Технология и методы программирования

Тема 6. Отладка и инструментация

Отладка. А зачем?

- Проверить, что всё работает так, как ожидается.
- Разобраться что и где надо исправить в коде, если найдена ошибка.

Методы отладки

- Логи (a.k.a. Print-debugging) — добавление в программу подробных логов, описывающих состояние программы
- Бисекция —способ локализации ошибки, с помощью которого:
  - Последовательно выключаются куски кода, чтобы понять в каком из них проблема
  - Наоборот, последовательно включаются куски кода
  - Методом перебора находится коммит, в котором содержится ошибка
- Применение отладчика
  - Локальная отладка (отлаживаем процесс на той же самой машине)
  - Удалённая отладка (отлаживаем процесс по сети на другой машине)
- Исследование дампа памяти (core dump)

Методы отладки. Продолжение

- Инструментирование/инструментация — добавление в уже скомпилированное ПО блоков кода, осуществляющих анализ в момент выполнения
- Трассировка — подробная запись действий, выполняемых программой, и сообщений о событиях, происходящих во время работы программы
- Профилирование — сбор информации о запущенном процессе. Например сбор информации о выделенной памяти

Отладчики. Типы

- На уровне ассемблера — позволяет смотреть содержимое регистров и дизассемблированный код.
- На уровне исходного кода — позволяет смотреть в использованные в коде переменные и видеть на какой строчке кода находится выполнение

Пояснение про ассемблер-дизассемблер

Отладочная информация

- Компилируемая программа отлаживается на уровне машинных команд (или команд ВМ интерпретатора языка, например python, java)
- Требуется сопоставление объектов низкого уровня (адресов, неструктурированных данных) с объектами высокого уровня (функциями, строками кода, типизированными данными)
- Компилятор может генерировать дополнительную отладочную информацию, в которой описано соответствие между ежду сгенерированным бинарным кодом и исходниками, из которых он был получен
- Форматы отладочной информации:
  - DWARF — отладочная информация в ELF файлах
  - Microsoft PDB — отладочная информация в отдельном PDB файле

DWARF. Содержимое файла

- Типы данных, используемые в программе (поддержка основных компилируемых языков программирования — Си, Си++, Fortran, Ada и родственных)
- Точки входа — адреса начал функций, их имена и сигнатуры.
- Отображение адресов в бинарном коде на имена файлов и номера строк исходного кода
- Описание переменных: имена, типы, расположение (смещение относительно стека)

Отладочная информация. Как сгенерировать

- В IDE собирать **debug build**
- В MSVC использовать флаг **/debug:FULL**
- В gcc/clang использовать флаг **-ggdb**

Базовый функционал отладчика

- **Точка останова (breakpoint)** — адрес инструкции, при достижении которой выполнение программы останавливается.
- **Точка отслеживания (watchpoint)** — адрес ячейки памяти, при обращении к которой на чтение или запись, выполнение программы останавливается
- **Пошаговое выполнение программы** — выполнение одной машинной команды или одной строки исходного кода (с использованием отладочной информации)

  В момент, когда выполнение программы приостановлено, отладчик предоставляет возможность просмотра содержимого регистров и памяти

Пошаговое выполнение. Виды

- **Step into** — шаг с «заходом». Если текущая строчка — вызов функции, то проваливаемся внутрь
- **Step over** — шаг с «обходом». Идём на следующую строчку вне зависимости от того, какая инструкция
- **Step out** — выход из текущей функции на предыдущую.

Отладчики. Примеры

Linux:

- Gdb (консольный отладчик)
- Ddd (графический интерфейс к gdb)

Windows:

- WinDBG
- X64Dbg

Ваша любимая IDE

DEMO

Gdb CheatSheet.

команда	описание	Пример использования
help	помощь по команде	help r
run	запустить процесс на выполнение	r arg1 arg2
break ADDR_NAME	установить точку останова на адресе или имени ADDR_NAME	b main
continue	Продолжить выполнение программы	c
info breakpoints	инфа о всех существующих точках останова	
info registers	инфа о содержимом регистров	
backtrace	показать стек вызова	bt
step [n], **stepi** [n]	перейти на следующую линию(s) инструкцию(si)	si 10 — прошагать 10 инструкций s — перейти на следующую линию
next [n], **nexti** [n]	перейти на следующую линию или инструкцию заходя внутрь функций	
finish	дождаться завершения функции	
list [func_name]	показать исходный код функции	list
disassemble [addr]	показать дизассемблированный листинг	disass main

Как работают отладчики? Точки останова

Точка останова реализуется через вставку команды **INT 3** в тело программы по нужному адресу

- **INT 3** кодируется одним байтом 0xCC
- Отладчик записывает 0xCC по нужному адресу (сохраняя предыдущее содержимое)
- Когда процессор выполняет инструкцию **INT 3** он генерирует отладочное прерывание, которое перехватывается отладчиком.

Bonus: https://wiki.osdev.org/CPU_Registers_x86-64

Как работают отладчики? Пошаговое выполнение

- Установка бита TF в регистре EFLAGS приводит к тому, что после выполнения каждой очередной команды генерируется отладочное исключение.

Как работают отладчики? Точки отслеживания

Архитектура x86 (и x86-64) включает отладочные регистры DR0..DR7, позволяющие, в том числе, задать четыре точки отслеживания.

Для каждой точки отслеживания указывается виртуальный адрес, размер (1, 2, 4 или 8 байтов) и один из четырёх режимов:

- отслеживать только чтение команд;
- отслеживать только обращения на запись;
- отслеживать любые обращения;
- отслеживать обращения на чтение и запись данных, но не исполнение команд.

Таким образом, этот механизм позволяет также задавать и точки останова без модификации образа программы в памяти

Инструментирование

Статическое инструментирование выполняется однократно перед запуском программы.

Статическое инструментирование может выполняться:

- на уровне исходного кода — вручную или автоматически;
- на уровне бинарного кода — вручную или автоматически.

“Print-debugging” — пример выполненного вручную статического инструментирования на уровне исходного кода с целью отладки программы

Динамическое инструментирование выполняется в процессе работы программы

Инструментирование. Примеры ПО

- Valgrind — средство для отладки использования памяти
- strace/ltrace — трассировщики системных вызовов/библиотечных функций
- gprof — профилировщик C кода.

Отладка/инструментирование. Серая сторона

Где ещё активно применяется отладка и инструментирование?

- Расширение/удаление функционала ПО без доступа к исходникам (плагины, кряки, читы)
- Обратная разработка (reverse engineering)

The end