

Debugging Système et Noyau
-
Travaux Pratiques

Aurélien Cedeyn

2018-2019

Préparation

- Vous disposez d'une machine virtuelle *pcocc* pour suivre ce TP.
- Template *pcocc* : debug
- Utilisateurs :
 - votre login
- Configuration *pcocc*

```
_____ $HOME/.pcocc/cloud-user-data/debug _____  
#cloud-config
```

users:

```
- name : <login>  
  sudo: ['ALL=(ALL) NOPASSWD:ALL']  
  ssh-authorized-keys:  
    - <clé publique ssh>  
- name : root  
  ssh-authorized-keys:  
    - <clé publique ssh>
```

- Configuration de la machine virtuelle
 - Système : CentOS-7.5
- Ce TP est noté, il vous est demandé de rendre ce que vous avez pu réaliser à la fin de celui-ci.
- Vous avez jusqu'au vendredi 14/12 23h59 pour envoyer le compte rendu complet du TP.
- Ces deux échéances constitueront votre note de TP.
- Les différentes réponses avec les sorties de vos commandes devront suivre le format suivant :
 - Un répertoire à vos nom et prénoms.
 - Un un fichier par question.

```
_____ Format des réponses _____  
$ ls aurelien.cedeyn/  
1.1.txt 1.2.txt 1.3.txt 1.4.txt  
2.1.txt 3.1.txt 3.2.txt 3.3.txt 3.4.txt
```

User space

1 L'espace utilisateur

1. Listez les processus de votre utilisateur.
2. Affichez les fichiers lus par la commande *ps*.
3. Affichez le nombre d'appels systèmes effectués par la commande *ps -elf*
4. La commande *lsof*.
 - Utilisez un des outils vu précédemment en cours pour voir les appels systèmes effectués par cette commande.
 - Quels fichiers ouvre-t-elle ?
 - Quels sont les appels aux bibliothèques externes effectués cette commande ?
 - Quelle est la fonction la plus appelée par *lsof* ?
 - À quoi sert-elle ?

2 La pile

1. Écrire un programme en C qui dépasse la taille de la pile.
 - **Indice :** *ulimit -a* permet de connaître les restrictions du système.
 - Quelles sont les différentes façons, selon vous, de dépasser la taille de la pile ?
 - Quelle erreur obtenez-vous ? Que signifie-t-elle ?
 - Comment corriger le programme ou l'environnement pour ne plus avoir cette erreur ?

3 La compilation/gdb

1. Compilez avec et sans les symboles de debug le programme C suivant :
Indice : Pour compiler avec les symboles de debug : `gcc -g source.c -o binaire`

```
infinite.c
#include <stdlib.h>
#include <unistd.h>

int check(char cond){
    return(cond == 0);
}

void loop(void){
    int a=0;

    while(check(a)){
        usleep(1000);
    }
}

int main(void) {
    loop();
    exit(0);
}
```

- Quelles différences observez vous entre les deux binaires ?
 - Observez les symboles de debug avec la commande `readelf`.
Indice : `man readelf`
2. Lancez le programme compilé avec les symboles de debug via `gdb`.
 - Affichez le code source dans `gdb`.
 - Débutez son exécution.
 - Interrompez-le et affichez sa pile d'appel.
 - Quittez `gdb`.
 3. Choisissez un processus sur la machine et prenez en un corefile.
 - Lancez `gdb` avec le corefile généré.
 - Affichez la pile d'appel du processus.
 - Quittez `gdb`.
 4. Attachez-vous au processus `crazy` qui tourne sur la machine avec `gdb`.
 - Affichez le code source dans `gdb`.
 - Placez un point d'arrêt (breakpoint) sur à la ligne 12 de la fonction `main`.

- Continuez le programme.
- Affichez la pile d'appel.
- Affichez la valeur de la variable *count*.
- Modifiez la pour que le programme se finisse.