МАРАФОН
CAN технология

Асинхронный регистратор

Техническое описание

Москва, 2022

Оглавление

1. Основные характеристики

Асинхронный регистратор выполнен как проект для среды разработки MS Visual Studio 2015.

1.1 Структурная схема регистратора

Регистратор позволяет записывать события, порожденные в различных областях программной системы: потоках и нитях, обработчиках прерываний и сигналов.

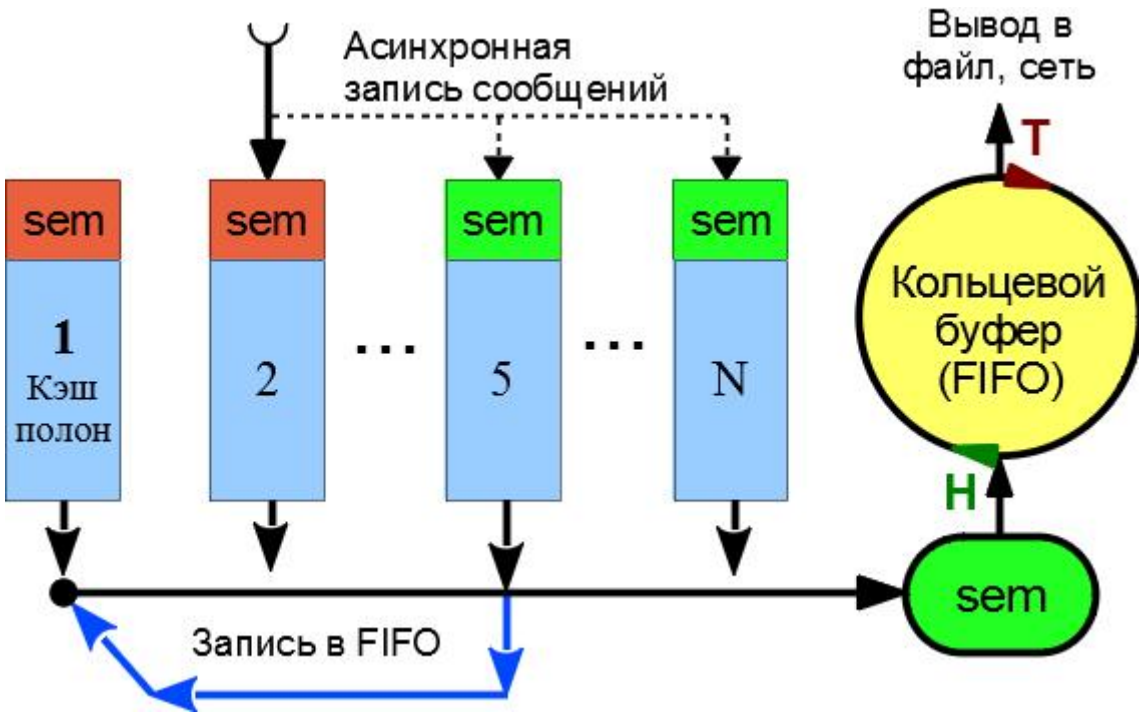

Рис 1. Асинхронный регистратор событий.

На рис. 1 приведена схема подсистемы регистратора, отвечающая за асинхронную запись сообщений. Для ее реализации создается линейный набор (кэш) буферов. Доступ к каждому буферу кэша защищен отдельным семафором. Первый буфер является выделенным и не используется для записи внешних сообщений. В него заносится на постоянное хранение сообщение о переполнении кэша. При возникновении такого события семафор первого буфера закрывается, регистрируя факт переполнения.
Непосредственно после записи сообщения предпринимается попытка пересылки всех накопленных в кэше данных в кольцевой буфер (FIFO). Запись данных а FIFO не является сигнало- или потоко- безопасной; поэтому она защищена собственным семафором.
Использование кэша в редких случаях может приводить к тому, что сообщения будут записаны в FIFO, а затем и в журнал в последовательности, отличной от очередности их занесения в кэш.

Рис 2. Подчистка FIFO регистратора.

На рисунке 2 показана схема подчистки кольцевого буфера при его заполнении. Когда голова буфера H упирается в его хвост T последний принудительно смещается, освобождая тем самым некоторое число элементов FIFO. При этом самые старые сообщения будут утеряны, но благодаря наличию кэша обеспечивается регистрация самого факта переполнения кольцевого буфера.

1.2 Временная метка события

Для установа временной метки события в примере регистратора используется стандартная C-функция календарного времени с секундным разрешением. Предполагается, что данная функция является сигнало- и потоко- безопасной.
Возможно использование временной метки высокого разрешения ~1 мС. Для ОС Windows см. https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps.
Для микроконтроллера также можно использовать относительную временную метку. Такая метка реализуется как счетчик, который инициализируется при каждом запуске контроллера и инкрементируется прерываниями одного из таймеров.

1.3 Семафорные операции

В программном коде регистратора используются двухфазные семафоры, которые реализованы на основе явных операций с переменными. В контекстно-последовательных приложениях (вложенные прерывания, сигналы, рекурсии) такое решение допускает не атомарные операции с семафорными переменными (инкремент, декремент, присваивание, проверка).
Для многопоточных приложений, когда контекст исполнения задач определяется внешним диспетчером и не является последовательным, использование критических секций для обеспечения атомарности семафорных операций обязательно.
В программе регистратора используются макросы критических секций, которые определены в файле logger_macros.h:
LOGGER_CRITICAL_INIT
Макрос инициализации критических секций программы. Вызывается однократно до первого использования механизма критических секций.
LOGGER_CRITICAL_BEGIN
Макрос входа в критическую секцию.
LOGGER_CRITICAL_END
Макрос выхода из критической секции.
Для контекстно-последовательных приложений эти макросы могут оставаться пустыми.

1.4 Файловая система для вывода журнала

Регистратор поддерживает две модели ведения журнала — с ограниченным числом файлов и без такого ограничения. Первый вариант удобен для встраиваемых приложений, второй — при использовании журнала на ЭВМ общего назначения с достаточным дисковым пространством.
Основное имя файлов журнала определяется параметром LOG_FILE_NAME_CONFIG. При формировании файла осуществляется преобразование его основного имени.
Для режима с ограниченным числом файлов непосредственно перед последним расширением имени файла добавляется численная метка в диапазоне от _001 до _999. При этом каждый файл содержит ограниченное число записей. Номер текущего файла журнала содержится в ASCII формате в файле с численной меткой _000.
Для режима без ограничений новые файлы журнала создаются ежечасно. Непосредственно перед последним расширением имени файла добавляется временн*а*я метка вида _yyyymmdd_hhmmss. Пустые файлы журнала удаляются.

2. Параметры и структуры данных регистратора

2.1 Наименование основных типов данных

boolean	Логическое значение false / true.
integer8	Целое 8 бит со знаком.
unsigned8	Беззнаковое целое 8 бит.
integer16	Целое 16 бит со знаком.
unsigned16	Беззнаковое целое 16 бит.
integer32	Целое 32 бита со знаком.
unsigned32	Беззнаковое целое 32 бита.
integer64	Целое 64 бита со знаком.
unsigned64	Беззнаковое целое 64 бита.
real32	32-х разрядное с плавающей точкой.
real64	64-х разрядное с плавающей точкой.
vis-string	Строка видимых ASCII символов (коды 0 и 20_h..$7E_h$).
octet-string	Байтовая строка (коды 0..255).

2.2 Структуры данных регистратора

struct eventlog {

time_t ts	временн*а*я метка события.
unsigned8 network	номер сети или подсистемы, в которой было порождено событие.
unsigned8 node	номер узла или устройства, в котором было порождено событие; для событий регистратора равен нулю.
unsigned8 cls	класс события.
unsigned8 type	тип события (info, warning, error и т.д.)
int16 code	код события; может быть записан в беззнаковом формате.
int32 info	дополнительная информация о событии; может быть записана в беззнаковом формате.

};

Структура **eventlog** содержит информацию о зарегистрированном событии. Формат структуры определяется задачами приложения.

struct eventcache {

int16 busy	семафор занятия кэша (инкрементный).
unsigned8 capture	флаг захвата кэша и занесения в него данных.
struct eventlog ev	событие регистратора.

};

Структура **eventcache** формирует кэш для размещения событий регистратора. Содержит переменные для организации двухфазных семафоров.

2.3 Основные параметры регистратора

Параметры регистратора определены в заголовочных файлах: __logger_defines.h, __logger_events.h.

- EVENTS_CACHE_SIZE
  Размер кэша регистратора. Один элемент (нулевой) используется для регистрации переполнения кэша.
  Минимальное значение 2, рекомендуемые от 4 до 8.
- EVENTS_FIFO_SIZE
  Размер FIFO регистратора. Размещает (EVENTS_FIFO_SIZE-1) событий.
  Минимальное значение 5, максимальное 64884.
- LOGGER_TIMERUSEC
  Период таймера регистратора в микросекундах. Таймер используется в отладочных целях.
- LOG_FILE_NAME_DEFAULT
  Имя файлов журнала по умолчанию. Используется в случае ошибки открытия конфигурируемого файла журнала. Файлы по умолчанию размещаются в директории программы регистратора.
- LOG_FILE_NAME_CONFIG
  Конфигурируемое имя файлов журнала.
- EVENT_CLASS_*
  Классы событий регистратора.
- EVENT_TYPE_*
  Типы событий регистратора (info, warning, error и т.д.).
- EVENT_CODE_*
  Коды отдельных событий. Для событий различных классов могут иметь одинаковые значения.

3. Модули регистратора

Размещение модулей приведено относительно «корневой» директории регистратора.

logger директория. Основные модули регистратора.

- logger_filename.c – выполняет преобразование имен файлов.
- logger_globals.c – определения внешних (глобальных) переменных и структур данных (при необходимости).
- logger_logfile.c – формирование сообщений о событиях в описательном виде.
- logger_logger.c – основной модуль регистратора.
- logger_output.c – вывод накопленных в FIFO событий в сеть. файл и т.д.

include директория. Модули определений и прототипов.

- __logger_defines.h – определение параметров регистратора.
- __logger_events.h – определение параметров параметров событий.
- logger_defunc.h – модуль прототипов функций.
- logger_genhead.h – основной модуль заголовков и подключений.
- logger_globals.h – внешние (глобальные) переменные (при необходимости).
- logger_header.h – базовый заголовочный модуль.
- logger_macros.h – определение макросов регистратора.
- logger_structures.h – определение структур данных.
- logger_typedefs.h – определение типов данных.

Корневая директория.

- logger_system_winlib.c – таймер и поддержка критических секций регистратора для ОС Windows.
- __logger_main.c – содержит запускаемую на выполнение функцию main(...) и главный цикл программы.

4. Функции регистратора

4.1 Функции прикладного интерфейса (API)

void log_event(struct eventlog *ev);

Основная функция прикладного интерфейса регистратора. Размещает событие в кэше и осуществляет его вывод в FIFO. Функция является сигнало- и потоко- безопасной.

Параметры:

- ***ev** – зарегистрированное событие.

void logger_test_event(unsigned8 cls, unsigned8 type, int16 code, int32 info);

Функция прикладного интерфейса для записи событий тест-мастера.

Параметры:

- **cls** – класс зарегистрированного события.
- **type** – тип события (info, warning, error и т.д.).
- **code** – код события.
- **info** – дополнительная информация о событии.

void push_logger(void);

Осуществляет извлечение и переправку накопленных в FIFO событий в выходной поток (сеть, файл и др). Вызывается из главного цикла программы.

void init_logger(void);

Инициализирует регистратор. Вызывается однократно при входе в программу.

void close_logger(void);

Закрывает регистратор. Вызывается однократно при выходе из программы.

4.2 Другие видимые функции регистратора

void logger_sleep(int32 microseconds);

Функция временн*о*й задержки.

Параметры:

- **microseconds** – временн*а*я задержка в микросекундах. Точное время задержки определяется разрешением соответствующего таймера системы. Любое положительное значение аргумента функции обеспечивает отличную от нуля задержку.

void logger_init_system_timer(void (*handler)(void));

Инициализация таймера регистратора. Windows таймер исполняется как отдельный поток (thread) приложения, а метод работы диспетчера ОС не гарантирует непрерывного выполнения этого потока.

Параметры:

- **handler** – функция обработчика таймера, имеет прототип: void logger_timer(void).

void logger_cancel_system_timer(void);

Прекращает либо завершает работу таймера.

void init_critical(void);
void enter_critical(void);
void leave_critical(void);

Функции обслуживания критических секций: инициализация, вход и выход. Служат для обеспечения атомарности семафорных операций и непрерывности сегментов кода при использовании регистратора в многопоточной среде. Такая ситуация возникает, например,

при использовании операционной системы Windows. Функции должны обеспечивать многократный (вложенный) вход и выход из критической секции. Функции внедряются в код библиотеки с помощью макросов LOGGER_CRITICAL_INIT, LOGGER_CRITICAL_BEGIN и LOGGER_CRITICAL_END, определенных в модуле logger_macros.h. Для однопоточных контекстно-последовательных приложений (вложенные прерывания, сигналы) код регистратора допускает не атомарные операции с семафорными переменными (инкремент, декремент, присваивание, проверка). Таким образом, эти макросы могут оставаться пустыми.

4.3 Внутренние функции регистратора

static void flush_events_cache(void)

Управляет пересылкой в FIFO событий регистратора, которые накоплены в кэше. Функция является сигнало- и потоко- безопасной. Для гарантированного вывода всех данных из кэша используется контроль повторных вызовов данной функции, при обнаружении которых пересылка автоматически возобновляется.

static void flush_cache_to_fifo(void)

Осуществляет непосредственную пересылку данных из кэша в FIFO. При заполнении FIFO производит его подчистку, удаляя самые старые сообщения. Функция не является сигнало- или потоко- безопасной и должна вызываться исключительно из управляющей функции flush_events_cache().

5. Вывод данных регистратора

Вывод данных регистратора осуществляется посредством функций:

void write_output(struct eventlog *ev);

Осуществляет выгрузку одной записи события из FIFO регистратора. Вызывается из функции push_logger() – из главного цикла программы.

Параметры:

***ev** – зарегистрированное событие.

int32 pre_process_output(void);

Осуществляет пред-обработку до выгрузки записей из FIFO регистратора. Вызывается из функции push_logger().

Возвращаемые значения:

- > 0 – максимальное число записей, которое может быть выгружено из FIFO.
- <= 0 – ошибка.

void post_process_output(void);

Осуществляет пост-обработку после выгрузки записей из FIFO регистратора. Вызывается из функции push_logger().

void close_output(void);

Закрывает подсистему вывода событий регистратора. Вызывается автоматически при закрытии регистратора.

void init_output(void);

Инициализирует подсистему вывода событий регистратора. Вызывается автоматически при инициализации регистратора.

5.1 Вывод журнала в текстовый файл

void transform_file_name(char *fname, char *initfn);

Преобразует имя файла таким образом, чтобы учесть размещение исполняемой программы. Для этого у операционной системы запрашивается информация о командной строке, с использованием которой была запущена программа. В итоге обеспечивается возможность формирования имен файлов с учетом их размещения относительно исполняемой программы.

Параметры:

- ***fname** – строка имени преобразованного файла.
- ***initfn** – строка имени исходного файла.

void time_stamp_file_name(char *fname, char *initfn, time_t ts);

Преобразует имя файла, добавляя к нему временную метку вида _yyyymmdd_hhmmss. Метка вставляется непосредственно до последнего расширения имени файла.

Параметры:

- ***fname** – строка имени преобразованного файла.
- ***initfn** – строка имени исходного файла.
- **ts** – временная метка.

void numerate_file_name(char *fname, char *initfn, unsigned16 num);

Преобразует имя файла, добавляя к нему номерную метку вида _* в диапазоне от _000 до _999. Метка вставляется непосредственно до последнего расширения имени файла.

Параметры:

- ***fname** – строка имени преобразованного файла.
- ***initfn** – строка имени исходного файла.
- **num** – численный номер метки в диапазоне от 000 до 999.

void write_event_to_file(struct eventlog *ev, FILE *log);

Разборщик событий регистратора для их занесения в файлы журнала в текстовом виде. Вызывается из функции write_output().

Параметры:

- ***ev** – зарегистрированное событие.
- ***log** – открытый файл журнала.

int32 pre_process_output(void);

Поддерживает систему файлов регистратора. Ежечасно создает новый файл событий. При отсутствии зарегистрированных событий удаляет текущий файл.

void post_process_output(void);

Производит дозапись всех оставшихся в файловом буфере данных.