Yet Another Fast Log

yafl is yet another complete technique to fastly: write, run and read log messages

Background

- Applications either do their job or log messages about the job, but not both
- Natural logging mechanism
- Easy to log a message
- Minimum constraints on the message itself
- Good logging mechanism
- Logs Everything
- Fast
- Usable
- Works for the developer and not vise versa

Applicability

- Embedded
- Critical mission areas
- No strings as part of the logic
- Where logs are part of the defined output of the program

Naturally Written Log Messages @ ${SOURCE}/main.cpp

```
LOG_DEMO_INFO( "messages per second: ", messegesPerSecond, "; calculation time (us):", fastCount, "; total messages:", sl.getCallCount() );
LOG_DEMO_DEBUG( "this text is mapped to unsigned long" );
LOG_DEMO_WARNING( "variables must be converted to unsigned long too:", v1, v2, v3 );
```

Preprocess

External Preprocess

- Generates "binary" source files
- Generates map file
- Adds "metadata"
- Keeps the generated code in the correct line (!)

Map File

- Contains information about the source file itself
- Maps between unsigned long to a human readable text

Map File @ ${BINARY}/main.cpp.map

```
'0xe642399bcaccbb09':
["/home/ran/work/yafl/yafl/logger/demo/main.cpp",{
'0xbeef000000000000':"Demo",
'0xbeef000000000001':"messages per second: ",
'0xbeef000000000002':"; calculation time (us):",
'0xbeef000000000003':"; total messages:",
'0xbeef000000000004':"Demo",
'0xbeef000000000005':
"this text is mapped to unsigned long" ,
'0xbeef000000000006':"Demo",
'0xbeef000000000007':
"variables must be converted to long too:",}],
```

Generated Source Code @ ${BINARY}/main.cpp

```
yafl::log::g_logQ.push(yafl::log::Log(0xe642399bcaccbb09,__LINE__,0x00000000ababab03,pthread_self(),
0xbeef000000000000,0xbeef000000000001,messegesPerSecond,0xbeef000000000002,fastCount,0xbeef000000000003,
sl.getCallCount() ,0xcdcdcdcdcdcdcdcd));
yafl::log::g_logQ.push(yafl::log::Log(0xe642399bcaccbb09,__LINE__,0x00000000ababab00,pthread_self(),
0xbeef000000000004,0xbeef000000000005,0xcdcdcdcdcdcdcdcd));
yafl::log::g_logQ.push(yafl::log::Log(0xe642399bcaccbb09,__LINE__,0x00000000ababab02,pthread_self(),
0xbeef000000000006,0xbeef000000000007,v1,v2,v3 ,0xcdcdcdcdcdcdcdcd))
```

Compile

Compile

- Errors points to the generated code and not to the original code
- Errors in the log message itself are strange (but easy to get used to)

Executable

Runtime

- Messages are constexpr copied
- A dedicated thread buffers the messages
- Messages are wittren to a file in batches
- Result: extremely low footprint on the working thread

Runtime

Known Issues (bugs)

- Commas cannot be added to string messages (need to write better m4 code)

Binary Log File (od -v -A n -t x8 -w200 fast.log)

```
0000016c2dfaeac5 4ea57b2b5505151a 000000000000002e
000000000ababab01 00007f0d67874740
beef000000000003 beef000000000004
0000000000000001 beef000000000005 cdcdcdcdcdcdcdcd  <snip>
```

Read

Reader

- Currently a python script that just translates the binary to text
- Can be a compiled code.
- May have few readers types: see improvements

Reader

Output

Text Log File

```
26/07/2019 14:11:55.566|/home/ran/work/yafl/yafl/logger/demo/main.cpp|38|INFO|0x00007f0d67874740|Demo|messages per second:  200000; calculation time (us): 5; total messages: 1############
26/07/2019 14:11:55.566|/home/ran/work/yafl/yafl/logger/demo/main.cpp|39|DEBUG|0x00007f0d67874740|Demo|this text is mapped to unsigned long ##################
26/07/2019 14:11:55.566|/home/ran/work/yafl/yafl/logger/demo/main.cpp|40|WARNING|0x00007f0d67874740|Demo|variables must be converted to long too: 100200300###############
```

Benchmark

Runtime:
- Results are mostly affected by I/O operations
- 4 CPUs
- Intel(R) Core(TM) i5-7600K CPU @ 3.80GHz
- Ubuntu x86_64
- 8Gb Ram
- Without I/O: ~11M messages per second
- With I/O: ~5M messages per second

Compile time
- Not tested, surly can be improved

Limitations

- Dynamic strings cannot be logged
- All types must be converted to long
- No "recursive" logs:

cout << a;
ostream& operator << (ostream& o, const A& a) { o << a.b;}
ostream& operator << (ostream& o, const B& b) { o << b.c;}
etc.

Cmake

- Cmake has a crucial role in binding it all together.
- It keeps the usual process untouched:

code -> compile -> run -> analyze
- each file is a custom project
- for each file a '<file>.gen.sh' is created to generate the code and map
- the .gen.sh file is executed as part of the build
- dependencies are enforced

cppcon the C++ conference

Ran Regev

regev.ran@gmail.com
@ran_regev
linkedin.com/in/ran-regev-8b57386
+972-50-3891316
https://github.com/regevran/yafl.git

Improvements

- Faster Runtime
- Faster Pre Compile
- Faster Post Processing
- yafl-grep
- yafl-tail
- yafl-etc
- IDE integration (compiled code is not the written code)
- Enable only in partial parts of the code (mix mode with other logs)

CMAKE,MAKE

Abstruct

Many programs reached a point that the messaging system itself requires runtime resources such that **the program either does its work *or* reports about it**, but not both.

YAFL has a github repository that as far as we can see it might be used as **a starting point, a kick-off for users that need YAFL** solution. It is definitely not out-of-the-box library to download and use

M4 Preprocess Code

```
29  ### -- log levels --
30  m4_define(YAFL_DEBUG, 0x00000000ababab00)
31  m4_define(YAFL_ERROR, 0x00000000ababab01)
32  m4_define(YAFL_WARNING, 0x00000000ababab02)
33  m4_define(YAFL_INFO, 0x00000000ababab03)
```

```
101  ### -- _map1 --
102  ### map1 stand for: _map one_ pair
103  ### maps a number to a string.
104  ### outputs the map to the map_area
105  ### and the number to the code_area
106  ### e.g.:
107  ### _map1( 0x4534, "my name is ran" )
108  ### => map-area:  0xbeef4534 - "my name is ran"
109  ### => code-area: 0xbeef4534
110
111  m4_define( _map1,
112          [*m4_ifelse(
113              m4_regexp($2,".*"), 0,
114              [*_map_area_dnl('_beef($1)':$2)*][*_map_comma*]
115              [*_code_area_dnl(_beef($1))*]
116              [*m4_define([*THE_HEX*],_next_hex($1))*],m4_dnl
117              [*_code_area_dnl($2)*]
118          )*]m4_dnl
119  )m4_dnl
```

```
162  ### -- YAFL_LOG --
163  m4_define( YAFL_LOG,
164          [*_code_area_dnl(_call_prefix)*]m4_dnl                          prefix
165          [*_code_area_dnl(0x[**]COMPILED_FILE)*][*_code_comma*]m4_dnl    file
166          [*_code_area_dnl(_line)*][*_code_comma*]m4_dnl                  line
167          [*_code_area_dnl($1)*][*_code_comma*]m4_dnl                     debug level
168          [*_code_area_dnl(_working_thread)*][*_code_comma*]m4_dnl        thread
169          [*_module(THE_HEX,[*$2*])*]m4_dnl                               module
170          [*m4_ifelse(
171              m4_eval($# > 2), YAFL_TRUE,
172              [*_code_comma*]m4_dnl
173          )*]m4_dnl
174          [*_mapper(THE_HEX,m4_shift(m4_shift($@)))*][*_code_comma*]m4_dnl  all the rest
175          [*_code_area_dnl(_call_suffix)*]m4_dnl                          suffix
176          [*_orig_area*]m4_dnl
177  )m4_dnl
```

Cmake Code

```
cat -n ../custom.build/log_precompile.cmake
35      add_custom_command(
36          OUTPUT  ${generator_sh_file}
37          COMMAND
38                  ${CMAKE_COMMAND}
39                  -DGENERATOR_IN_FILE=${YAFL_SOURCE_DIR}/m4/generate_code_and_map.sh.in
40                  -DGENERATOR_OUT_FILE=${generator_sh_file}
41                  -DSOURCE_FILE=${source_file}
42                  -DTARGET_FILE=${target_file}
43                  -DMAP_TARGET_FILE=${map_target_file}
44                  -DYAFL_PREPROCESSOR_DIR=${YAFL_SOURCE_DIR}/m4/
45                  -P ${YAFL_SOURCE_DIR}/custom.build/configure_generator.cmake
46
47          DEPENDS ${source_file}
48      )
49
50      add_custom_command(
51          OUTPUT                  ${target_file}
52          COMMAND                 ${generator_sh_file}
53          MAIN_DEPENDENCY         ${generator_sh_file}
54          DEPENDS                 ${source_file}
55      )
56
57      string( REPLACE "/" "_" generator_target generator-${target}-${bare_file})
58
59      add_custom_target(
60          ${generator_target}
61          DEPENDS     ${target_file}
62      )
```

```
Scanning dependencies of target generator-logger_m4log-m4log_defs.cpp
[ 68%] Generating m4log_defs.cpp.gen.sh
[ 69%] Generating m4log_defs.cpp
[ 69%] Built target generator-logger_m4log-m4log_defs.cpp

[ 79%] Building CXX object
logger/m4log/CMakeFiles/logger_m4log.dir/m4log_defs.cpp.o
```

```
cat -n ../m4/generate_code_and_map.sh.in
1  #!/bin/bash
2
3  # expecting the following variables to be set by the caller to cmake-configure
4  # SOURCE_FILE               ==      ${SOURCE_FILE}
5  # TARGET_FILE               ==      ${TARGET_FILE}
6  # MAP_TARGET_FILE           ==      ${MAP_TARGET_FILE}
7  # YAFL_PREPROCESSOR_DIR     ==      ${YAFL_PREPROCESSOR_DIR}
8
9  fileId=$(md5sum ${SOURCE_FILE} | head -c16)
10 echo -n "'0x$fileId':[\"${SOURCE_FILE}\",(" >${MAP_TARGET_FILE}
11
12 varForWait=$(\
13     /bin/m4 \
14     --prefix-builtins \
15     -DCOMPILED_FILE=$fileId \
16     ${YAFL_PREPROCESSOR_DIR}/yaflpp.m4 \
17     ${YAFL_PREPROCESSOR_DIR}/modules.m4 \
18     ${SOURCE_FILE} \
19     | \
20     1>/dev/null tee \
21     >(${YAFL_PREPROCESSOR_DIR}/code_area.sed >${TARGET_FILE}) \
22     >(${YAFL_PREPROCESSOR_DIR}/map_area.sed  >>${MAP_TARGET_FILE}))
23
24 wait
25
26 echo ")]," >>${MAP_TARGET_FILE}
```

There are
many log solutions

YAFL is
yet another one