accu 2023

LINUX DEBUGINFO FORMATS

GREG LAW

Abstract

Many different Linux debugging tools are available - as well as the traditional debuggers (GDB, LLDB) we have checkers (Valgrind, the sanitizers), tracing tools (strace, ltrace), time-travel debuggers (rr, UDB). They all rely on debug info to map from the executable back to the source-code. Most of us know to pass the -g option to gcc to generate debuggable binaries, but there is much more to it than that.

This talk covers what exactly is in debug info, the different compiler options to control its generation, and the different kind of object files and why you might want them (e.g. split dwarf files for quicker loading). We also introduce ways to manage this information, including the new debuginfod service.

undo

Abstract

Many different Linux debugging tools are available - as well as the traditional debuggers (GDB, LLDB) we have checkers (Valgrind, the sanitizers), tracing tools (**strace**, ltrace), time-travel debuggers (rr, UDB). They all rely on debug info to map from the executable back to the source-code. Most of us know to pass the -g option to gcc to generate debuggable binaries, but there is much more to it than that.

This talk covers what exactly is in debug info, the different compiler options to control its generation, and the different kind of object files and why you might want them (e.g. split dwarf files for quicker loading). We also introduce ways to manage this information, including the new debuginfod service.

undo

Abstract

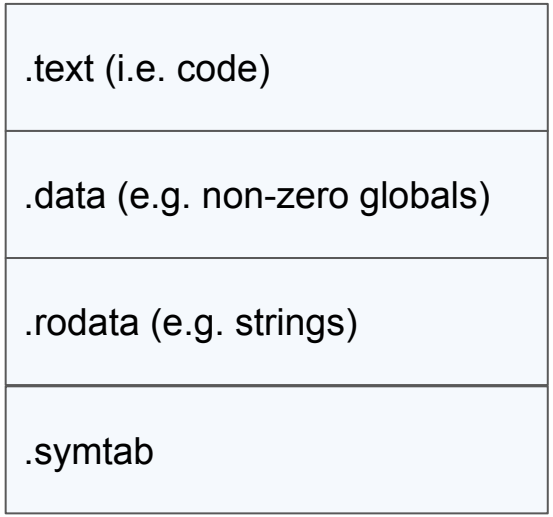
Many different Linux debugging tools are available - as well as the traditional debuggers (GDB, LLDB) we have checkers (Valgrind, the sanitizers), tracing tools (**strace**, ltrace), time-travel debuggers (rr, UDB). They all rely on debug info to map from the executable back to the source-code. Most of us know to pass the -g option to gcc to generate debuggable binaries, but there is much more to it than that.

This talk covers what exactly is in debug info, the different compiler options to control its generation, and the different kind of object files and why you might want them (e.g. split dwarf files for quicker **loading**). We also introduce ways to manage this information, including the new debuginfod service.

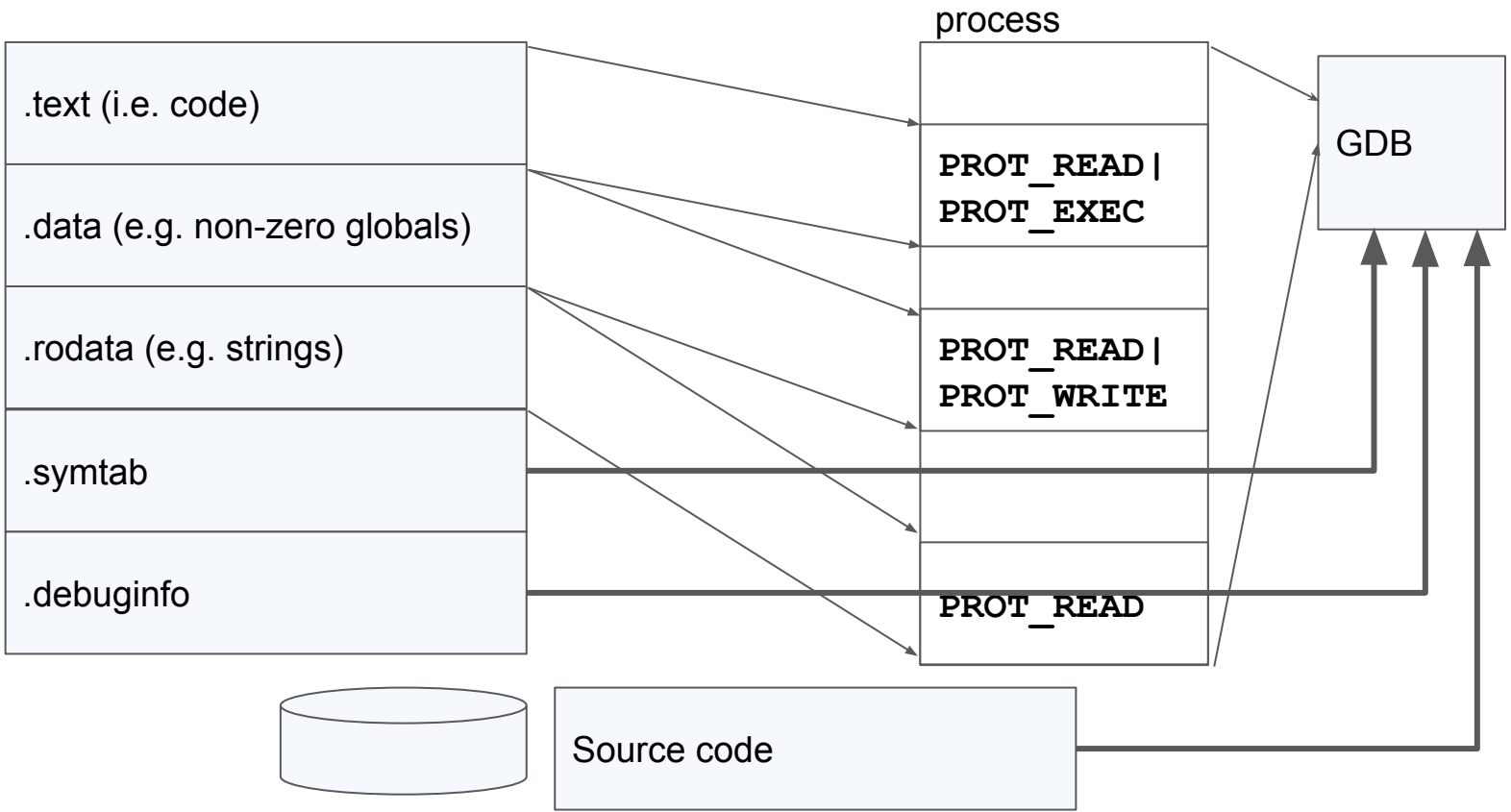
undo

What does -g mean?

```
gcc hello.c      => a.out
```



`gcc -g hello.c` => `a.out`



undo

readelf & addr2line

- Symbol tables and relocations
  - .rel.dyn - Dynamic relocations, pairs of integers
  - .rel.plt - Same but for PLT (Program Linkage Table) (DSO linkage)
  - .got - Global Offset Table (another way of doing global relocation)

BSS

Zero-initialised read-write data.

“Block Started By Symbol”

Note the NOBITS type.

What about read-only zero-initialised data?

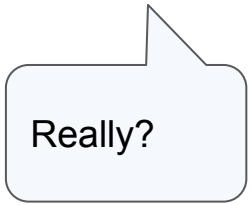
Debug info is not free

- `-g` doesn't impact the generated code - at all.
- `-g` doesn't increase the runtime footprint of your program (much).
- `-g` does impact the size of your binaries - a LOT.
- `-g` can increase your compile and link times.

Wait, link times?

Linker needs to apply relocations to all translation units.

This means the linker needs to parse all debug info of all translation units.



Split DWARF to the rescue

-gsplit-dwarf means:

In the resulting .dwo file, all debug info related to:

- Types, classes
- Identifiers

And in the .o, just

- Anything relative to a PC address.

DWP files

dwp -e EXE

Good luck!

undo


IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

undo

“Debug symbols” vs “debug info”

Debug info

So many utilities

`readelf` `eu-readelf`

`objdump` `eu-objdump`

`dwarf-dump`

BFD

debuginfod

debuginfod serves debug information over HTTP

(a bit like Microsoft Symbol Server)

```
sudo apt install debuginfod
debuginfod
DEBUGINFOD_URLS=localhost:8002 gdb a.out
```

debuginfod servers

https://debuginfod.elfutils.org/

Ubuntu, Debian, OpenSUSE and CentOS run debuginfod servers.

Client support in GDB, Valgrind, SystemTap

Backtrace

```
0000119d <foo>:
   119d:       55              push   %ebp
   119e:       89 e5           mov    %esp,%ebp

          ...

   11ab:       5d              pop    %ebp
   11ac:       c3              ret
```

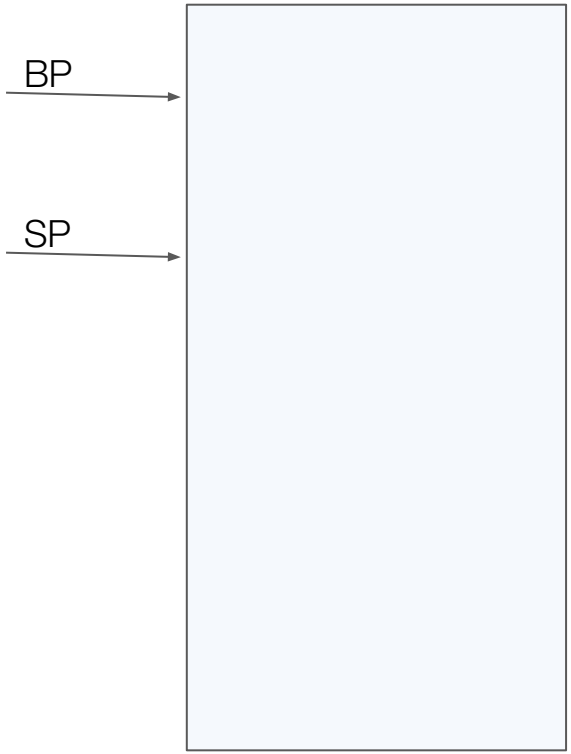
undo

Backtrace

```
0000119d <foo>:
   119d:       55              push   %ebp
   119e:       89 e5           mov    %esp,%ebp

          ...

   11ab:       5d              pop    %ebp
   11ac:       c3              ret
```



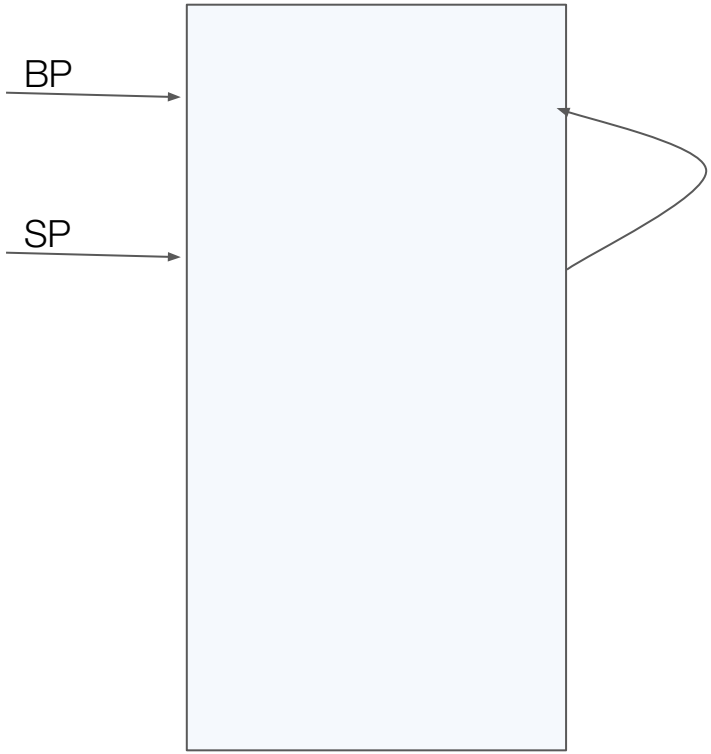
undo

Backtrace

```
0000119d <foo>:
   119d:        55              push   %ebp
   119e:        89 e5           mov    %esp,%ebp

            ...

   11ab:        5d              pop    %ebp
   11ac:        c3              ret
```



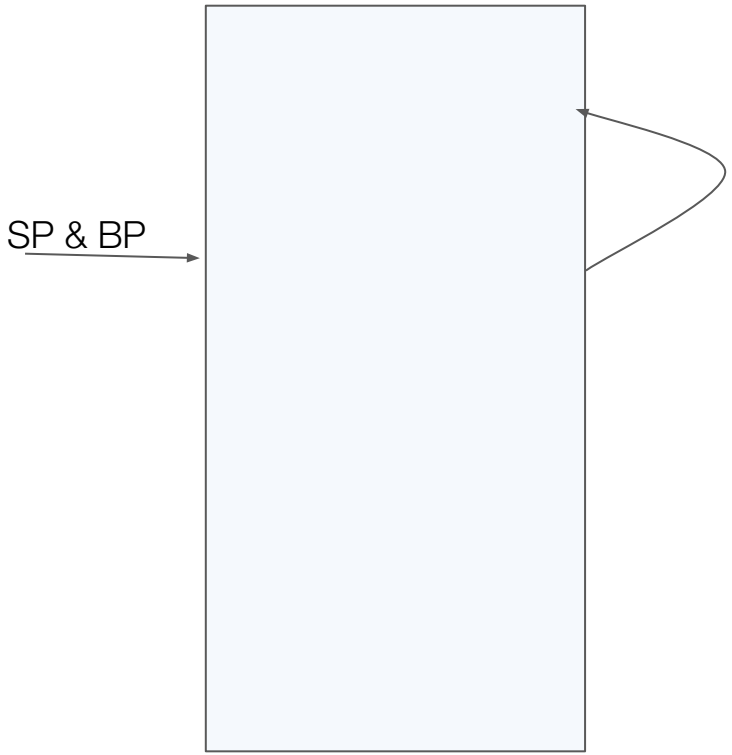
undo

Backtrace

```
0000119d <foo>:
   119d:       55              push   %ebp
   119e:       89 e5           mov    %esp,%ebp

          ...

   11ab:       5d              pop    %ebp
   11ac:       c3              ret
```



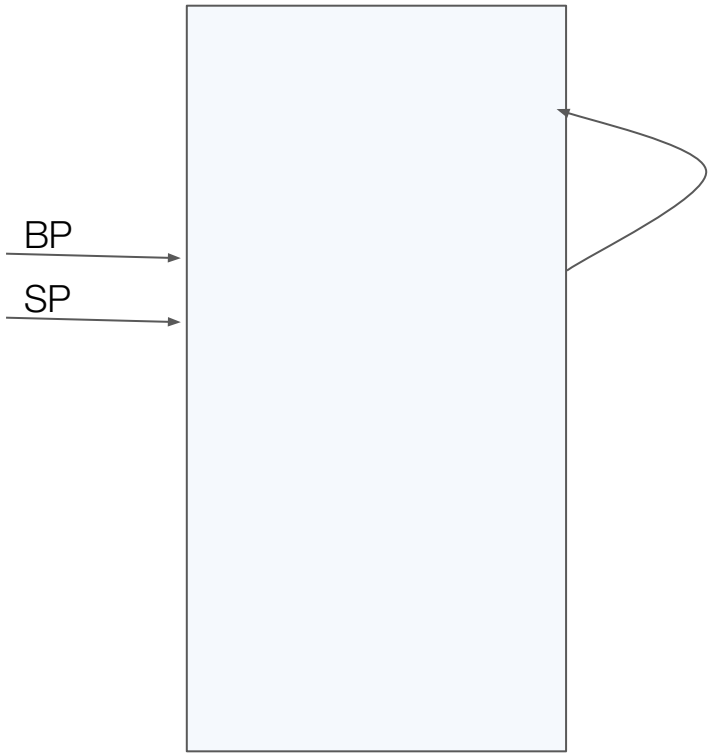
undo

Backtrace

```
0000119d <foo>:
   119d:       55              push   %ebp
   119e:       89 e5           mov    %esp,%ebp

          ...

   11ab:       5d              pop    %ebp
   11ac:       c3              ret
```



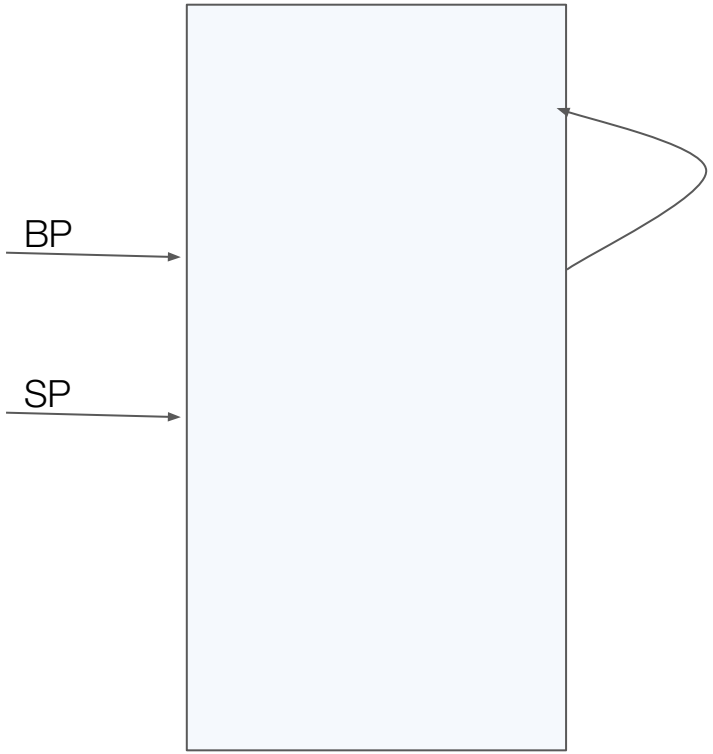
undo

Backtrace

```
0000119d <foo>:
   119d:       55              push   %ebp
   119e:       89 e5           mov    %esp,%ebp

          ...

   11ab:       5d              pop    %ebp
   11ac:       c3              ret
```



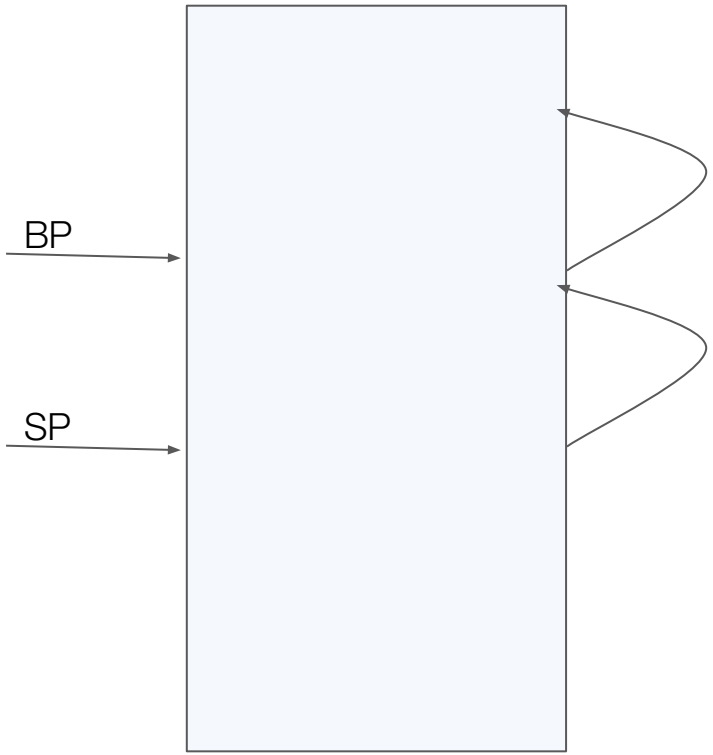
undo

Backtrace

```
0000119d <foo>:
   119d:        55              push   %ebp
   119e:        89 e5           mov    %esp,%ebp

          ...

   11ab:        5d              pop    %ebp
   11ac:        c3              ret
```



undo

But the compiler *knows*!

CFI - Call Frame Instructions

Can see this in the assembly generated by gcc

CFI is in both the `.debug_frame` and the `.eh_frame` sections.

gcc usually emits only `.eh_frame` (mandatory on x86-64)

- unless you say `-fno-asynchronous-unwind-tables`

undo

@gregthelaw

undo