEST	00000084	TEST	00000084	TEST	000000DC	TEST	000000DC
TEST	000000DC	TEST	000000DC	TEST	000000EA	TEST	000000EA
TEST	000000FC	TEST	000000FC	TEST	00000108	TEST	00000108

Результаты

В результате данной курсовой работы было проведено исследование способов построения и вывода трасс поведения загрузочных модулей в операционной системе z/OS, написана повторновходимая программа на HLASM (High Level of Assembler – язык ассемблера для мэйнфреймов) для удобного вывода полученных трасс (с возможностью легко сопоставлять данные в трассе с листингом исходного кода) с открытым исходным кодом. Исходный код этой программы, равно как и инструкции для сборки и запуска, можно найти по адресу (1).

Список литературы

1. **SLIP Trace main page. [В Интернете] 2014 г. https://code.google.com/p/emc-spbsu-coop-mainframe/wiki/SLIPTrace.**

2. IBM. *z/OS V1R13.0 MVS System Commands (SA22-7627-27).* 2012.

3. —. *z/OS V1R12.0 MVS Diagnosis Tools and Service Aids.* 2010.

4. —. 50 лет мэйнфреймам. [В Интернете] 07 04 2014 г. [Цитировано: 07 04 2014 г.] habrahabr.ru/company/ibm/blog/218501/.