Технологии QNX и КПДА в России

Санкт-Петербург, 23 ноября 2017

Анализ низкоуровневых системных событий при профилировании и отладке приложений

Игорь Рондарев, ООО «СВД Встраиваемые Системы»

СВД Встраиваемые Системы

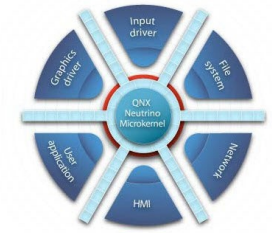
SWD Software

Содержание

- Введение в системное профилирование
- Методы анализа системных событий
- Библиотека libtraceparser
  - Общие сведения
  - Программный интерфейс (API)
  - Примеры использования
- Выводы и рекомендации
  - «С чего начать?»

Системное профилирование - основы

- Технология системного профилирования (**System Profiling**)
  - Позволяет анализировать поведение приложений на системном уровне, т. е. в **процессе их функционирования и взаимодействия с другими компонентами системы**
  - Позволяет протоколировать происходящие в системе события (относящиеся к работе приложений, системных служб, оборудования) с наименьшим влиянием на её поведения (**низкие накладные расходы**)
    - Используются т. н. «**события**» (**events**)
  - Включает в себя **различные инструменты** для обработки и анализа полученной информации



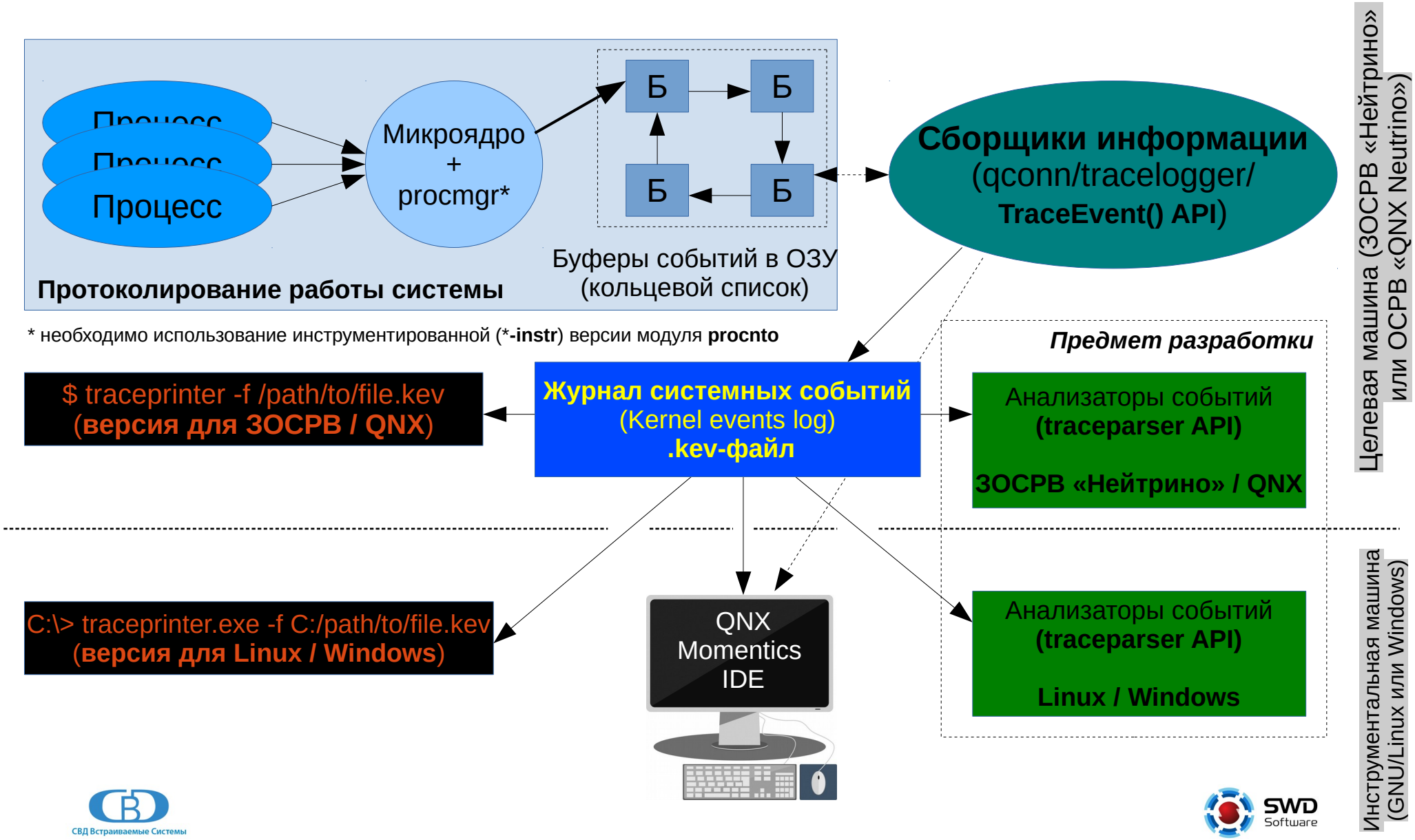
СВД Встраиваемые Системы

SWD Software

Системное профилирование — типы системных событий

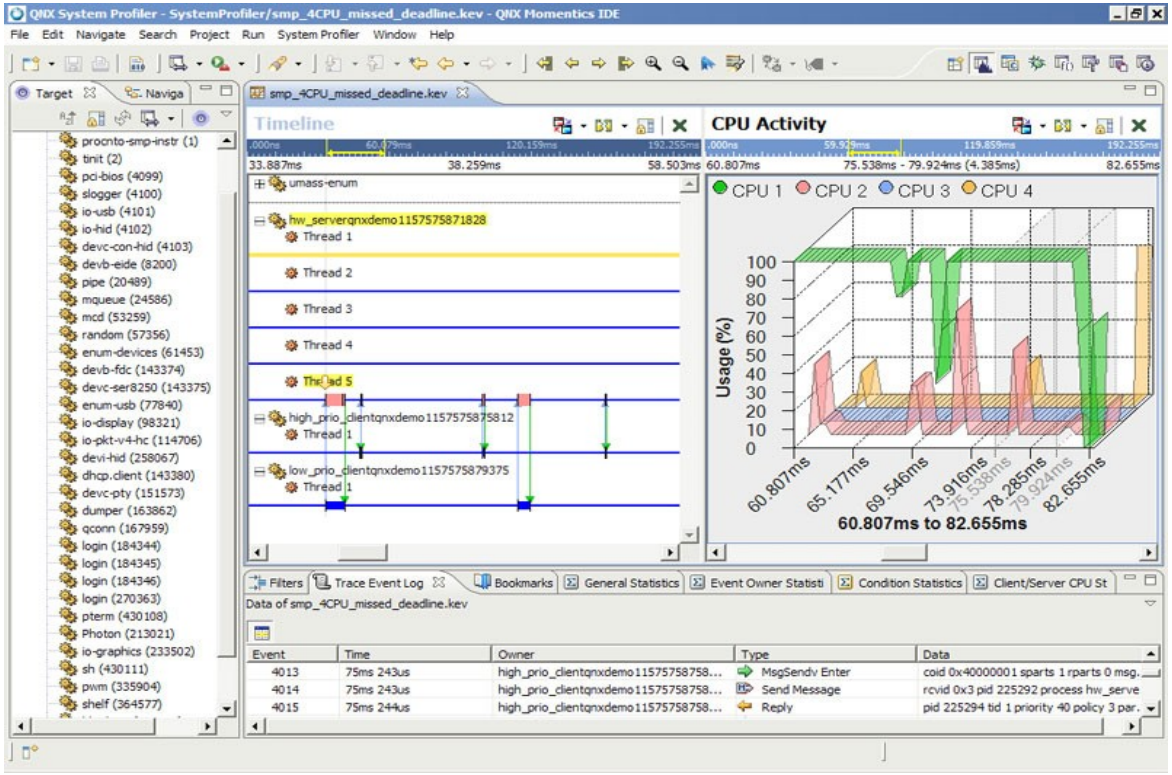
- Создание/завершение процессов и потоков
- Прерывания
- Смена состояния потоков (диспетчеризация)
- Системные вызовы
- **Пользовательские события (т. н. «user events»)**
  - Позволяют отмечать наиболее важные фрагменты в коде приложения и снабжать их произвольной дополнительной информацией для последующего анализа
  - Содаются с помощью функций, входящих в состав **TraceEvent() API**, в частности, группы функций **trace_log*()**

Системное профилирование — общая схема



Методы анализа системных событий

- Использование инструментария из состава среды разработки QNX Momentics IDE (набор инструментов **System Profiler**)
  - Богатая функциональность, возможность оценить и проанализировать данные с различных точек зрения (фильтры, анализ взаимосвязей, статистика и т.д.)
  - Ограниченные возможности автоматизации действий, высокие требования к характеристикам инструментальных ПК (в первую очередь, к производительности процессора и объёму ОЗУ)



СВД Встраиваемые Системы

SWD Software

Методы анализа системных событий (продолжение)

- Консольная утилита traceprinter
  - **Присутствует и на целевых, и на инструментальных машинах**
  - Не всегда применима в исходном виде (избыточность, необходимость использования внешних инструментов работы с текстом - таких, как **sed, awk, grep** и т. д.)

```
t:0x4f0a013d CPU:01 KER_CALL:MSG_RECEIVEV/14 chid:0x00000001 rparts:1
t:0x4f0a0c45 CPU:01 THREAD  :THRECEIVE     pid:4103 tid:1
t:0x4f0a1771 CPU:01 THREAD  :THRUNNING     pid:225304 tid:1
t:0x4f0a28a7 CPU:01 KER_EXIT:MSG_SENDV/11 status:4 rmsg:"" (0x00000000)
t:0x4f0acf3b CPU:01 KER_CALL:SIGNAL_PROCMASK/30 pid:0 tid:0
t:0x4f0adfcf CPU:01 KER_EXIT:SIGNAL_PROCMASK/30 ret_val:0 old_sig_blocked->bits[0]:0x00000000
t:0x4f0af507 CPU:01 KER_CALL:SIGNAL_PROCMASK/30 pid:0 tid:0
t:0x4f0afc0d CPU:01 KER_EXIT:SIGNAL_PROCMASK/30 ret_val:0 old_sig_blocked->bits[0]:0x00000000
t:0x4f0b8779 CPU:01 KER_CALL:SIGNAL_PROCMASK/30 pid:0 tid:0
t:0x4f0b8e67 CPU:01 KER_EXIT:SIGNAL_PROCMASK/30 ret_val:0 old_sig_blocked->bits[0]:0x00000000
t:0x4f0b9a2f CPU:01 KER_CALL:SIGNAL_PROCMASK/30 pid:0 tid:0
t:0x4f0ba0c7 CPU:01 KER_EXIT:SIGNAL_PROCMASK/30 ret_val:0 old_sig_blocked->bits[0]:0x00000000
t:0x4f0bd583 CPU:01 KER_CALL:MSG_SENDV/11 coid:0x00000001 msg:"" (0x00100102)
t:0x4f0bdf99 CPU:01 COMM    :SND_MESSAGE   rcvid:0x00000018 pid:4103
t:0x4f0bf26d CPU:01 THREAD  :THREPLY       pid:225304 tid:1
t:0x4f0bf8dd CPU:01 THREAD  :THRUNNING     pid:4103 tid:1
t:0x4f0c0127 CPU:01 COMM    :REC_MESSAGE   rcvid:0x00000018 pid:4103
t:0x4f0c05ed CPU:01 KER_EXIT:MSG_RECEIVEV/14 rcvid:0x00000018 rmsg:"" (0x00100102)
t:0x4f199be2 CPU:00 INT_ENTR:0x00000000 (0)       IP:0xf008be4e
t:0x4f19a482 CPU:00 INT_HANDLER_ENTR:0x00000000 (0)       PID:1 IP:0xf005c3a0 AREA:0x00000000
t:0x4f19be0c CPU:00 INT_HANDLER_EXIT:0x00000000 (0) SIGEVENT:INTR EVENT
t:0x4f19c49e CPU:00 INT_EXIT:0x00000000 (0) inkernel:0x01000001
t:0x4f33cd0d CPU:01 KER_CALL:MSG_REPLYV/15 rcvid:0x00000018 status:0x00000004
t:0x4f33e05b CPU:01 COMM    :REPLY_MESSAGE tid:1 pid:225304
t:0x4f33e859 CPU:01 THREAD  :THREADY       pid:225304 tid:1
t:0x4f33f187 CPU:01 KER_EXIT:MSG_REPLYV/15 ret_val:0 empty:0x00000000
```

СВД Встраиваемые Системы

SWD Software

Методы анализа системных событий (продолжение)

- Создание собственных анализаторов с использованием библиотеки **libtraceparser**
  - позволяет автоматизировать решение широкого класса задач, связанных с диагностикой, профилированием, тестированием и отладкой ПО
  - целесообразно, в первую очередь, для анализа уже известных механизмов и особенностей работы приложений
- Создание собственных средств сбора данных (аналогов утилиты **tracelogger**)
  - API открыт и документирован (*TraceEvent() API*)
    - Hint: код некоторых компонентов, в т.ч. **tracelogger**, **traceprinter** и **libtraceparser** (был) доступен в рамках проекта OpenQNX

Библиотека libtraceparser

- Общие сведения
  - Статическая библиотека libtraceparser.a (размер ~20 КБ)

```
# i486-pc-nto-qnx6.5.0-gcc -o my_analyzer my_analyzer.c -ltraceparser
```

  - Входит в состав **ЗОСРВ «Нейтрино»** и **QNX 6.x/7.0**
  - Может использоваться при разработке анализаторов, запускаемых как непосредственно **на целевых** платформах, так и **на инструментальных**
    - сборки **libtraceparser** для **Linux и Windows** доступны по запросу (support@kpda.ru)
  - Документация
    - **QNX® Neutrino® System Analysis Toolkit User's Guide**
    - Раздел **«Диагностическая версия микроядра»** на сайте **http://docs.kpda.ru**

traceparser API (программный интерфейс)

- Заголовочные файлы
  - ✔ **#include <sys/traceparser.h>** **/* traceparser API */**
  - ✔ **#include <sys/trace.h>** **/* TraceEvent API — коды событий и функции */**
  - ✔ **#include <sys/kercalls.h>** **/* коды системных вызовов (если требуются) */**
- Функции, управляющие механизмом анализа системных событий (парсером)

Функция или семейство функций	Назначение
*traceparser_**init**()/traceparser_**destroy**()*	Инициализация/деинициализация парсера
*traceparser_**cs**()/traceparser_**cs_range**()*	Установка обработчиков (т. н. **callback-функций**) на индивидуальные типы событий или на группы типов
*traceparser_**get_info**()*	Получение информации о текущем состоянии парсера, а также дополнительных сведений об анализируемых событиях
*traceparser_**debug**()*	Настройка режима вывода
traceparser()	Запуск парсера на заданном наборе входных данных (напр., **.kev-файле**)

Примеры использования

```
#include <stdio.h>
#include <sys/kercalls.h>
#include <sys/trace.h>
#include <sys/traceparser.h>

static unsigned int g_usr_ev_counter;

/* EXAMPLE: callback for simple counting user events */
static int usr_ev(struct traceparser_state* tp_state, void* d, unsigned h,
                  unsigned t, unsigned* p, unsigned l)
{
    printf("Got user event %d with %d bytes of payload (ts = 0x%x).\n",
           _NTO_TRACE_GETEVENT(h),l, t);
    g_usr_ev_counter++;
    return 1;
}

int main(int argc, char **argv)
{
    char * input_file = (argc > 1) ? argv[1] : "/dev/shmem/tracebuffer.kev";
    struct traceparser_state* tp_state;

    /* Configuring parser's output with _TRACEPARSER_DEBUG_* flags */
    unsigned tp_flags = _TRACEPARSER_DEBUG_NONE;

    /* Initialize the traceparser state structure */
    tp_state = traceparser_init(NULL);

    /* Configure parser output mode */
    traceparser_debug(tp_state, stdout, tp_flags);

    /* Setting callback to specific range of events (e.g. _NTO_TRACE_USER) */
    traceparser_cs_range(tp_state, NULL, usr_ev, _NTO_TRACE_USER,
                         _NTO_TRACE_USERFIRST, _NTO_TRACE_USERLAST);

    /* Parsing kev data */
    if(0 == traceparser(tp_state, NULL, input_file))
        printf("Total number of user events: %u\n", g_usr_ev_counter);
    else
        perror("traceparser");  /* use traceparser_get_info() for additional information */

    traceparser_destroy(&tp_state); /* Freeing traceparser state structure */

    return 0;
}
```

Callback-функция для событий класса **_NTO_TRACE_USER** («пользовательские» события)

Срабатывает при нахождении очередного события соответствующего класса во входном массиве событий (.kev-файле)

Инициализация

Настройка режима вывода

Установка обработчиков

Выбор файла, содержащего исходные данные и **запуск анализатора**

Деинициализация

СВД Встраиваемые Системы

SWD Software

Возможности

- Гибкая конфигурация
  - Подключение **callback**-ов только к интересующим событиям
  - Возможность просмотра информации о системе, на которой эти события были собраны (в т.ч. содержимого «системной страницы» **syspage**)
- Возможость многократного анализа информации
  - несколько итераций запуска функции traceparser(), в т.ч. с возможностью модификации списка callback-ов
  - Пример:
    - **Проход №1**: подсчёт числа анализируемых объектов, составление списков/таблиц, выделение памяти
    - **Проход №2**: расчёт значений интересующих параметров приложения

Примеры использования (продолжение)

- Жизненный цикл объектов (экземпляров классов, структур, потоков, процессов и т.д.)

Пример: анализ времени жизни объектов (user event'ы в конструкторах и деструкторах классов)

```
#define     MYCLASSWITHTRACE_CREATE_ID      (_NTO_TRACE_USERFIRST+10)
#define     MYCLASSWITHTRACE_DESTROY_ID     (_NTO_TRACE_USERFIRST+11)
class MyClassWithTrace
{
public:
    unsigned m_objectId;
    SomeClassWithTrace()        /* constructor */
    {
        trace_logb(MYCLASSWITHTRACE_CREATE_ID,
            m_objectId,0);
    }
    ~SomeClassWithTrace()   /* destructor */
    {
        trace_logb(MYCLASSWITHTRACE_DESTROY_ID,
            m_objectId,0);
    }
};
```

Функции **trace_log*()** - создание пользовательского события, содержащего в качестве полезной нагрузки некоторые данные (переменные, массивы, структуры), а также служебные данные (временная метка, ядро CPU и т.д.)

Зафиксированы и доступны для дальнейшей обработки:

- Неявно: время создания объекта, время удаления объека (временные метки)
- Явно: идентификатор объекта (в численном виде)

Примеры использования (продолжение)

- Пример обработки событий (измерение временного отрезка)

```
/* EXAMPLE: callback for SIMPLIFIED object lifetime counting (see t details) */
static unsigned obj_lt;

static int obj_creat(struct traceparser_state* tp_state, void* d, unsigned h,
                     unsigned t, unsigned* p, unsigned l)
{
    obj_lt = *p;    return 1;
}
static int obj_destr(struct traceparser_state* tp_state, void* d, unsigned h,
                     unsigned t, unsigned* p, unsigned l)
{
    if(obj_lt) obj_lt = *p – obj_lt; return 1;
}

int main(int argc, char **argv)
{
    ...

    /* Setting callback to specific events (e.g. _NTO_TRACE_USER) */
    traceparser_cs(tp_state, NULL, obj_creat, _NTO_TRACE_USER,
                                   MYCLASSWITHTRACE_CREATE_ID)
    traceparser_cs(tp_state, NULL, obj_destr, _NTO_TRACE_USER,
                                 MYCLASSWITHTRACE_DESTROY_ID)
    ...
}
```

Примеры использования (продолжение)

- Формирование отчётов на основании диагностических данных, переданных с минимальным влиянием на поведение функционального ПО («легковесное» журналирование)
  - Пример: текстовое сообщение длиной 16 байт:
    - **printf(): 25µs, slogf(): 7µs, trace_logf(): 1µs**
  - Ключ slogger -u EventId
    - **Дублирование slog()-сообщений в пространство Kernel Events**
- Функциональное тестирование (анализ промежуточных результатов после завершения работы программы)
- и т. д.

Примеры использования (продолжение)

- Измерение времени выполнения функций, алгоритмов, механизмов межпроцессного взаимодействия и т.д.

Пример: user event'ы в ключевых точках кода вычислительных алгоритмов

```
#define    PROCESSING_START_ID        (_NTO_TRACE_USERFIRST+20)
#define    PROCESSING_STOP_ID         (_NTO_TRACE_USERFIRST+21)

unsigned math_calculations(void* data)
{
    unsigned result;

    trace_logf(PROCESSING_START_ID, "%s: started", __FUNCTION__);
    ...
    /* some_variable_time_long_running_calculation_on_data */
    ...
    trace_logf(PROCESSING_STOP_ID, "%s: result: %u", __FUNCTION__,
                                    result);

    return (result);
};
```

Зафиксированы и доступны для дальнейшей обработки:

- Неявно: время начала выполнения алгоритма, время окончания выполнения алгоритма
- Явно: имя вызывающей функции, результат (в символьном виде)

Примеры использования (продолжение)

- Пример обработки событий (журналирование)

```
/* EXAMPLE: callback for simple user events logging */
FILE* some_log_fp;

static int usr_ev(struct traceparser_state* tp_state, void* d, unsigned h,
                  unsigned t, unsigned* p, unsigned l)
{
    char* message = (char*)p;
    message[l-1] = 0;
    fprintf(some_log_fp, "(ts = 0x%x) %s\n", t, message);
    return 1;
}

int main(int argc, char **argv)
{
    ...

    /* Setting callback to specific range of events (e.g. _NTO_TRACE_USER) */
    traceparser_cs_range(tp_state, NULL, usr_ev, _NTO_TRACE_USER,
                         PROCESSING_START_ID, _PROCESSING_STOP_ID)
    ...
}
```

Вывод анализатора:

```
(ts = 0x01020304) math_calculations: started
(ts = 0xFFFFFFFE) math_calculations: result: 42
```

Достоинства и недостатки **libtraceparser**

- **Достоинства**
  - Позволяет автоматизировать решение широкого класса задач
  - Отсутствие привязки к платформе
    - Наличие сборок библиотеки для **ЗОСРВ «Нейтрино» / QNX / Linux / Windows**
  - Компактный API
  - Подробная документация
- **~~Недостатки~~ Особенности**
  - Требует тщательного исследования задачи перед созданием анализатора (потоки, объекты, интересующие события и т.д.)
  - По умолчанию работает только напрямую с «.kev»-файлами (не работает в динамике)
    - **Примечание**: с точки зрения технологии системного профилирования анализ событий «на лету» возможен только при использовании соответствующих низкоуровневых механизмов

С чего начать?

- **Документация**
  - **QNX® Neutrino® System Analysis Toolkit User's Guide**
  - **Раздел «Диагностическая версия микроядра»**
  - **Учебный курс**
    - **Разработка, отладка и анализ производительности с помощью QNX Momentics IDE**
- **Примеры**
  - документация (описание API)
  - проект OpenQNX*
    - **traceprinter**, libtraceparser, tracelogger
- **Техническая поддержка**
  - support@kpda.ru

* данный исходный код официально не поддерживается («no warranty, use at your own risk»)

Вопросы?

- Вопросы?

Спасибо за внимание

Игорь Рондарев

инженер-программист

ООО «СВД Встраиваемые Системы»

тел.: +7 (812) 346-8956
факс: +7 (812) 346-8953

http://www.kpda.ru | http://forum.kpda.ru | http://www.swd.ru

СВД Встраиваемые Системы

SWD Software