
Rapport de Master M1

**Abdeljalil Bouchelga , Aicha Maaoui**

Calcul Haute Performance Simulation (CHPS)

# Implémentation d'un Debugger en C

Mars 2022

Institut des Sciences et Techniques des Yvelines (ISTY)

# Contents

# Chapter 1

# Architecture De base

## 1.1 L'objectif du projet

Notre objectif est de réaliser un outil qui permet de démarrer un processus et le déboguer, ou s'attacher à un processus existant et parcourir le code en une seule étape, définir des points d'arrêt et les exécuter, examiner les valeurs des variables et empiler les traces.

## 1.2 l'architecture de base du programme

### 1.2.1 Ptrace

Ce projet est principalement basé sur l'appel système appelé ptrace. Il s'agit d'un appel système unique avec une multitude de types de requêtes différents. Celui-ci permet au traceur (debugger ou tracer) de suivre un observé (programme à déboguer ou tracee). Ainsi, le traceur peut changer la mémoire de l'observé, changer ses registres, tracer ses appels système ou mettre des points d'arrêt.

Les requêtes de ptrace utilisés, *"ptrace(2)"* 2021 & *"Playing with ptrace, Part I"* 2002:

- **PTRACE_CONT:** Requête qui demande que l'inférieur continue l'exécution,
- **PTRACE_TRACEME:** Processus enfant demande au noyau du système d'exploitation de laisser son parent le tracer,
- **PTRACE_GETSIGINFO:** Récupérer des renseignements sur le signal qui a provoqué l'arrêt,

- **PTRACE_PEEKTEXT:** Lire un mot à l'adresse addr dans la mémoire du tracee, renvoyant le mot comme résultat de l'appel à ptrace (),
- **PTRACE_GETREGS:** Copier les registres généraux ou du processeur en virgule flottante du tracee, vers l'adresse data du tracer,
- **PTRACE_SETREGS:** Modifier les registres généraux ou du processeur en virgule flottante du tracee, à partir des données d'adresse dans le traceur.

## 1.3 L'attende du tracee

### 1.3.1 waitpid

Nous avons utilisé l'appel systéme waitpid pour notifié le tracer sur l'état du tracee, Nous passrons l'identifiant du processus que nous voulons attendre à travers le pid. status est un paramètre de sortie qui encode la raison pour laquelle le trace s'est arrêté. Pour décoder la raison, nous utilisons des macros pour surveiller nos tracee.

### 1.3.2 Backtrace

Un backtrace est un suivi des fonctions appelées avant la fonction courante à travers la pile d'appels. Ainsi, pour obtenir un backtrace, il faut parcourir la pile jusqu'à ce que ptrace renvoie -1. L'appel backtrace() remplit un tableau avec le compteur de programme de chaque fonction appelante, et l'appel séparé backtrace_symbols() peut rechercher les noms symboliques pour chaque adresse. La sortie affiche la trace avec l'adresse de chaque site d'appel de fonction, et nous avons ajouté l'option -rdynamic pour obtenir des noms symboliques utiles, *"gcc backtrace support"* 2009.

# Chapter 2

# Description du Debugger

## 2.1 Analyseur du Programme

### 2.1.1 Bibliothèques Partagées

Les bibliothèques partagées sont des bibliothèques chargées par le programme lors du démarrage, *"Dynamically Loaded (DL) Libraries"* 2003.

**Affichage en utilisant SHELL:**

L'affichage des dépendances de la bibliothèque partagées (objets partagés) est possible à l'aide du fichier *"shared_lib.sh"*, où on utilise, *"ldd(1)"* 2021:

- ***ldd*:** affiche l'ensemble de dépendance de l'exécutable,
- ***"objdump -p /path/to/program — grep NEEDED"*:** affiche uniquement les dépendances directes de l'exécutable.

**Affichage en utilisant le code C:**

L'affichage des chemin d'accès des bibliothèque partagées (objets partagés) est aussi possible à l'aide du fichier *"showObject.c"*, où on utilise la fonction *dl_iterate_phdr()*, *"dl_iterate_phdr(3)"* 2021. Les adresses virtuelles auxquelles les segments ELF de l'objet sont chargés sont aussi données pour chaque objet partagé.

### 2.1.2 Variables d'environment

Les variables d'environnement font partie de l'environnement dans lequel un processus s'exécute. Le programme en C *envp.c* permet de retourner le contenu des variables d'environnement en utilisant la fonction *getenv()*. La variable d'environnement retournée dans ce projet est *PATH*.

### 2.1.3 Affichage des noms de fonctions du programme exécuté

L'affichage des noms de fonctions d'un programme executé est fait par l'intermédiaire du code en c *displayNameFunction*.

### 2.1.4 Affichage des registres du programme exécuté

L'affichage des registres d'un programme executé est fait par l'intermédiaire du code en shell *registers.sh*. Pour cela, on utilise *objdump* et la flag associée *-d* qui permet de démonter les sections d'un program donné pour afficher des informations sur des fichiers objets.

## 2.2 Les points d'arrêt (Breakpoints)

### 2.2.1 Principe

Les points d'arrêt sont importants dans l'implémentation d'un Debugger. Ils sont bases sur l'interruption logicielle (" *Traps*"); i.e. la suspension temporaire de l'exécution d'un programme par un CPU, *"Interruption Logicielle"* 2022.

Lors de la suspension, l'état de l'exécution est enregistré. Le CPU se dirige vers l'addresse où se trouve l'interruption, après il reprend l'exécution.

### 2.2.2 Implémentation des points d'arrêt

*INT* est une instruction en langage assembleur (*INT X*, *X* est l'interruption logicielle) pour les processeurs $x86$ qui génère une interruption logicielle, *"INT (x86 instruction)"* 2022.

#### INT 3

L'instruction *INT 3* est une instruction d'octet définie et utilisée par les débogueurs, *"INT (x86 instruction)"* 2022. Elle permet de remplacer temporairement une instruction dans un programme en cours d'exécution afin de définir un point d'arrêt de code.

### Opcode de INT 3

L'opcode (code opération) est une partie d'une instruction en langage machine qui spécifie l'opération à effectuer. L'opcode pour *INT 3* est $0xCC$.

### Point d'accès (Entry Point)

Le point d'entrée est un point où l'exécution d'un programme commence et où le programme a accès aux arguments de la ligne de commande, *"Entry point"* 2022. L'addresse du point d'entrée est obtenue en exécutant la commande: "*readelf -h program*".

### Définition des points d'arrêt avec *INT 3*

Les étapes de définition des points d'arrêt pour un debugger sont les suivantes:

- Cherche l'adresse cible du data où on va mettre les points d'arrêt,
- Remplacez le premier octet de l'adresse cible par l'instruction *INT 3*,
- Le débogueur demande au système d'exploitation d'exécuter le processus, qui trouve l'instruction *INT 3* où il va s'arrêter,
- Le débogueur reçoit le signal indiquant l'arrêt du traced (l'enfant),
- Le debogueur remplace *INT 3* par l'instruction d'origine dans l'adresse cible,
- Le pointeur d'instruction de l'enfant est exécuté, alors on diminue sa position de $-1$.

# References


*"dl_iterate_phdr(3)"* (2021). Linux manual page. URL: https://linux.die.net/man/3/dl_iterate_phdr.

*"Dynamically Loaded (DL) Libraries"* (2003). Program Library HOWTO. URL: https://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html.

*"Entry point"* (2022). Wikipedia. URL: https://en.wikipedia.org/wiki/Entry_point.

*"gcc backtrace support"* (2009). codingrelic. URL: https://codingrelic.geekhold.com/2009/05/pre-mortem-backtracing.html.

*"INT (x86 instruction)"* (2022). Wikipedia. URL: https://en.wikipedia.org/wiki/INT_(x86_instruction).

*"Interruption Logicielle"* (2022). Wikipedia. URL: https://fr.wikipedia.org/wiki/Interruption_(informatique).

*"ldd(1)"* (2021). Linux manual page. URL: https://man7.org/linux/man-pages/man1/ldd.1.html.

*"Playing with ptrace, Part I"* (2002). Linux Journal. URL: https://www.linuxjournal.com/article/6100?page=0,1l.

*"ptrace(2)"* (2021). Linux manual page. URL: https://man7.org/linux/man-pages/man2/ptrace.2.html.

# Appendix A

# Appendix: GitHub

Le projet est déposé dans le *GitHub*. La clé SSH est la suivante:

**git@github.com:Chaichas/Debugger_AISE.git**