Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Трассировка многомодульного приложения в среде операционной системы z/OS

Курсовая работа студента 344 группы
Ефремова Ростислава Сергеевича

Научный руководитель **Вояковская Н.Н.**
ЗАО «Ланит-Терком», главный конструктор

Санкт-Петербург
2015

Оглавление

1. Введение

Динамический анализ бинарного кода - методика, позволяющая с помощью информации о ходе выполнении программы на различных данных, делать выводы о том, как она работает. Задача динамического анализа особенно актуальна для операционной системы z/OS. Имеется много старого кода, который сейчас поддерживают, развивают и дополняют новым функционалом. Динамический анализ требует информации о переходах внутри модулей и между модулями. Перехват и трассировка прыжков между модулями является более сложной и общей задачей, которая решена в ряде коммерческих продуктов, неизвестным нам образом, так как их код закрыт. Справимся с этой задачей - с большой вероятностью будем иметь решение и для переходов внутри.

1.1 Постановка задачи

Целью данной курсовой было исследование различных методов трассировки переходов между модулями, сравнение их производительности и реализация тестовой программы для отслеживания простых прыжков типа BC в программах 31-битного режима в primary address space mode для операционной системы z/OS, укомплектованных ADATA и скомпилированных с опцией THREAD. Во время трассировки должна сохраняться реентерабельность исследуемого приложения.

2. Существующие аналоги

Рассмотрим три популярных отладчика, которые в своем составе предлагают некоторые средства неинтерактивной отладки:

- IDF (Interactive Debug Facility)
- z/XDC (Extended Debugging Controller)
- TDF (Trap Diagnostic Facility)

IDF - стандартная утилита IBM для отладки, которая поставляется вместе с HLASM Toolkit. В IDF User's Guide указано, что для отладки она использует либо SVC 97 (то есть SVC hooking) либо ESTAE/ESPIE в зависимости от указанных опций. Из этого сразу следуют сильные естественные ограничения в работе отладчика, которые будут более подробно описаны в разделе 3. В частности, они привели к низкой популярности IDF среди системных программистов.

Кроме IBM, другие компании тоже занимаются разработкой средств трассировки и отладки в среде операционной системы z/OS.

z/XDC - пожалуй самый мощный инструмент для отладки и сбора трассировочной информации. Он позволяет отлаживать приложения, исполняемые в практически любом окружении операционной системы z/OS: SRB, SVC, RTM, а так же использующие разные типы адресных пространств, различные уровни привилегированности. Хотя статей о том как работает z/XDC мало, по Release Guide и анализу памяти приложения в момент работы отладчика (предоставленной коллегами из EMC) можно сделать вывод о том, что точно используются следующие методы перехвата: ESTAE, SVC hooking. Кроме того, в целом неизвестно, какими средствами обеспечивается связь отладчика и исследуемой программы, и в каких случаях замена команд происходит до запуска исследуемой программы. Так как код отладчика

закрыт и он является платным, изучение этих вопросов затруднительно. Далее в курсовой работе рассмотрена самостоятельная реализация SVC hooking для трассировки многопоточного приложения.

TDF - сравнительно новая программа, основанная на технологии TRAP, о чем явно указано в презентации о продукте. Из этого автоматически следуют ограничения в ее работе: нельзя исследовать приложения secondary address space mode и программы, использующие TRAP механизм. Так же интерес представляет то как устанавливается TRAP для исследуемой программы и как эта программа запускается. В курсовой работе далее показано, как теоретически можно обойти эти ограничения, а так же приведено сравнение скорости работы TRAP и SVC hooking технологий.

3. Подходы к перехвату переходов между модулями

В целом к построению графа динамических переходов можно подойти с разных позиций:

1. Создание (или модификация) виртуальной машины
2. Использование стандартных аппаратных средств организации отладки
3. Внесение ошибок и их перехват
4. Замена SVC

Рассмотрим каждый пункт подробнее.

1. Создание виртуальной машины - весьма трудоемкая задача. Однако для Mainframe уже существует открытый бесплатный эмулятор Hercules. Его можно модифицировать для отслеживания определенных событий. Однако хочется иметь программу на целевой платформе, так как перенос некоторых проектов с Mainframe на виртуальную машину может занять много времени и сил
2. Исследование показало, что платформа Mainframe содержит аппаратное средство для реализации отладчиков, и называется оно PER - Program Event Recording. Его использует стандартная часть операционной системы z/OS - SLIP Trace

   Однако использование PER приводит в нашей ситуации к следующим проблемам:

   - Придется обрабатывать все "прыжки" вне зависимости от того: внутренние они или внешние
   - Все события перехода для выделенного куска памяти будут записываться в один dump. Таким образом, например, могут

перемешаться записи о "прыжках" из разных subtask. Их нужно будет разделять

- Если прыжок происходит в область памяти, которая не входит в заданный диапазон адресов (а он может быть только один), то невозможно получить адрес точки назначения перехода. Значит, надо выбирать память таким образом, чтобы она включала в себя инструкцию перехода и адрес точки назначения. Но по сути это и является задачей: определением перехода. В принципе можно попробовать выбрать память так, чтобы в ней лежали все модули интересующего нас приложения, но нет никакой гарантии, что между ними не окажется кода, исполняющегося в другом потоке или чего-нибудь похуже. Возникает идея выделить address space целиком, но это приводит к катастрофическому падению производительности, так как система не справляется с обработкой переходов всех task, subtask, SRB, потому что их очень много

- Невозможность установки своего обработчика PER: ESPIE (сервис для установки обработчиков прерываний) позволяет назначить обработчик только для прерываний с кодами 0-15, а PER имеет код 80. Другими доступными методами установка так же невозможна, так как IH - Interrupt Handler поддерживает область real memory с обработчиком прерываний от z/OS. Это FLIH - First Level Interrupt Handler. Написать свой FLIH не представляется возможным из-за отсутствия доступа к определенной технической документации

Другое стандартное средство организации трассировки - инструкция TRAP.

В системе команд Mainframe есть две команды: TRAP2 и TRAP4. Они

изначально были предназначены для решения проблемы 2000 года, но нашли применение и в организации отладки, так как представляют быстрый и удобный сервис для перехода в некоторую подпрограмму с почти полным восстановлением контекста. Поскольку команда TRAP2 занимает 2 байта, на нее можно заменить любую инструкцию и попасть в TRAP подпрограмму.

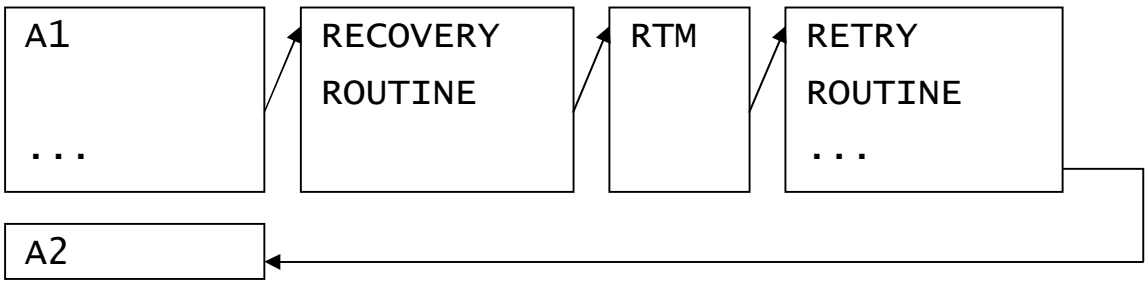
Однако при использовании команды TRAP возникают четыре проблемы:

- Окружение восстанавливается не полностью
- В secondary address space mode команду TRAP вызвать нельзя
- TRAP не может восстановить home space mode, если процессор не находится в привилегированном режиме (supervisor state)
- Установка TRAP окружения для другого потока сопряжена с определенными трудностями: в родительском потоке мы не имеем доступа к структурам данных, где нужно выставить необходимые флаги и адреса
- При анализе многопоточного приложения придется создавать свой экземпляр Trap-Program для каждого subtask

3. Внесение ошибок и их перехват - очень распространенный метод организации отладки и перехвата переходов. Команда перехода заменяется, например, на код X'0000'. Исследование показало, что в z/OS имеется система для обработки прерываний (в том числе по ошибкам) - IH - Interrupt Handler, и система исключительно для обработки ошибок - RTM - Recovery Termination Manager. Важно, что IH вызывается первым в случае прерывания по ошибке.

z/OS предоставляет широкий спектр инструментов для перехвата ошибок (ESTAE, ESTAI, FRR), макрос ESPIE для перехвата прерываний.

Механизм ESTAE/ESTAI позволяет назначить двухступенчатый обработчик ABEND состоящий из пользовательских подпрограммы восстановления и подпрограммы возврата.



При возникновении исключения управление передается таким образом: программа->RTM->подпрограмма восстановления->RTM->подпрограмма возврата. В общем говоря механизм ESTAE/ESTAI создан не для перехвата ошибок в чужих программах, что делает его не очень удобным инструментом для реализации трассировщика. Кроме того, лишь ESTAI официально позволяет обрабатывать ошибки в другом потоке. Хотя, есть "хакерский" способ назначить для другого потока и обработчик ESTAE.

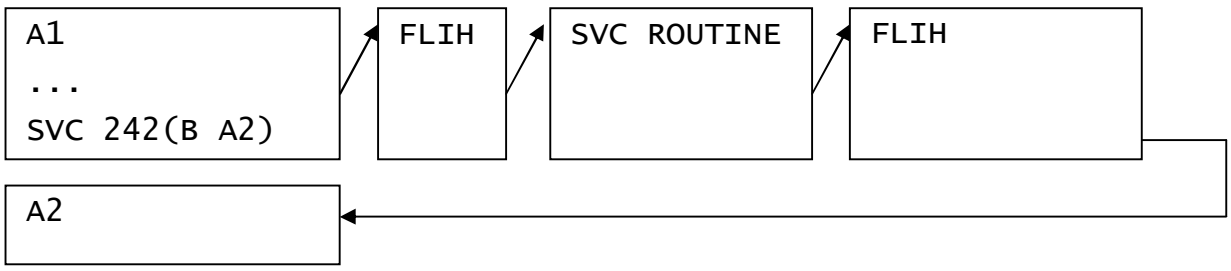
Оказывается, что при восстановлении после обработки ошибки ESTAI подпрограммой нельзя перейти в режим супервизора, если исследуемая программа исполнялась в режиме задачи. Этот факт приводит к следующим проблемам в подпрограмме возврата, а значит и в точке восстановления:

- ASC mode будет тот же что и у программы вызвавшей ATTACH. Исследуемая программа же может изменить свой address space mode, который тоже должен сохраниться после обработки перехода. В противном случае исследуемая программа может

вылететь, при попытке работы, например, с secondary address space

- Всегда разрешены IO interrupts и external interrupts. Запрет или разрешение IO прерывания могут быть важны, например, если исследуемая программа работает с канальной подсистемой
- Режим адресации тот же что при вызове ATTACH, если не определен иной с помощью макроса SETRP. Дело в том, что исследуемая программа может динамически изменить режим адресации и работать с 64-битными адресами. Внезапное переключение в 31 битную адресацию может плохо сказаться на ее работе
- Состояние флага CC (condition code) в PSW не определено. Может нарушиться логика условных переходов в программе
- Состояние флага R в PSW не определено (PER enable). Программа может быть чувствительна к прерыванию PER
- Состояние флага T в PSW не определено (DAT enable). Программа может отключить DAT, а потом обнаружить его включенным, что нарушит логику ее работы

4. Замена SVC - метод получил распространение в последнее время и в контексте организации трассировки называется HOOKING. В некотором смысле он является развитием метода внесения и перехвата ошибок. SVC - supervisor call - двухбайтная инструкция для осуществления системных вызовов.



Можно написать свою подпрограмму обработки SVC и заменять в исследуемой программе все необходимые инструкции на команду ее вызова. Это в целом является достаточно удобным и быстрым механизмом.

Минусы данного подхода:

- SVC нельзя вызывать из SRB
- SVC нельзя вызывать в cross memory mode
- Существует проблема установки соединения с трассировщиком, что, впрочем, решаемая задача.

Таким образом, были сформулированы основные подходы к трассировке переходов между модулями. С некоторыми ограничениями пригодны SVC, ESTAI, и комбинированные схемы SVC+ESPIE/SVC+TRAP. Комбинированные схемы используют SVC для установки ESPIE или TRAP окружения следующим образом:

1. Заменить обработчик SVC 242
2. Заменить команду в entry point исследуемой программы на инструкцию SVC 242
3. В обработчике SVC 242 устанавливаем TRAP/ESTAE/ESPIE и исполняем оригинальную команду

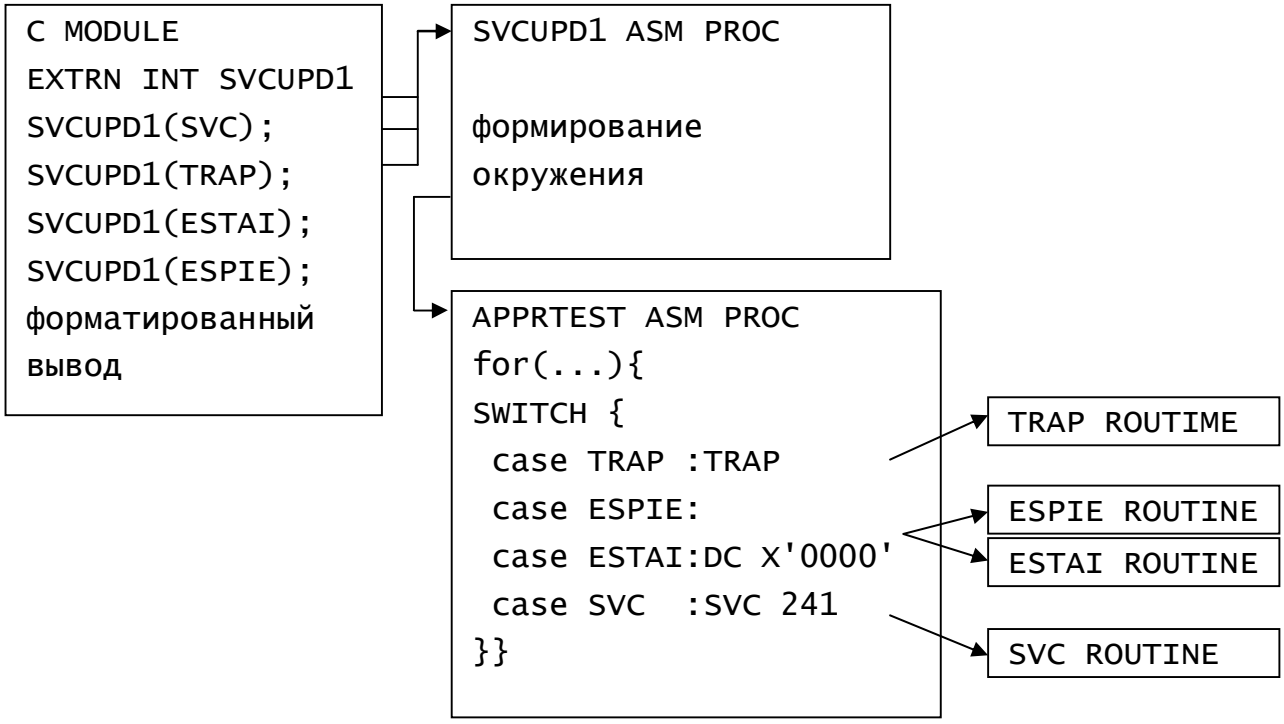
Это позволяет работать в том числе с приложениями secondary address space mode, например, с помощью ESPIE или ESTAE. Дополнительный

статический анализ перед запуском приложения позволит определить конкретные методы пригодные для трассировки разных участков кода приложения.

4. Оценка производительности предложенных методик

Для оценки производительности было разработано приложение, реализующее четыре типа перехвата переходов:

- Внесение ошибки и ее перехват с помощью ESTAI
- Внесение ошибки и ее перехват с помощью ESPIE
- Замена целевой инструкции на TRAP2
- Замена целевой инструкции на SVC



Разработанное приложение состоит из пяти модулей:

- TESTSYS - модуль, написанный на языке C, предназначен для вызова тестов, форматирования полученных данных и вывода их в набор данных
- SVCUPD1 - модуль, написанный на HLASM, устанавливает корректное окружение для тестирования (ESTAI, ESPIE SVC,TRAP) и замеряет время работы APPRTEST

- APPRTEST - сам тестирующий модуль, написанный на HLASM, вызывает некоторое количество раз либо ошибочную команду, либо TRAP2, либо SVC
- Модуль, написанный на HLASM и содержащий TRAP ROUTINE
- Модуль, написанный на HLASM и содержащий SVC ROUTINE

При разработке приложения возникло несколько проблем, которые были успешно преодолены:

- Использование inline JCL процедур при компиляции приводило к ошибке multiple source files, которая была решена использованием каталогизированных JCL процедур
- Оказалось, что компилятор GCCMVS транслирует имена функций в восьмибуквенные метки простым удалением лишних символов - имена функций были скорректированы.

Были получены следующие результаты на стократном повторении теста для "применения" метода трассировки на трех командах (единица измерения времени 0.000016 секунды):

```
********************************** TOP OF DATA **********************************
Start loop: 3
Step count: 100
=============================================
|ESTAI      |SVC        |TRAP       |ESPIE      |
|TIME       |TIME       |TIME       |TIME       |
|      12789|        928|          3|         88|
|      12927|        523|          2|        131|
|      12766|        370|          2|        137|
                    . . .
|      13940|        444|          3|        160|
|      13835|        412|          3|        120|
|      14253|        386|          5|        119|
|      12146|        371|          3|        213|
|      13213|        412|          3|        126|
|      15173|        571|          3|        136|
|      14342|        625|          3|        132|
|      14148|        539|          9|        143|
|      14530|        412|          3|        162|
|      14407|        498|          5|        170|
|      15977|        435|          3|        138|
|      14264|        411|          3|         96|
|      14406|        413|          3|        123|
|      13795|        566|          5|        134|
|      14418|        506|          5|        126|
|      14794|        456|          3|        124|
|      13710|        846|          3|        146|
|      14356|        539|          4|        125|
|      12841|        438|          3|         85|
|      13299|        438|          3|         92|
=============================================
|   13872.62|     484.84|       3.79|     134.58|general average
=============================================
|1923044.00|   17886.49|       4.20|     979.24|general dispersion
******************************** BOTTOM OF DATA ********************************
```

Наиболее производительным подходом оказался TRAP.

В итоге наиболее разумным оказывается реализовать трассировщик с использованием SVC. Его несложно реализовать, у него будет высокая скорость работы, у него не так много ограничений, а так же на нем можно отработать идеи для построения комбинированной схемы.

5. Разработка трассировщика

Предполагается, что система имеет доступ к ADATA - информации о компиляции исследуемой программы.

Система состоит из двух программ:

- Анализатора исследуемой программы, использующего ADATA. Он выбирает команды перехода для их замены в дальнейшем на команду SVC и генерирует требуемую таблицу для AMASPZAP и программы-монитора
- Программы-монитора, которая устанавливает окружение, запускает исследуемую программу, собирает и сохраняет трассировочную информацию

Информация от анализатора монитору передается через временный набор данных. Для запуска системы написана JCL процедура. Она делает следующее:

1) Вызывает анализатор, который строит две таблицы команд: одну для AMASPZAP, другую для программы-монитора. В первой содержатся смещения команд, которые нужно заменять. А во второй еще и их оригиналы
2) Делает резервную копию библиотеки с исполняемыми файлами
3) Вызывает AMASPZAP, который меняет требуемые команды
4) Запускает программу-монитор
5) Восстанавливает библиотеку из резервной копии

Таким образом, чтобы реализовать перехват прыжков типа BC в программах 31-битного режима в primary address space mode для операционной системы z/OS система:

1. Найдет все вхождения команды BC в бинарном файле исследуемой программы путем просмотра ADATA и заменит их с помощью AMASPZAP на SVC 241
2. Захватит SVC 241
3. Запустит исследуемую программу с подходящим набором входных данных
4. По окончании исполнения исследуемой программы сохранит трассировочную информацию в наборе данных

Этапы разработки:

1. Анализатор ADATA
2. Программа-монитор
3. Обработчик SVC 241

В ходе разработки возникло две существенные проблемы:

1. Получение монитором информации о модуле для организации прозрачной трассировки
2. Установка связи обработчика SVC с монитором для сохранения и вывода трассировочных записей

Первая проблема может быть решена двумя способами:

1. Исследование показало, что можно получить всю необходимую информацию о модуле самостоятельным просмотром системных структур данных: TCB, RB, LLE, CDE
2. Оказалось, что существует макрос CSVINFO, который может дать лишь часть необходимой информации по имени модуля, например,

нельзя определить точку загрузки. Поэтому первый способ оказался предпочтительнее

Вторая проблема решается с помощью использования специальных полей системной структуры DUCT (она уникальна для каждой подзадачи), которые помечены как "For use by programming".

Предлагается следующий алгоритм:

1. Программа-монитор захватывает SVC 241 и сохраняет адрес структуры данных, используемой для хранения смещений и оригиналов "испорченных" команд, в теле SVC рутины
2. Программа-монитор делает ATTACH исследуемой программы
3. При вызове SVC241 проверяется, поднят ли TRAP флаг:
    3.1. Если нет, то создается структура для хранения трассировочных записей, адрес структуры сохраняется в памяти, содержащей код SVC. Поднимается TRAP флаг. Адрес служебной области данных сохраняется в поле "For use by programming" системной структуры данных DUCT
    3.2. Если TRAP флаг поднят, то вызывается процедура добавления трассировочной записи, адрес которой заранее сохранен в теле SVC рутины и исполняется оригинал команды

В некоторых ситуациях использование SVC невозможно, например, например, внутри SRB или если исследуемая программа исполняется в cross memory mode. В таких случаях можно заменять команды перехода на TRAP. А в случаях, которые не подходят и для SVC и для TRAP, например, если исследуемая программа работает в режиме secondary address space, можно использовать ESTAI. Это обеспечит наибольшую скорость и наиболее полный охват имеющихся вариантов. Однако это не реализовано.

Процедура добавления трассировочной записи определяет по имени модуля, адресу в памяти и ADATA координаты точки перехода и точки назначения - это векторы вида (MODULE NAME, CSECT NAME, OFFSET). Для увеличения скорости определения координат, адреса модулей, их имена и имена CSECT хранятся в структуре данных, которая автоматически синхронизируется с системными структурами, как только в ней по какому-то запросу не нашлась соответствующая запись (на самом деле надо учитывать еще "внешние" воздействия на цепочку RB, но это уже более сложная задача).

Для трассировки многопоточного приложения используется один монитор на все потоки приложения и раздельные области памяти для сохранения трассировочных записей. Для этого усовершенствован механизм инициализации: процедура выделения памяти под записи для нового потока стала потокобезопасной. Указатель на данные же, специфичные для каждой конкретной подзадачи, хранится в поле "use for programming" системной структуры DUCT, которая уникальна для каждого потока. Это позволяет получать к ним доступ к данным из обработчика SVC без блокировок, мьютексов и просмотров очередей.

6. Возможности трассировщика

Система PARSE+SPYM позволяет трассировать команды BC для многопоточного приложения 31-битного режима в primary address space mode для операционной системы z/OS.

На вход системе подается библиотека с загрузочными модулями приложения (DD BACKUP.SYSUT1) и библиотека с соответствующей ADATA информаций (DD BLDTBL.ADATA). В параметре MAIN процедуры SPYMRUN указывается имя главного модуля.

На основе ADATA программа PARSE принимает решение, какие команды надо заменить на SVC 241 и генерирует два набора данных: один для AMASPZAP, другой для SPYM.

SPYM по набору данных строит таблицу команд, устанавливает SVC 241 и вызывает главный модуль исследуемой программы. По окончании ее работы информация записывается в DD GO.SPYM$OUT. Запись представляет собой два вектора вида (MODULE NAME, CSECT NAME, OFFSET): один для точки перехода другой для точки назначения. Это позволяет построить некоторое покрытие кода, что может быть полезно для определенных задач статического анализа.

7. Результаты

Разработано два приложения:

1. Программа для оценки скорости трассировки с использованием различных методик
2. Разработана система PARSE+SPYM, позволяющая трассировать команды BC в 31-битном режиме в primary address space mode для многопоточных приложений операционной системы z/OS с сохранением реентерабельности исследуемого приложения

Система реализована с использованием GCCMVS, HLASM, JCL и является open source. Кроме того, для использования с GCCMVS в среде z/OS реализована библиотека dirent.h (работа с библиотеками наборов данных) и создан ряд макросов (установка TRAP окружения, использование DYNAMIC ALLOCATION).

Система была испробована на нескольких тестовых программах. При разработке каждая функция системы была протестирована в отдельном модуле с заданным окружением.

Возможна дальнейшая модификация системы, а следующие факторы облегчат развитие и поддержку:

- Использование C и HLASM, а не чистого ассемблера
- Реализация SVC hooking позволяет развивать комбинированные схемы трассировки (SVC+ESPIE или SVC+TRAP)
- Программа для оценки скорости трассировки реализует все основные методики, которые фактически можно брать и переносить в PARSE+SPYM

Репозиторий: https://github.com/aton4eg/emc-spbsu-coop-mainframe

8. Список литературы

IBM. z/Architecture IBM Principles of Operation (SA22-7832-03). 2004

IBM. z/OS MVS Programming: Authorized Assembler Services Guide (SA22-7608-15). 2010

IBM // IBM: DUCT. URL: http://www.vm.ibm.com/pubs/cp510/DUCT.HTML (дата обращения 15.10.2014)

IBM. MVS Programming: Authorized Assembler Services Reference, Volume 1 (ALE-DYN) (SA22-7608-15). 2010

IBM. MVS Programming: Authorized Assembler Services Reference, Volume 2 (EDT-IXG) (SA22-7610-18). 2010

IBM. MVS Programming: Authorized Assembler Services Reference, Volume 3 (LLA-SDU) (SA22-7611-12). 2010

IBM. MVS Programming: Authorized Assembler Services Reference, Volume 4 (SET-WTO) (SA22-7612-12). 2010

IBM. Toolkit Feature Interactive Debug Facility User's Guide (GC26-8709-07). 2013

IBM. z/OS MVS System Codes (SA22-7626-22). 2010

Trap Diagnosis Facility Fact Sheet // Trap Diagnostic Facility. URL: http://www.arneycomputer.com/tdf/tdff.pdf (дата обращения 20.10.2014)

David B. Cole. z/XDC® RELEASE GUIDE z/XDC®Release z1.13 for z/OS // ColeSoft Marketing, Inc. | No Guesswork. See the Code. 2011. www.colesoft.com URL: http://www.colesoft.com/documentation/z1d/pdf/z-xdc%20release%20guide%20%28z1.13%29.pdf (дата обращения 10.10.2014)

Appendix A. Уровни безопасности: APF, Problem/Supervisor state

Для предотвращения несанкционированного доступа программ к системным ресурсам в z/OS существует несколько важных механизмов: APF (authorized program facility), problem/supervisor state и PSW key. APF - это регистрация библиотеки, которая дает программам из нее доступ к изменению своего STATE и PSW KEY.

Problem/supervisor state это два уровня безопасности. Problem не имеет доступа к "авторизованным" функциям системы и может обращаться только к тем областям памяти, что разрешены PSW key. Supervisor может абсолютно все.

Appendix B. Защита памяти: PSW key

PSW key это один из механизмов защиты памяти. Память делится на страницы по 4096 байт. Каждый блок имеет ключ и при попытке чтения/записи этот ключ сравнивается с ключом в PSW. Если в PSW key лежит 0, то мы имеем доступ к любому блоку памяти. Иначе ключ должен совпадать. Или подчиняться некоторым другим правилам (но это несущественно).

Appendix C. Определения

ATTACH - макрос для запуска бинарного модуля в другом потоке.

PSW - Program Status Word. Регистр процессора, содержащий, в частности, program counter и регистр флагов.

SVC - Supervisor Call. Позволяет вызывать подпрограммы исполняемые в supervisor mode

ABEND – (Abnormal End) прерывание по ошибке.

ESTAI, ESTAE, FRR - механизмы позволяющие устанавливать пользовательские обработчики ошибок

ESPIE - механизм, позволяющий устанавливать пользовательские обработчики прерываний с кодами 1-15.

Address space - метод разделения программ и виртуализации. Загрузочный модуль, исполняемый в одном address space в problem state изолирован от других address space и не подозревает о их существовании.

Primary address space - address space из которого процессор получает машинные инструкции.

Secondary address space - address space из которого процессор получает операнды для машинных инструкций.

Home address space - address space в котором находится TCB для программы.

Primary address space mode - режим работы программы при котором PAS=SAS.

Secondary address space mode - режим работы при котором PAS!=SAS.

ADATA - информация о компиляции ассемблерного модуля.

Task (задача) - аналог потока. А одном address space может работать несколько task

Subtask (подзадача) - аналог task. Запускается с помощью системного макроса ATTACH

TCB - Task Control Block. Описатель Task.

SRB - Service Request Block. Это специальная рутина, которую можно внедрить в другой Address space, где она будет исполняться асинхронно по отношению ко всем task (асинхронное взаимодействие)

Cross memory mode - метод синхронного взаимодействия между разными address space.

RTM - Recovery Termination Manager. В некотором смысле система обработки ошибок.

FLIH - First Level Interrupt Handler. Система вызова обработчиков прерываний

IH- Interrupt Handler. Обработчик прерывания.

Appendix D. GCCMVS+HLASM

Чтобы писать быстрее и эффективнее использовался GCCMVS. Существует три подхода для согласования двух языков:

- Использование ассемблерных вставок
- Использование ассемблерных вставок и макроса LOAD
- Статическая линковка

Самым удобным и честным способом является статическая линковка. Она позволяет унифицировать программные интерфейсы и не требует динамических проверок. Линковка двух модулей осуществляется параметром INCLUDE. Для этого надо изменить GCCCL - процедуру компиляции и линковки компилятора GCCMVS, а в коде программы указать с помощью extern все внешние объекты.
Соглашение о связях в GCCMVS подразумевает передачу аргументов в функцию через R1. Он указывает на область памяти, где в прямом порядке (слева направо к увеличению адресов) идут аргументы. Результат возвращается через R15. Учтя все это можно писать C-совместимые функции на HLASM и вызывать как C из HLASM так и HLASM из C.