

GDB stub

Xavier Deguillard
—
xavier@lse.epita.fr

GDB stub

Xavier Deguillard – xavier@lse.epita.fr

LSE

9 décembre 2011

GDB stub

Xavier Deguillard

—

xavier@lse.epita.fr

- Permet de debugger une cible distante, avec collaboration de celle-ci.
- Le kernel sait qu'il est debuggé.
- Le stub est le bout de code kernel qui communique avec le gdb client.


Why ?

GDB stub

Xavier Deguillard

—
xavier@lse.epita.fr

```
IRQ 3 -> 16 from bus pci
IRQ 1 -> 1 from bus isa
IRQ 3 -> 3 from bus isa
IRQ 4 -> 4 from bus isa
IRQ 6 -> 6 from bus isa
IRQ 7 -> 7 from bus isa
IRQ 8 -> 8 from bus isa
IRQ 9 -> 9 from bus isa
IRQ 12 -> 12 from bus isa
IRQ 13 -> 13 from bus isa
IRQ 14 -> 14 from bus isa
IRQ 15 -> 15 from bus isa
LINT1: NMI 0
Kernel panic: Assert : ../include/kernel/list.h:31: elm-
#0: C010148D
#1: C010B817
#2: C010BA5E
#3: C0107ECE
#4: C0100F9B
#5: C0100F7A
#6: C010105C
#7: C0100463
```



GDB stub

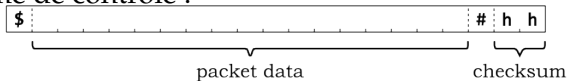
Xavier Deguillard

—

xavier@lse.epita.fr

Le protocole GDB ne spécifie pas la couche physique qui doit être utilisée pour communiquer entre le client et le stub. Dans le cas du debugging kernel, le serial est souvent préféré pour sa simplicité.

Il s'agit d'un protocole en mode texte, avec acquittement et somme de controle :



L'acquittement peut soit être positif (envoi du caractère '+'), soit négatif (envoi du caractère '-').

Le type du paquet est déterminé par la première lettre qui le compose.

- '?' : Raison de l'arrêt,
- 'q' : Requêtes particulières,
- 'H' : Opérations sur les threads,
- 'g' : État des registres,
- 'G' : Modification des registres,
- 'm' : Lecture mémoire,
- 'M' : modification mémoire,
- 's' : step d'une instruction,
- 'c' : continue

En règle générale, le stub est en attente d'une connexion, soit parce que le kernel est dans un état irrécupérable (kernel panic), soit parce qu'il veut être debuggé lors du démarrage du kernel. Du côté stub, cela signifie exécuter l'instruction de breakpoint (int3 sous x86).

Du côté client :

```
(gdb) target remote /dev/ttyUSB0
```

Le client va alors chercher à savoir ce qui est supporté par le serveur, comme la taille maximale d'un paquet ainsi que l'état des registres pour déterminer la position dans le code.

L'étape suivante est bien souvent de poser un breakpoint :

```
(gdb) break my_buggy_function
```

Avant d'indiquer au stub ou poser un breakpoint, le client doit avant tout déterminer l'adresse de celui-ci. Pour cela, le client doit désassembler le code, il envoie pour cela des paquets 'm' au serveur.

Le breakpoint n'est ensuite pas directement posé, il le sera lors d'une reprise d'exécution.

(gdb) continue

Les paquets envoyés sont :

- 'G' : Modification de l'EIP courant pour exécuter l'instruction remplacée par le breakpoint :,
- 's' : Step d'une instruction,
- 'Z' : Pose les breakpoints,
- 'c' : Continue,
- '?' : Attente d'une réponse du serveur.

Ce qui ne peut être debuggé

GDB stub

Xavier Deguillard

—

xavier@lse.epita.fr

- La gestion des interruptions,
- Le serial,
- Le stub en lui-même.

GDB stub

Xavier Deguillard

—

xavier@lse.epita.fr

[http://www.embecosm.com/appnotes/ean4/
embecosm-howto-rsp-server-ean4-issue-2.html](http://www.embecosm.com/appnotes/ean4/embecosm-howto-rsp-server-ean4-issue-2.html)

GDB stub

Xavier Deguillard

—

xavier@lse.epita.fr

Des questions ?