C++ Craft: #7

Поиск утечек памяти

Подсистемы серверных приложений

system_utilities

Visual Studio Way

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>

int main()
{
	_CrtSetDbgFlag ( _CRTDBG_ALLOC_MEM_DF |
		_CRTDBG_LEAK_CHECK_DF );
	_CrtDumpMemoryLeaks();

	// actions
}
```

http://msdn.microsoft.com/en-us/library/x98tx3cf%28v=vs.90%29.aspx

Visual Studio Std Output

{79} normal block at 0x00000000000D2860, 14 bytes long.

Data: < > CD CD CD CD CD CD CD CD CD CD CD CD CD CD

http://habrahabr.ru/post/82514/

Visual Leak Detector

- Плюсы
  - Те же Возможности отображения утечек памяти
  - Удобная настройка вывода в файл
  - Подключать и использовать проще чем стандартные средства
- Минусы
  - Работает только в MSVC 2008+
  - Требует компиляции
  - Необходимо подключать в каждый модуль с раздельной точкой входа (exe, dll).

https://vld.codeplex.com/

VLD

Чтобы подключить:

1) #include «vld.h»

2*) Properties -> Linker -> Input -> Additional Dependencies += vld.lib

3*) Разместить dll в %PATH% (или рядом с exe файлом.

4*) Подключить??? Конфигурационный файл

Boost Test Framework

- Плюсы
  - «Из коробки»
- Минусы
  - Только в MSVC*

Valgrind

http://valgrind.org/

Valgrind

- Memcheck – обнаружение утечек памяти
- Cachegrind – анализ выполнения кода
- Massif – анализ выделения памяти различными частями программы
- Helgrind – анализ многопоточного кода

- Есть ещё

http://alexott.net/ru/linux/valgrind/Valgrind.html

Usage

```
$ valgrind ./bin_32/Debug/valgrinde

==26372== LEAK SUMMARY:
==26372==    definitely lost: 70 bytes in 5 blocks
==26372==    indirectly lost: 0 bytes in 0 blocks
==26372==      possibly lost: 0 bytes in 0 blocks
==26372==    still reachable: 0 bytes in 0 blocks
==26372==         suppressed: 0 bytes in 0 blocks
```

Usage with --leak-check=full

valgrind --tool=memcheck --leak-check=full ./bin_32/Debug/valgrinded

```
==26393== 14 bytes in 1 blocks are definitely lost in loss record 1 of 5
==26393==    at 0x402B24C: operator new[](unsigned int) (vg_replace_malloc.c:378)
==26393==    by 0x80489C9: memory_leak_examples::bad_string::bad_string()
   (bad_string.cpp:5)
==26393==    by 0x8048901: main (main.cpp:7)
==26393==
==26393== 14 bytes in 1 blocks are definitely lost in loss record 2 of 5
==26393==    at 0x402B24C: operator new[](unsigned int) (vg_replace_malloc.c:378)
==26393==    by 0x80489C9: memory_leak_examples::bad_string::bad_string()
   (bad_string.cpp:5)
==26393==    by 0x804890D: main (main.cpp:8)
```

?

Quiz

```
#include <iostream>

int main()
{
	int a = 5;
	float b;
	std::cout << sizeof(++a + b);
	std::cout << a << std::endl;
	return 0;
}
```

Системы конфигурирования

- Механизм настройки приложения для определённых условий

- boost::program_options
- Config4Cpp ?

https://github.com/config4star/config4cpp

Требования к системам конфигурирования

- Возможность ввода большого количества параметров
- Возможность доступа из всех необходимых частей приложения (без копирования)
- Возможность создания “namespace”
- Возможность разбиения конфигурационных файлов и автоматического включения

Примеры конфигурационных файлов

```
foo
{
    timeout = "5 seconds";
    log
    {
            level = "2";
            dir = "/tmp";
    }
    colour = "green";
    int_list = ["1", "2", "3"];
    temperature = "29 C";
}
```

Примеры конфигурационных файлов

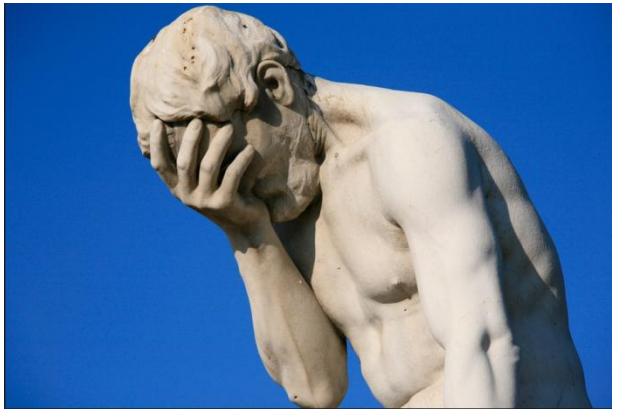
```
CtaTradeLines = t1

CtaTradeLine.t1.multicast_addr = 233.200.79.128
CtaTradeLine.t1.port = 62128
CtaTradeLine.t1.log_file = _cta_in_t01.log

include <other_file_name>.ini
```

Системы журналирования

- Механизм поиска ошибок для работающих систем
- Механизм отладки приложений в условиях отсутствия компилятора



- Syslog-ng
- Boost::Log
- log4cpp

Требования к системам журналирования

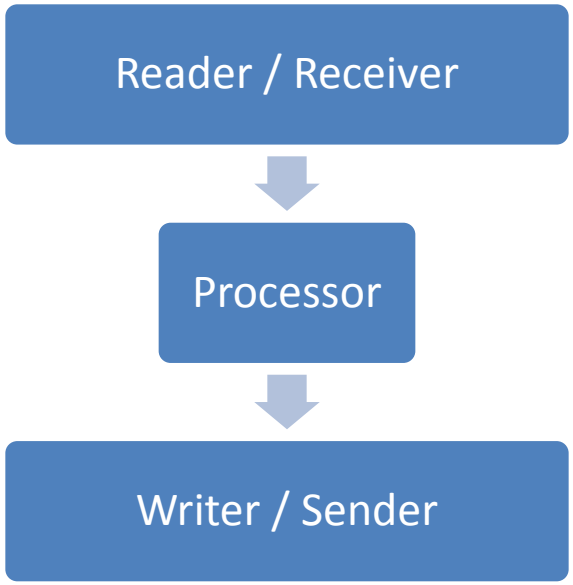
- Корректная выдача требуемых сообщений разделённых на типы
- Выдача времени генерации по необходимости
- Параметризируемый вывод
- Контроль размера файлов журналов на hdd (или где ещё)
- Возможность контроля влияния на приложение

Примеры работы

```
Algorithm: artificial_stock_trading_algorithm
   (ArbReflectorAlgo#0.1.6.5 - Pursuit the market) loaded.
System started
Threads created and started

[2013-Apr-17 14:18:59.228022:NOTE   ]: used default:
   filtering_available = no
[2013-Apr-17 14:18:59.229022:NOTE   ]: used default:
   global.big_image_drawing.dir =
   D:/usr/_environments/bs_stat/results/images/
[2013-Apr-17 14:18:59.229022:NOTE   ]: used default:
   global.instrument.parameters.file.path.postfix =
   \InstInfo.txt
```

Системы передачи информации между модулями



Callback mechanism

Interprocess communication Network Thread safe queue

Thread Safe Queue

- Передача данных между модулями системы
- Реализация различных механизмов остановки приложения
- Контроль отсутствия утечек памяти при остановке приложения
- Возможность ожидания до появления новых данных
- Возможность контроля и ограничения роста очереди (защита от перегрузок)

Thread Safe Queue Variants

- Boost::lockfree
- Your implementation

Thread Safe Queue Example

```
// ---------- processing thread
while (true) {
	size_t* s = mq_.wait_pop();
	if (!s) break;
	// actions (potential memory leak)
	delete s;
}
// ---------- reader thread
size_t * s = new size_t( rand() % 100000 );
bool res = mq_.push( s );
if (!res) {
	delete s;
	// exit
}
// ---------- control thread
mq_.stop_processing();
```

system_utilities

https://github.com/sidorovis/system_utilities

system_utilities

- Loggers:
  - file_logger
  - ts_logger
  - limited_file_logger
  - …
- Property_reader
- System_processor
- Ts_queue
- …

?