UPMC
SORBONNE UNIVERSITÉS

NMV - NI436

TP 04 – Débogage du noyau Linux

Maxime Lorrillere et Julien Sopena

Octobre 2015

Le but de ce TP est de se familiariser avec l'outil de débogage *KGDB*. À travers plusieurs exemples de bogues, on étudiera ses différentes fonctionnalités. On reviendra aussi sur la bonne utilisation des listes dans le noyau.

Mise en garde : ce TP reprend l'environnement de programmation mis en place lors du deuxième TP. Il suppose entre autre l'utilisation dans qemu de l'image *nmv-archlinux.img*, ainsi que l'existence d'un fichier personnel *myHome.img* correspondant au `/root`. Il suppose aussi que vous ayez un fichier de configuration pour le noyau linux 4.2.3 et que vous ayez dans `/tmp` un exemplaire des sources.

Exercice 1 : Connection du débogueur KGDB

Le noyau Linux possède un débogueur qui est accessible par l'intermédiaire de plusieurs interfaces différentes. L'une d'entre elles, *KGDB*, permet de déboguer une machine cible à partir d'une machine hôte en utilisant GDB.

Dans ce TP, vous utiliserez KGDB dans votre machine hôte (celle qui vous a servi à compiler votre noyau), pour déboguer votre machine virtuelle.

Question 1

À l'aide de l'outil de configuration du noyau que vous avez manipulé lors du deuxième TP (`make nconfig`), vérifiez si **KGDB** est activé dans votre noyau et activez le si tel n'est pas le cas. **Indice :** la plupart des options que nous allons manipuler aujourd'hui sont dans la section *Kernel Hacking*.

Question 2

Lors d'un *crash* du noyau, **KGDB** se lance dans la VM et il est alors possible de s'y connecter à l'aide de GDB. Pour cela, il est nécessaire d'indiquer un port série à KGDB qu'il utilisera pour communiquer avec GDB. De plus, ce port série doit être accessible depuis la machine hôte, ce qui est rendu possible par QEMU, qui permet de représenter un port série de différentes façons (fichier, PTY, socket, etc.).

Ouvrez le script qui vous permet de lancer des machines virtuelles (qemu-run-externKernel.sh). Quel moyen de communication avons nous choisi pour connecter GDB (machine hôte) à KGDB (machine virtuelle) ?

Question 3

L'option de démarrage du noyau `kgdbwait` permet de bloquer votre VM dès le démarrage pour attendre qu'un débogueur s'y connecte. Ajoutez cette option de démarrage au noyau de votre VM, puis lancez GDB sur votre machine hôte en utilisant le noyau compilé (`vmlinux`). Connectez le à votre machine virtuelle grâce à la commande **target remote** en tenant compte du moyen de communication trouvé à la question précédente.

Question 4

Dans **KGDB**, quelle est la différence entre la commande **info thread** et la commande **monitor ps** ? Vous pouvez vous aider des commandes **help** (ex : *help thread info*) et **monitor help**.

Question 5

Que fait la commande **continue** ?

Question 6

Pour donner la main à **KGDB** sans attendre que le noyau *crash*, il possible d'envoyer une commande particulière au noyau (*SysRq*). Pour cela, il suffit d'écrire 'g' dans le fichier /proc/sysrq-trigger de votre machine virtuelle pour l'interrompre :

```
echo g > /proc/sysrq-trigger
```

Exercice 2 : Prise en main de KGDB

Question 1

La variable globale **init_uts_ns** est utilisée dans le noyau pour stocker les informations retournée par l'appel système **uname**. Dans les sources du noyau, trouvez où est initialisée cette structure et analysez les champs qu'elle contient.

Question 2

En utilisant les commandes **print** et **set variable** de **KGDB**, affichez le contenu de cette structure et modifiez la pour que **uname -r** renvoie un autre nom.

Exercice 3 : Mon premier bug

Dans la suite de ce TP, nous vous fournissons 2 modules noyau, volontairement buggés. Le but n'est pas ici de trouver les erreurs par vous même en analysant le code mais à l'aide de **KGDB**.

Question 1

Observez les sources du module **hanging** et expliquez son fonctionnement ?

Question 2

Compilez le module **hanging** pour votre noyau. Pour cela, vous procéderez comme pour les TP précédents en définissant la variable d'environnement `KERNELDIR` à la valeur du chemin contenant les sources de votre noyau (**/tmp/linux-4.2.3**).

Copiez le module dans votre machine virtuelle en utilisant SSH et chargez le. Observez bien les logs et attendez quelques secondes (~1mn) : votre noyau crash-t-il ?

En analysant les premières lignes des messages que vous observez, pouvez-vous expliquer ce qu'il s'est passé ?

Question 3

Pour interrompre votre noyau et donner la main à **KGDB** au moment où apparaît ce message, il faut ajouter une option de compilation à votre noyau qui le crashera (*panic*) lorsque ce message apparaît.

Modifiez la configuration du noyau pour activer cette option. Recompilez le noyau, et relancez la machine virtuelle pour prendre la main avec **KGDB** au moment où apparaît ce message.

Question 4

Dans **KGDB**, affichez la pile d'appels (**backtrace**) au moment du crash. Est-ce que cela correspond au code du module que vous avez observé ?

Question 5

Utilisez les commandes **monitor ps** et **monitor btp** pour afficher correctement la pile d'appels du module au moment du bug.

Question 6

Dans **KGDB**, vérifiez que votre module est bien chargé à l'aide la commande **monitor lsmod**. La première adresse affichée correspond à l'adresse mémoire de la `struct module` du module. À quoi correspond la seconde adresse ?

Question 7

Comment corriger ce bug ? Proposez une solution qui ne modifie que le module, et une solution qui ne modifie que la configuration du noyau.

Exercice 4 : Affichages dynamiques

Le noyau dispose d'une fonctionnalité appelée *dynamic debug*, qui permet d'activer/de désactiver à la volée les affichages de débogage du noyau (`pr_debug`). C'est une approche idéale lorsque vous souhaiter intégrer beaucoup d'affichages de débogage dans votre code en limitant leur impact sur les performances.
Documentation : `Documentation/dynamic-debug-howto.txt`

Question 1

Le module **prdebug** affiche périodiquement le nombre d'interruptions reçues par le processeur 0.

Observez les sources de ce module et essayez de comprendre son fonctionnement. Compilez le et testez le dans votre VM. Pourquoi aucun affichage n'apparaît dans les logs du noyau ?

Question 2

Le fichier `/sys/kernel/debug/dynamic_debug/control` permet de configurer le fonctionnement des affichages dynamiques.

Chargez le module **prdebug** dans la VM. Pour activer tous les affichages de débogage générés par le module **prdebug**, utilisez la commande suivante :

```
echo -n 'module prdebug +p' > /sys/kernel/debug/dynamic_debug/control
```

En vous aidant de la documentation du *dynamic debug*, expliquez la composition de cette commande.

Question 3

Il est possible d'ajouter dynamiquement certaines informations, telles que le nom du module, de la fonction ou même le numéro de la ligne où est présent le printk.

Toujours pour le module **prdebug**, activez les affichages du nom du module, du nom des fonctions et des numéros de lignes.

Question 4

Il est possible de composer les différentes règles de correspondance (module, fonction, fichier, ligne, etc.) d'une commande de configuration pour afficher, par exemple, un message d'un numéro de ligne de code particulière d'un fichier (`.c`) particulier.

Déchargez puis recharger le module **prdebug**. Observez que la configuration du *dynamic debug* a été réinitialisée. En vous aidant de sa documentation, activez **uniquement** l'affichage du message de débogage qui affiche le nombre de d'interruptions écoulées.

Exercice 5 : Débogage d'un module

Dans cet exercice plusieurs bugs ont été introduits dans le module **kcputstat**. L'objectif de cet exercice est de détecter, analyser et corriger ces bugs à l'aide des options de débogage du noyau et de **KGDB**.

Attention : ne pas modifier les sources du module si ce n'est pas explicitement demandé dans l'énoncé.

Question 1

Le module **kcpustat** collecte les statistiques d'utilisation CPU (comme la commande *time*) et les affiche dans la console du noyau.

Prenez votre temps, observez bien les sources du module et assurez vous d'avoir compris son fonctionnement.

Question 2

Compilez le module **kcpustat** pour votre noyau, copiez le dans votre VM et chargez le. Attendez quelques secondes : que ce passe-t-il ?

Question 3

Connectez vous à votre machine virtuelle avec **KGDB**. Observez que le résultat des commandes **backtrace** et **monitor bt** n'est pas exploitable.

À l'aide des commandes **monitor lsmod** et **add-symbol-file**, chargez les symboles de votre module (le fichier *.o*) en indiquant à **KGDB** l'adresse chargement du module.

Question 4

Quelle ligne de code du module **kcpustat** pose problème ? Proposez une correction qui ne supprime pas la ligne en question.

Question 5

Testez de nouveau le module **kcpustat**. Que ce passe-t-il ? Connectez vous à votre machine virtuelle avec **KGDB** de la même façon qu'à la question précédente. De quelle structure de donnée vient le problème ?

Question 6

Le noyau dispose d'options de débogage pour certaines structures de données, qui peuvent être activées à la compilation. En modifiant la configuration du noyau, activez l'option de débogage dont vous avez besoin. Recompilez votre noyau, le module **kcpustat**, et testez de nouveau.

Vous devriez avoir de nouvelles informations sur la console du noyau. Qu'en déduisez vous ? Proposez une correction pour ce bug.

Question 7

Le module **kcpustat** ne gère pas correctement la mémoire. Le noyau dispose d'un outil permettant de tracer les fuites mémoire. En modifiant la configuration du noyau, activez cet outil. Familiarisez vous sur son fonctionnement en regardant sa documentation disponible dans le répertoire Documentation.

Question 8

Utilisez l'outil de détection de fuites mémoire pour détecter les fuites provoquées par le module **kcpustat**. D'où vient le problème ? Proposez une correction.