CTF and BTF debug formats in the GNU Toolchain: status update and what’s next

Indu Bhagat  indu.bhagat@oracle.com
David Faust  david.faust@oracle.com
Wei-min Pan  weimin.pan@oracle.com
• Introduction: CTF/BTF debug formats
• Updates since GNU Tools track 2020
• CTF/BTF debug info generation in GCC
• What's next
CTF/BTF Debug Formats

• Two distinct debug formats to convey type information
  – Compact C Type Format (CTF) describes C types (mid 2000s)
    • Solaris Kernel → Linux
    • Current version = V3
  – BPF Type Format (BTF) is inspired by Solaris CTF. With a focus on BPF programs/kernel (first LLVM/kernel patches circa 2018)
    • Current version = V1

• Remarkable similarities owing to a common ancestor
  – But they are binary incompatible, distinct formats
## CTF/BTF Debug Formats

Distinct formats with independent evolution history and use-cases

| Most recent additions | CTF_K_SLICES for representing bitfields | BTF_KIND_FLOAT for fp types  
| BTF_KIND_TAG for attributes |
|------------------------|----------------------------------------|---------------------------|
| Support for multiple CUs | CTF has representation of parent-child dictionaries (archives) | BTF does not have such a representation |
| Library support | libctf (binutils >=2.36) for ld, gdb | [No linker support] libbpf (kernel) does BPF program loading, performs relocations, ... |
| Miscellaneous | No src location information | BTF/CO-RE (.BTF.ext), src location information, relocations, ... |
Updates since GNU Tools track 2020

- CTF and BTF debug formats are now fully supported
  - GCC: -gctf, -gbtf generates the .ctf/.BTF section
  - Binutils ld, objdump: --ctf=.ctf
    - CTF Type de-duplication contributed in 2020.
  - GDB: Support for CTF Archives
- CTF Spec and mailing lists @ ctfstd.org
- Is it ready for uptake?
  - Yes! Please try it out and report issues on bugzilla or mailing lists
Implementation Notes
Implementation in GCC

- (dbx_debug_hooks) dbxout.c
- (xcoff_debug_hooks) xcoffout.c
- (vms_debug_hooks) vmsdbgout.c
- (dwarf2_debug_hooks) dwarf2out.c

CTF Container via dwarf2ctf

- CTF
- BTF
- DWARF
CTF/BTF debug info generation in GCC

DWARF generation and emission
(dwarf2out.h / dwarf2out.c)

- ctf_debug_init()
- ctf_debug_do_die()
- ctf_debug_preprocess()
- ctf_debug_early_finish()
- ctf_debug_finish()
- ctf2out_early_finish()
- ctf2out_finish()

GCC Internal DWARF → Internal CTF
(dwarf2ctf.h / dwarf2ctf.c)

- ctf_add_function()
- ctf_add_array()
- ctf_add_sou()

CTF Container
(ctfc.h / ctf.c)

CTF context

CTF Emission
(ctfout.c)

BTF Emission
(btfout.c)

- ctf_init_preprocess()
- ctf_output()
- ctf_init_preprocess()
- ctf_output()
BTF support in GCC

- BTF generation for any target via -gbtf
  - No debug levels are necessary
- For BTF generation and emission, the CTF container is:
  - Post-processed after initialization
    - BTF encodes bitfields differently
    - Unrepresented types are removed
  - Pre-processed before emission
    - Add BTF_KIND_FUNC and BTF_KIND_DATASEC records
- For BPF backend, -gbtf generates BTF/CO-RE
CTF Support in GDB

- Implements a CTF reader, using libctf, to read and set CTF data
  - Skips reading CTF data if DWARF debugging info exists
- Enables GDB commands such as ptype, print, and whatis to support CTF types
- Supports single compilation unit and CTF Archives
- Next task:
  - Support for CTF V4 (backtraces and other CTF format changes)
What's Next
Next Steps

• Continue to support CTF/BTF in GNU Toolchain
• Towards CTF V4
  – Inviting community discussion and participation on the public mailing list at ctfstd.org
Future CTF Work (GCC)

- Testing of interaction between debug flags which affect the generation of DWARF DIEs and -gctf / -gbtf
  - `[debug_info_level] -gLEVEL OR -gtoggle`
  - */ CTF/BTF generation feeds off DWARF dies. For optimal CTF/BTF, switch debug info level to 2. If off or at level 1, set it to level 2, but if already at level 3, don't lower it. */

<table>
<thead>
<tr>
<th>Flag Configuration</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g0 -gctf -gtoggle</code></td>
<td>0 → 2 → 0</td>
<td>emits no DWARF, emits no CTF</td>
</tr>
<tr>
<td><code>-gctf -g0 -gtoggle</code></td>
<td>2 → 0 → 2</td>
<td>emits DWARF, emits CTF</td>
</tr>
</tbody>
</table>

- Need to differentiate between **user-specified** debug info level (output/emission) vs **internal** GCC debug info level (debug info generation)
Future CTF work (GCC)

- Modularize / Refactor dwarf2out.c / dwarf2out.h
  - Usage: DWARF die creation, add and get attribute APIs
  - Functionality: Split off a dwarf2cfi.h
- Prepare for CTF V4
  - Make the CTF Container (CTFC) version aware
  - Command line options -gctf-version=<NUMBER> to choose CTF version
CTF version 4 – Backtraces

Requirements

• Generate Backtraces with the following requirements:
  - Exact at each instruction boundary
  - With original value of the arguments at the point of function call
  - Keep it simple and compact. No complex expression encoding, no location lists, no stack machine
Each ABI defines the parameter passing rules
- #1: Assign storage class to the argument
  - High-level language type $\rightarrow$ machine type
- #2: Assign reg/stack location to each argument, given its machine type

Proposal
- Debug format specifies #1
- Backtrace client does #2
CTF version 4 – Backtraces

• The natural location of the argument is not encoded explicitly in the format. It is inferred from position and class of the argument and the ABI (by the client)
  – [AMD64] INTEGER / SSE / SSEUP / X87 / X87UP / … / MEMORY
  – [AARCH64] INTEGER / SIMD / FP / SCALABLE VECTOR / SCALABLE PREDICATE / MEMORY
### Examples for argument passing

<table>
<thead>
<tr>
<th>// all integers</th>
<th>Representation [AMD64]</th>
</tr>
</thead>
</table>
| func1 (int a, int b, int c, int d, int e, int f); | a in RDI, b in RSI, c in RDX, d in RCX, e in R8, f in R9  
Storage class = \[ I I I I I I \]
Implicit loc info = \[ \text{REG} \text{REG} \text{REG} \text{REG} \text{REG} \text{REG} \text{REG} \] |

<table>
<thead>
<tr>
<th>// small aggregates</th>
<th></th>
</tr>
</thead>
</table>
| typedef struct structparm {  
   int a, b; double d; } structparm;  
func2 (int a, structparm s); | a in RDI, s.a and s.b in RSI, s.d in XMM0  
Storage class = \[ I 2 \{ I \text{NO\_CLASS FP} \} \]
Implicit loc info = \[ \text{REG} 2 \{ \text{REG} - \text{REG} \} \] |

<table>
<thead>
<tr>
<th>// large aggregates</th>
<th></th>
</tr>
</thead>
</table>
| typedef struct { float lg1[10]; } structlarge;  
func7 (__int128 a, structlarge b); | // large is defined as any composite struct with more
than 8 eightbytes in size.  
a in RDI+RSI, b on stack  
Storage class = \[ 2 \{ I I \} 10 \{ \text{MEM} \} \]
Implicit loc info = \[ 2 \{ \text{REG REG} \} 10 \{ \text{MEM} \} \] |
CTF version 4 – Backtraces

[Phase 1]

- **[Phase 1]** Get the original value of the argument from its natural location if it has not been clobbered
- Two sections with relevant information
  - .ctf section will contain the Callsite and Callsite parameter records
  - A new section (.ctf_frame) will contain the unwinding info
    - [Callsite Index] Given a PC, get ref to the CTF callsite record
    - [CFA Index] Given a PC, get the CFA
Each object file contains a .ctf and .ctf_frame section which the linker will merge/de-dup if necessary.

The client needs an ABI specific method of deciphering the exact register/stack location of the argument given its storage class.
Thanks!

Visit: ctfstd.org
Email:
nick.alcock@oracle.com (Binutils, CTF)
indu.bhagat@oracle.com (GCC, CTF/BTF)
david.faust@oracle.com (GCC, BPF, BTF/CO-RE)
jose.marchesi@oracle.com (GCC, BPF, CTF/BTF, GNU Poke)
weimin.pan@oracle.com (GDB, CTF)