Трассировка многомодульного приложения в среде операционной системы z/OS

Ефремов Р.С.
математико-механический факультет
СПбГУ
Санкт-Петербург, Россия
slava.efremov1994@mail.ru

***Аннотация*—Динамический анализ бинарного кода - методика, позволяющая с помощью информации о ходе выполнения программы на различных данных, получать данные о том, как она работает. Задача динамического анализа особенно актуальна для операционной системы z/OS, так как для нее существует множество программ, написанных десятки лет назад, которые до сих пор поддерживаются и развиваются. Mainframe - специфичная бизнес платформа, и публикации, освещающие в ее контексте некоторые проблемы достаточно редки. В частности, это относится к исследованиям в области динамического анализа в среде операционной системы z/OS. В данном исследовании рассмотрена эта проблема, а также определены основные методики получения графа динамических вызовов по результатам исполнения исследуемой программы. Кроме того, произведено сравнение их производительности, уровня воздействия на окружение исследуемой программы. Дополнительно в статье описан реализованный автором прототип трассировщика, который строит таблицу переходов для исследуемой программы. В ходе исследования оказалось, что решение центральной задачи динамического анализа: трассировки, сопряжено с рядом трудностей, связанных со специфичностью платформы. Тем не менее, выявлено два наиболее предпочтительных метода, которые используются в таких популярных коммерческих отладчиках как TDF и z/XDC. Эти подходы позволяют сохранить реентерабельность приложения, имеют слабые ограничения на окружение, являются достаточно производительными. Результаты исследования открывают возможности для разработки своих продуктов, осуществляющих динамический анализ, создания отладчиков. Кроме того, в работе рассматриваются некоторые слабо освещенные вопросы относительно архитектуры операционной системы z/OS.**

Ключевые слова—Mainframe; граф динамических вызовов; z/OS;

I. Введение

Динамический анализ бинарного кода - методика, позволяющая с помощью информации о ходе выполнения 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.

Целью данной работы 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 информацией о компиляции (ADATA) и скомпилированных с опцией THREAD. Во время трассировки должна сохраняться реентерабельность исследуемого приложения.

II. Существующие аналоги

A. IDF - Interactive Debug Facility [1]

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 [1]. Из этого сразу следуют существенные ограничения в работе отладчика, которые будут более подробно описаны в разделе 3. В частности, они привели к низкой популярности IDF среди системных программистов.

B. z/XDC - Extended Debugging Controller [2]

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 также 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 компании EMC) можно сделать вывод о том, что точно используются следующие методы перехвата: ESTAE, SVC hooking [2]. 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 статье рассмотрена самостоятельная реализация SVC hooking для трассировки многопоточного приложения.

C. TDF - Trap Debug Facility [3]

TDF - сравнительно новая программа, основанная на технологии TRAP, о чем явно указано в презентации продукта [3]. Из этого автоматически следуют ограничения в ее работе: нельзя исследовать приложения secondary address space mode и программы, использующие TRAP механизм. Также интерес представляет то, как устанавливается TRAP для исследуемой программы, и как эта программа запускается. В статье далее показано, как теоретически можно обойти эти ограничения, а также приведено сравнение скорости работы TRAP и SVC hooking технологий.

III. Подходы к перехвату переходов между модулями

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

A. *Создание (или модификация) виртуальной машины*

Создание виртуальной машины - весьма трудоемкая задача. Однако для Mainframe уже существует открытый бесплатный эмулятор Hercules. Его можно модифицировать для отслеживания определенных событий. Однако динамический анализ продуктов использующих возможности Mainframe, которые не поддерживаются в Hecules, будет просто невозможен. Например, это могут быть FICON или нестандартное hardware [4].

B. *Использование стандартных аппаратных средств организации отладки*

Изучение документации показало, что платформа Mainframe содержит аппаратное средство для реализации отладчиков, и называется оно PER - Program Event Recording [5]. 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. Можно попробовать выбрать память так, чтобы в ней лежали все модули интересующего нас приложения, но нет никакой гарантии, что между ними не окажется, например, кода, исполняющегося в другом потоке. Возникает идея выделить address space целиком, но это приводит к катастрофическому снижению производительности, так как система не справляется с обработкой переходов всех task, subtask, SRB, потому что в крупных системах их очень много
- Невозможность установки своего обработчика PER: ESPIE (сервис для установки обработчиков прерываний) позволяет назначить обработчик только для прерываний с кодами 0-15, 17 [6], а PER имеет код 80 [5]. Другими доступными методами установка также невозможна, так как IH - Interrupt Handler поддерживает область real memory с обработчиком прерываний от z/OS. Это FLIH - First Level Interrupt Handler [7]. Написать свой FLIH не представляется возможным из-за отсутствия доступа к определенной технической документации

Другое стандартное средство организации трассировки - инструкция TRAP [5]. 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 предоставляют быстрый и удобный сервис для перехода в некоторую подпрограмму с почти полным восстановлением контекста. Поскольку команда TRAP2 занимает 2 байта, её можно записать поверх любой другой инструкции и попасть в TRAP подпрограмму.

Однако при использовании команды TRAP возникают пять проблем:

- Окружение восстанавливается не полностью [5]
- В secondary address space mode команду TRAP вызвать нельзя [5]
- TRAP не может восстановить home space mode, если процессор не находится в привилегированном режиме (supervisor state) [5]
- Установка TRAP окружения для дочернего 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 устанавливать Trap-окружение для каждого subtask

C. Внесение ошибок и их перехват

Внесение ошибок и их перехват - очень распространенный метод организации отладки и перехвата переходов. Команда перехода заменяется, например, на код X'0000'. 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 [6]. Важно что IH вызывается первым в случае прерывания по ошибке.

z/OS предоставляет широкий спектр инструментов для перехвата ошибок (ESTAE, ESTAI, FRR) [8], макрос ESPIE [8] для перехвата прерываний.

Механизм ESTAE/ESTAI позволяет назначить двухступенчатый обработчик ABEND, состоящий из пользовательских подпрограммы восстановления и подпрограммы возврата [6].

При возникновении исключения управление передается таким образом: программа->RTM->подпрограмма восстановления->RTM->подпрограмма возврата [6]. 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. Стоит заметить, 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:

- Address space mode будет тот же что и у программы, вызвавшей ATTACH. Исследуемая программа же может изменить свой ASC, который должен не измениться после обработки перехода. В противном случае исследуемая программа может завершиться с ошибкой, при попытке работы, например, с secondary address space [6]
- Всегда разрешены IO interrupts и external interrupts. Запрет или разрешение IO прерывания могут быть важны, например, если исследуемая программа работает с канальной подсистемой [6]
- Режим адресации тот же что при вызове ATTACH , если не определен иной с помощью макроса SETRP [6]. 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 [6]. Может нарушиться логика условных переходов в программе
- Состояние флага R в PSW не определено (PER enable) [6]. Программа может быть чувствительна к прерыванию PER
- Состояние флага T в PSW не определено (DAT enable) [6]. Программа может отключить DAT, а потом обнаружить его включенным, что нарушит логику ее работы

D. 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 [5].

Можно написать свою подпрограмму обработки SVC [6] 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 [5]
- SVC нельзя вызывать в cross memory mode [5]
- Существует проблема установки соединения с трассировщиком, что, впрочем, решаемая задача

Итак, выявлены 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:

- Заменяют обработчик SVC 242
- Заменяют команду в entry point исследуемой программы на инструкцию SVC 242
- В обработчике SVC 242 устанавливают TRAP/ESTAE/ESPIE и исполняют 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.

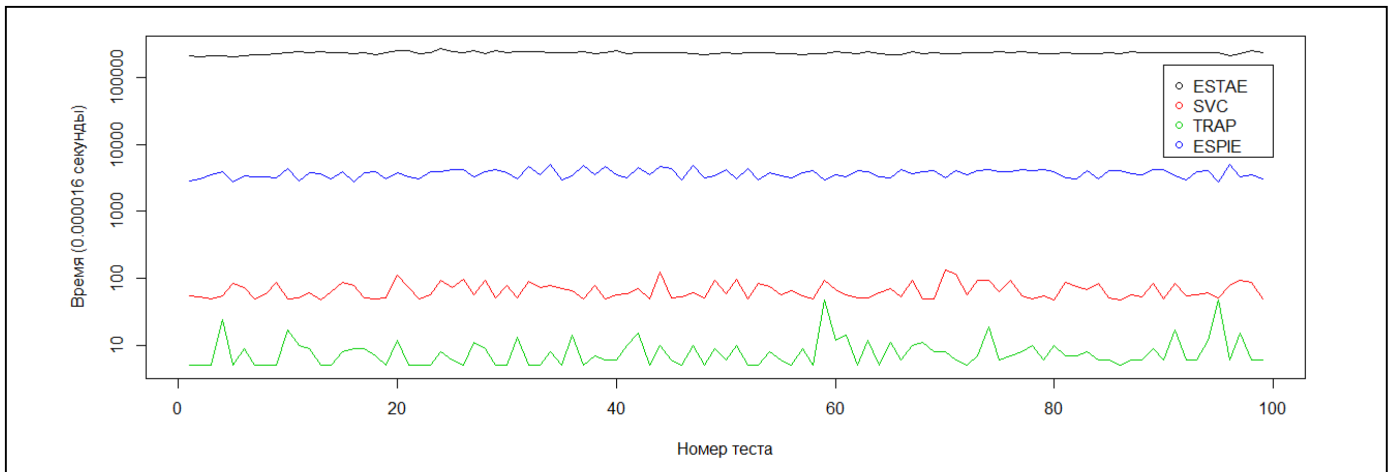

Рис. 1. График производительности, построенный по ста повторениям теста "применения" каждого метода трассировки на пятидесяти командах (единица измерения времени 0.000016 секунды)

IV. Оценка производительности предложенных методик

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

- Внесение ошибки и ее перехват с помощью ESPIE
- Внесение ошибки и ее перехват с помощью ESTAI
- 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

Стократное повторение теста "применения" метода трассировки на пятидесяти командах показало, что самым производительным подходом является использование TRAP facility или SVC, это показано на рис. 1. Стоит отметить, что это основной метод в продукте TDF. Затем идут ESPIE и ESTAI. Реализация TRAP подхода сопряжена с определенной трудностью: отсутствием легкого доступа к привилегированному режиму. Это затрудняет организацию коммуникации между программой - монитором и генератором трассировочных событий. У SVC механизма такой проблемы нет, а реализация трассировщика с его использованием позволит в дальнейшем добавить поддержку TRAP.

V. 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. Анализатор 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. Для запуска системы написана JCL процедура. Она делает следующее:

1. Вызывает анализатор
2. Создает 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. Установит свой обработчик 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:

- Анализатор ADATA
- Программа-монитор
- Обработчик SVC 241

В ходе разработки возникло две существенные проблемы:

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

Первая проблема может быть решена двумя способами:

- Изучение системных структур данных показало, что можно получить всю необходимую информацию о модуле самостоятельным просмотром системных структур данных: TCB, RB, LLE, CDE
- Оказалось, что существует макрос CSVINFO, который может дать лишь часть необходимой информации по имени модуля. Например, нельзя определить точку загрузки [9]. Поэтому первый способ оказался предпочтительнее

Вторая проблема решается с помощью использования специальных полей системной структуры DUCT [10] (она уникальна для каждой подзадачи), которые помечены как "For use by programming".

Предлагается следующий алгоритм:

1. Программа-монитор устанавливает обработчик SVC 241 и сохраняет адрес структуры данных, используемой для хранения смещений и оригиналов "испорченных" команд, в теле SVC рутины
2. Программа-монитор делает ATTACH [11] исследуемой программы
3. При вызове SVC241 проверяется, поднят ли TRAP флаг:
   a. Если нет, то создается (или расширяется под новый поток) структура для хранения трассировочных записей. Ее адрес хранится в памяти, содержащей код SVC, что затем позволяет легко организовать вывод собранной информации в датасет в конце работы трассировщика. Поднимается TRAP флаг. Адрес служебной области данных сохраняется в поле "For use by programming" системной структуры данных DUCT
   b. Если TRAP флаг поднят, то вызывается процедура добавления трассировочной записи. Адрес процедуры заранее сохранен в теле SVC рутины. В конце эмулируется исполнение оригинала команды, и управление возвращается

В некоторых ситуациях использование SVC невозможно, 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 пока 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. Создан механизм инициализации, включающий в себя потокобезопасную процедуру выделения памяти под записи для нового потока. Указатель на данные 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 из обработчика SVC без блокировок, мьютексов и просмотров очередей.

```
******************************** TOP OF DATA *********************************
THREAD=START====================
 DEPARTURE                    DESTINATION
 MDL NAME  | CSECT NM  | OFFSET | MDL NAME  | csect NM  | OFFSET |

SMPL1M1    | SMPL1M1   |      8 | SMPL1M1   | SMPL1M1   |     14 |

SMPL1M1    | SMPL1M1   |     34 | SMPL1M1   | SMPL1M1   |     80 |

SMPL1M2    | SMPL1M2   |      8 | SMPL1M2   | SMPL1M2   |     14 |

SMPL1M3    | SMPL1M3   |      8 | SMPL1M3   | SMPL1M3   |     14 |

SMPL1M4    | SMPL1M4   |      8 | SMPL1M4   | SMPL1M4   |     14 |

SMPL1M4    | SMPL1M4   |     5E | SMPL1M3   | SMPL1M3   |     5C |

SMPL1M3    | SMPL1M3   |     76 | SMPL1M2   | SMPL1M2   |     5C |

SMPL1M2    | SMPL1M2   |     76 | SMPL1M1   | SMPL1M1   |     B0 |

SMPL1M1    | SMPL1M1   |     CA |           |           |      0 |
THREAD=END======================

THREAD=START====================
 DEPARTURE                    DESTINATION
 MDL NAME  | CSECT NM  | OFFSET | MDL NAME  | csect NM  | OFFSET |

SMPL1M6    | SMPL1M6   |      8 | SMPL1M6   | SMPL1M6   |     14 |

SMPL1M6    | SMPL1M6   |     50 | SMPL1M6   | SMPL1M6   |     58 |

SMPL1M6    | SMPL1M6   |     58 | SMPL1M6   | SMPL1M6   |     54 |

SMPL1M6    | SMPL1M6   |     54 | SMPL1M6   | SMPL1M6   |     5C |

SMPL1M6    | SMPL1M6   |     84 |           |           |      0 |
THREAD=END======================

******************************** BOTTOM OF DATA ******************************
```

Рис. 2. Пример результата работы прототипа трассировщика на пятимодульном двухпоточном приложении

VI. Об использовании трассировщика

Система PARSE+SPYM позволяет трассировать команды BC 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 о замене определенных команд 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. Трассировочная запись представляет собой два вектора вида (MODULE NAME, CSECT NAME, OFFSET): один для точки перехода другой для точки назначения. Пример итоговой таблицы на рис. 2. Таблица позволяет построить некоторое покрытие кода, что может быть полезно для определенных задач динамического и статического анализа.

VII. Результаты

Разработано два приложения:

- Программа для оценки скорости различных методик трассировки
- 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 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

Путем возможной замены модуля PARSE, расширения возможностей по эмуляции исполнения команд и добавления постпроцессинга трассы, подобная система может быть, например, использована в следующих случаях:

- Поиск состояния гонок по SLIP трассе изменений памяти и трассе работы примитивов синхронизации, полученной с помощью системы
- Модификация и отладка приложений с помощью трассы исполнения
- Построение профайлера для сбора информации о медленно работающих участках кода

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

Литература

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

[2] 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)

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

[4] Hercules Version 3: Frequently-Asked Questions. URL: http://www.hercules-390.org/hercfaq.html (дата обращения 29.10.2015)

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

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

[7] IBM. ABCs of z/OS System Programming, Том 10 (5694-A01). 2012

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

[9] IBM. MVS Programming: Assembler Services Reference ABE-HSP (SA23-1369-00). 2013

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

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